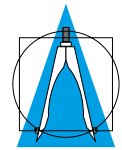


ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Java для IT-профессий

Учебник

ИЗДАТЕЛЬСТВО FOLIANT

Нур-Султан-2019

УДК 004
ББК 32.816
Х 29

Автор:
Харди Дирк

Редактор оригинального издания:
Барт Александер

Рецензенты
Шангытбаева Г.А. – *PhD доктор, Актюбинский региональный государственный университет имени К. Жубанова*
Абдрахманова С.В. – *преподаватель спецдисциплин первой категории КГУ «Карагандинский профессионально-технический колледж»*
Файзрахманова Р.Р. – *преподаватель-практик КГУ «Карагандинский профессионально-технический колледж»*

Х 29 **Java для IT-профессий:** Учебник / Пер. с немецкого. – Нур-Султан: Фолиант, 2019. – 280 с.

ISBN 978-601-338-401-6

«Java для IT-профессий» – учебник по одному из самых популярных в мире языков программирования. Благодаря разделению на три части – теоретическую, практическую и проектную – обучающиеся могут не только изучить основы, но и применить знания на практике.

Первая часть книги – теоретическая. Здесь изложены принципы функционирования структур Java, концепция объектно-ориентированного программирования, работа с БД, разработка приложений Android. Рассматривается GUI-программирование с AWT, GUI-конструктор NetBeans, принципы создания апплетов.

Вторая часть книги содержит практические задания разных уровней сложности. Большое количество примеров настоящего кода позволяет освоить на практике теоретический блок.

В третьей части представлены учебные ситуации, прочно связанные с изученными темами. В данном разделе автором предложена не только разработка кода, но и контроль выполнения этой работы.

Учебник предназначен для студентов учебных заведений технического и профессионального образования, обучающихся по специальности 1300000 «Связь, телекоммуникация и информационные технологии», а также для начинающих и более опытных программистов.

УДК 004
ББК 32.816

ISBN 978-601-338-401-6

Предисловие

Программная платформа Java была разработана в начале 90-х годов, чтобы получить отдельную систему из современного языка программирования и системного окружения. Для этого было необходимо кроссплатформенное программирование, так как на любой платформе (даже у кофе-машины) требовалось лишь наличие системного окружения.

Прорыв технологии Java произошел в связи с развитием интернета в конце 90-х. Благодаря Java и соответствующим техникам (таким как апплеты) веб-программирование сильно продвинулось вперед. Сегодня программы Java используются во всех сферах: не только в веб-программировании, но и в качестве приложений для ПК, в мобильных вычислениях *(Mobile Computing)* или в качестве встроенных систем (*Embedded Systems*).

Поэтому работа с Java включает в себя не только изучение объектно-ориентированного языка программирования, но и углубленный разбор программной платформы Java – это очень важный аспект для обучающихся **IT**-профессиям**.**

Структура книги

Цель данной книги – представление языка **Java** максимально наглядно, с ориентацией на практику и обучение. Поэтому книга написана с практическим подходом. Автор считает, что именно в школьном образовании необходимо усиленно подготавливать учеников к сложным темам программирования путем наглядного и практического обучения. После этого общие аспекты программирования или даже разработки программного обеспечения можно лучше понимать и применять на практике.

Книга состоит из трех частей.

Первая часть книги представляет собой информационный блок и предлагает системное введение в язык **Java** и основы программной платформы **Java**. Введение в GUI-программирование, соединение с базами данных и разработка приложений **Android** (новая часть во 2-м издании) завершают данный информационный блок.

Вторая часть книги – это сборник практических заданий. После получения соответствующих навыков из информационного блока задания из этой части могут подготовить к дальнейшему разбору тем, а разные уровни сложности заданий позволят провести обширную работу на занятиях.

В третьей части книги содержатся ситуации, основанные на учебном модуле «Разработка и развертывание систем приложений» из базисного учебного плана для IT-профессий (специальная разработка приложений специалистами по программированию). Ситуации составлены по учебным модулям и в идеале должны представлять комплексные действия (планирование, проведение, контроль). По этой причине учебные ситуации составлены таким образом, что наряду с фазой планирования в центре внимания стоит не только выполнение программы, но и соответствующие методы тестирования для контроля программы и процесса разработки. Учебные ситуации можно также рассматривать как идеи для проекта.

Книга подходит для всех профильных курсов подготовки в сфере IT. Благодаря дифференцированным проектным заданиям она может быть использована во всех IT-профессиях (особенно специалистами по информатике), а также информационно-техническими ассистентами.

В качестве средства разработки в данной книге используется **NetBeans IDE 8.0.1**. Эта среда разработки доступна для бесплатного скачивания в сети Интернет.

Мы будем благодарны за предложения и критику к данной книге (возможно по E-Mail).

Дирк Харди
Летом 2015 г.
E-Mail: Hardy@DirkHardy.de
Издательство «Europa-Lehrmittel»
E-Mail: Info@Europa-Lehrmittel.de

Содержание

Предисловие ... 3

Структура книги ... 3

Часть 1 Введение в Java ... 11

1 Введение в технологию Java ... 13
1.1 Технология Java ... 13
1.1.1 Возникновение технологии Java ... 13
1.1.2 Характеристики технологии Java ... 14
1.1.3 Компоненты технологии Java ... 14
1.1.4 Компиляция программ Java ... 14
1.2 Язык Java ... 15
1.2.1 Развитие языка Java ... 15
1.2.2 Характеристики языка Java ... 15
1.2.3 Ключевые слова в Java ... 16
1.2.4 Процедурное, структурированное и объектно-ориентированное программирование Java ... 16
1.2.5 Составляющие программы Java ... 17

2 Первая программа Java ... 19
2.1 Создание проекта Java ... 19
2.2 Первая программа Java ... 22
2.2.1 Основная структура Java ... 22
2.2.2 Пакеты ... 22
2.2.3 Класс «Hello, World!» и главный метод main ... 23
2.2.4 Вывод данных на экран ... 24
2.2.5 Важные правила программы Java ... 25
2.3 Основополагающие конвенции в Java ... 25
2.3.1 Идентификатор (имя) в Java ... 25
2.3.2 Разделительный знак ... 26
2.3.3 Комментарии в Java ... 27
2.4 Типы данных и переменные ... 29
2.4.1 Переменные в Java ... 29
2.4.2 Простые типы данных ... 29
2.4.3 Объявление переменных ... 30
2.4.4 Операции с простыми типами данных ... 31
2.4.5 Постоянные переменные ... 32

3 Ввод и вывод в Java ... 33
3.1 Система вывода в Java ... 33
3.1.1 Вывод переменных ... 33
3.2 Ввод с консоли ... 35
3.2.1 Ввод цепи символов ... 35
3.2.2 Конвертация ввода ... 35

4 Операторы в Java 38
4.1 Арифметические операторы 38
4.1.1 Простые типы данных и их арифметические операторы 38
4.1.2 Оператор модуля 39
4.1.3 Операторы инкремента и декремента 39
4.2 Реляционные и логические операторы 40
4.2.1 Реляционные операторы 40
4.2.2 Логические операторы 41
4.3 Битовые и другие операторы 42
4.3.1 Логические битовые операторы 42
4.3.2 Битовые операции сдвига 43
4.3.3 Преобразование типов оператором CAST 43
4.3.4 Присваивание и связанное присваивание 44
4.4 Ранг оператора 45

5 Селекция и итерация 47
5.1 Селекция 47
5.1.1 Представление выбора с помощью блок-схемы программы 47
5.1.2 Одиночный выбор с помощью оператора if 48
5.1.3 Двойной выбор с помощью оператора if-else 48
5.1.4 Ветвление с помощью операторов if и if-else 50
5.1.5 Множественный выбор с помощью оператора switch 51
5.2 Итерационные циклы с постусловием и предусловием, циклы со счетчиком 54
5.2.1 Цикл do-while 54
5.2.2 Цикл while 56
5.2.3 Цикл for 57
5.2.4 Досрочный выход и пропуск итерации 59

6 Понятие классов в Java 60
6.1 Первый класс в Java 62
6.1.1 Структура класса Java 62
6.1.2 Типы значений и ссылок 64
6.2 Методы в Java 64
6.2.1 Структура метода 64
6.2.2 Возвращаемое значение метода 66
6.2.3 Локальные переменные 67
6.2.4 Передаваемые параметры метода 68
6.2.5 Перегрузка методов 72
6.2.6 Дополнительная информация о методах 73
6.3 Другие элементы классов 74
6.3.1 Конструкторы и деструктор 74
6.3.2 Ссылка this 78
6.3.3 Статические элементы класса 79
6.3.4 Константные элементы класса 80
6.4 Перечисляемые типы 80
6.4.1 Простые перечисления 80
6.4.2 Классы перечислений 81

7 Наследование в Java 83
7.1 Наследование в Java 83
7.1.1 Простое наследование 83
7.1.2 Применение наследования в Java 84
7.1.3 Доступ к атрибутам 86
7.1.4 Финальные классы 87
7.2 Полиморфизм 87
7.2.1 Класс объект 87

7.2.2 Присваивание в рамках иерархии наследования ... 89
7.2.3 Перезапись методов ... 90
7.3 Абстрактные базовые классы ... 93
7.3.1 Абстрактный базовый класс ... 93
7.4 Интерфейсы в Java ... 94
7.4.1 Структура интерфейса ... 94

8 Массивы в Java ... 97
8.1 Одномерные и многомерные массивы ... 97
8.1.1 Одномерные массивы ... 97
8.1.2 Цикл for each ... 99
8.1.3 Многомерные массивы ... 101
8.1.4 Копирование массивов ... 104
8.1.5 Массивы объектов ... 106
8.1.6 Передача массивов методам ... 107
8.2 Сортировка массивов ... 109
8.2.1 Сортировка методом выбора ... 109
8.2.2 Статический метод сортировки sort ... 111
8.2.3 Интерфейс Comparable ... 112
8.3 Особые классы массивов ... 113
8.3.1 Класс ArrayList ... 113
8.3.2 Класс HashMap ... 114

9 Файловые операции в Java ... 117
9.1 Чтение и запись файлов ... 118
9.1.1 Последовательное чтение и запись ... 118
9.1.2 Прямой доступ к файлам ... 120
9.2 Чтение и запись текстовых файлов ... 122
9.2.1 Запись текстовых файлов с помощью PrintWriter ... 122
9.2.2 Чтение текстовых файлов с помощью сканера ... 123
9.3 Сериализация объектов ... 124
9.4 Методы класса File ... 127
9.4.1 Методы класса File ... 127
9.4.2 Ведение папок ... 128

10 Темы Java для продвинутого уровня ... 130
10.1 Исключения – Exceptions ... 130
10.1.1 Исключения и перехват ошибок (try and catch) ... 130
10.1.2 Системные исключения ... 132
10.1.3 Финальный блок ... 134
10.1.4 Вызов исключений ... 135
10.1.5 Создание собственных классов исключений ... 136
10.2 Обобщенное программирование ... 137
10.2.1 Обобщенные методы ... 137
10.2.2 Обобщенные классы ... 138
10.2.3 Использование обобщенных списковых классов ... 139

11 GUI-программирование с помощью AWT ... 141
11.1 GUI-программирование ... 141
11.1.1 История GUI-программирования ... 141
11.1.2 Структура AWT ... 142
11.1.3 Основные понятия GUI-программирования ... 142
11.2 Первая программа GUI ... 143
11.2.1 Использование класса Frame ... 143
11.2.2 Написание собственного класса Frame ... 144
11.3 Вывод текстовой и графической информации ... 145
11.3.1 Paint и первый вывод текста ... 145

11.3.2 Добавление клиентской части ... 147
11.3.3 Простой графический вывод ... 147
11.3.4 Многострочный вывод текста ... 150
11.4 Событийно-ориентированное программирование ... 152
11.4.1 Основа событийно-ориентированного программирования ... 152
11.4.2 Типы событий и приемники событий ... 153

12 Элементы управления с помощью AWT и классов Swing ... 157
12.1 Элементы управления с помощью AWT ... 157
12.1.1 Простые элементы управления ... 157
12.1.2 Использование элементов управления ... 157
12.1.3 Реагирование на события ... 158
12.1.4 Пример приложения с простыми элементами управления ... 159
12.1.5 Упорядочивание элементов управления с помощью менеджера Layout ... 162
12.2 Элементы управления с помощью классов Swing ... 164
12.2.1 Основы классов Swing ... 164
12.2.2 Элементы управления Swing ... 165
12.2.3 Использование простых элементов управления Swing ... 166
12.2.4 Look and Feel ... 169
12.2.4 Look and Feel ... 169
12.3 Более сложные элементы классов Swing ... 169
12.3.1 Дерево JTree ... 169
12.3.2 Создание узлов в JTree ... 170
12.3.3 Обзор важных методов JTree ... 172
12.3.4 Реагирование на события JTree ... 172
12.3.5 Создание таблицы с помощью JTable ... 174
12.3.6 Обзор важных методов JTable ... 175
12.3.7 Реагирование на события JTable ... 176
12.3.8 Снабжение элементов управления полосой прокрутки ... 177

13 Меню, диалоги и апплеты ... 180
13.1 Создание меню с помощью AWT ... 180
13.1.1 Создание меню ... 180
13.1.2 Реагирование на события меню ... 181
13.1.3 Создание контекстного меню ... 182
13.1.4 Создание меню с помощью классов Swing ... 183
13.2 Диалоги ... 185
13.2.1 Использование стандартных диалогов ... 185
13.2.2 Создание собственных диалогов ... 188
13.3 Создание апплетов ... 191
13.3.1 Основы апплетов ... 191
13.3.2 Класс апплетов ... 192
13.3.3 Запуск апплетов ... 193
13.3.4 Элементы управления в апплетах ... 193
13.3.5 Создание апплетов с помощью классов Swing ... 195

14 GUI-конструктор NetBeans ... 197
14.1 GUI-конструктор NetBeans ... 197
14.1.1 Создание формы JFrame ... 197
14.1.2 Добавление элементов управления ... 199
14.1.3 Реагирование на события ... 200
14.2 Интеграция более сложных элементов управления ... 202
14.2.1 Контейнер регистра (панель регистра) ... 202
14.2.2 Панель инструментов (Toolbar) ... 203
14.2.3 Контейнер прокрутки (Scrollpane) ... 203

15 Соединение с базой данных 204
15.1 Доступ к базе данных с помощью Java 204
15.1.1 Соединение с базой данных с помощью JDBC 204
15.1.2 Сброс операций 207
15.2 Доступ к другим базам данных 208
15.2.1 Добавление драйвера 208
15.2.2 Другие драйверы баз данных 209

16 Разработка приложений Android 210
16.1 Основы приложений Android 210
16.1.1 Установка Android SDK 211
16.1.2 Подготовка NetBeans для проектов Android 212
16.2 Разработка Android-Apps 213
16.2.1 Создание проекта Android 213
16.2.2 Элементы прикладной программы Android 215
16.2.3 Адаптация расположения прикладной программы 217
16.2.4 Другие контейнеры расположения и элементы управления 218
16.2.5 Программирование элементов управления 219
16.2.6 Запрос базы данных SQLite 221

Часть 2 Задания 225

Задания 226
1 Задания к главе «Введение в технологию Java» 226
2 Задания к главе «Первая программа Java» 226
3 Задания к главе «Ввод и вывод в Java» 227
4 Задания к главе «Операторы в Java» 228
5 Задания к главе «Селекция и итерация» 230
6 Задания к главе «Понятие классов в Java» 234
7 Задания к главе «Наследование в Java» 237
8 Задания к главе «Массивы в Java» 240
9 Задания к главе «Файловые операции в Java» 245
10 Задания к главе «Темы Java для продвинутого уровня» 251
11 Задания к главе «GUI-программирование с помощью AWT» 253
12 Задания к главе «Элементы управления с помощью AWT или классов Swing» 256
13 Задания к главе «Меню, диалоги и апплеты» 258
14 Задания к главе «GUI-конструктор NetBeans» 259
15 Задания к главе «Соединение с базой данных» 261
16 Задания к главе «Разработка приложений Android» 262

Часть 3 Учебные ситуации 265
Учебная ситуация 1: Презентация с вводной информацией о языке Java (на русском или английском) 266
Учебная ситуация 2: Подготовка клиентской документации для реализации среды разработки в Java (на русском или английском) 267
Учебная ситуация 3: Разработка шифра для внутренней системы памяти отдела поддержки сетевой компании 270
Учебная ситуация 4: Планирование, внедрение и анализ электронной анкеты 270
Учебная ситуация 5: Разработка программного обеспечения для представления метеорологических данных с помощью схемы «Модель – Вид – Контроллер» 270
Учебная ситуация 6: Разработка приложения для решения головоломки «Судоку» 276

Приложение A: Структурированная техника документации 270
Блок-схема программы (PAP): 279
Пример блок-схемы программы: 280

Алфавитный указатель 281

Часть 1

Введение в Java

1.1 Технология Java 13
1.2 Язык Java 15
2.1 Создание проекта Java 19
2.2 Первая программа Java 22
2.3 Основополагающие конвенции в Java 25
2.4 Типы данных и переменные 29
3.1 Вывод в Java 33
3.2 Ввод с консоли 35
4.1 Арифметические операторы 38
4.2 Реляционные и логические операторы 40
4.3 Битовые и другие операторы 42
4.4 Ранг оператора 45
5.1 Селекция 47
5.2 Итерационные циклы с постусловием и предусловием, циклы со счетчиком 54
6.1 Первый класс в Java 62
6.2 Методы в Java 64
6.3 Другие элементы классов 74
6.4 Перечисляемые типы 80
7.1 Наследование в Java 83
7.2 Полиморфизм 87
7.3 Абстрактные базовые классы 93
7.4 Интерфейсы в Java 94
8.1 Одномерные и многомерные массивы 97
8.2 Сортировка массивов 109
8.3 Особые классы массивов 113
9.1 Чтение и запись файлов 118
9.2 Чтение и запись текстовых файлов 122
9.3 Сериализация объектов 124
9.4 Методы класса File 127
10.1 Исключения – Exceptions 130
10.2 Обобщенное программирование 137
11.1 GUI-программирование 141
11.2 Первая программа GUI 143
11.3 Вывод текстовой и графической информации 145
11.4 Событийно-ориентированное программирование 152
12.1 Элементы управления с помощью AWT 157
12.2 Элементы управления с помощью классов Swing 164

13.1 Создание меню с помощью AWT 180
13.2 Диалоги 185
13.3 Создание апплетов 191
14.1 GUI-конструктор NetBeans 197
14.2 Интеграция более сложных элементов управления 202
15.1 Доступ к базе данных с помощью Java 204
15.2 Доступ к другим базам данных 208
16.1 Основы приложений Android 210
16.2 Разработка прикладных и программ Android 213

1. Введение в технологию Java

1.1 Технология Java

1.1.1 Возникновение технологии Java

В начале 90-х годов в компании *Sun Microsystems* был разработан язык программирования, который можно было бы использовать не только на ПК, но и на разных электронных устройствах (например, на портативных мини-компьютерах или даже в умных кофемашинах). Этот язык программирования должен был называться *OAK*. Однако это название было защищено, так что разработчикам пришлось придумать новое имя: ***JAVA***[1].

Первая версия Java была представлена *Sun Microsystems* в 1995 г. Язык с возможностью доступа к Интернет – его можно было записать в определенном браузере (*HotJava*). Компания *Netscape* заключила договор с *Sun Microsystems* в 1996 г., и так Java получила стремительное распространение через известный браузер «*Netscape*» (*Netscape Navigator*).

Сейчас Java является мощным инструментом для разработки Интернет- и десктоп-приложений. Также она участвует в разработках *Mobile Computing*.

На следующем рисунке представлен обзор компонентов.

[1] Название Java связано с большой страстью разработчиков к кофе, так как Java во многих странах (включая США) является синонимом кофе.

1.1.2 Характеристики технологии Java

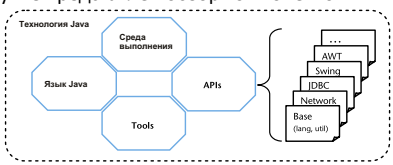
Большое преимущество Java – это кроссплатформенность. Исходный код Java переводится не на машинный код, а на один из видов промежуточного кода (байт-код). Потом этот байт-код может быть воспроизведен на всех платформах, имеющих соответствующую среду выполнения Java. Важнейшими характеристиками технологии Java являются:

- Объектно-ориентированность: Java – полностью объектно-ориентированный язык.
- Кроссплатформенность: Для большинства платформ была разработана среда выполнения Java, поэтому можно говорить об относительно высоком уровне кроссплатформенности. Существует, к примеру, среда выполнения у производственных систем Solaris (система «Unix»), Linux, Windows, также Mac OS X – тем самым, важнейшие производственные системы уже затронуты.
- Надежность: Java-программы работают под контролем в среде выполнения Java. Например, так называемый *garbage collector*, «сборщик мусора», обеспечивает надежное использование функции стирания памяти.
- Современная разработка приложения: С помощью Java можно программировать современные распределенные системы. Также доступ к базам данных поддерживается с помощью крупных библиотек.

1.1.3 Компоненты технологии Java

Java-технология состоит из различных компонентов, которые обеспечивают применение вышеописанных характеристик на практике. Наряду с собственным языком Java и средой выполнения к ним также относятся различные программные интерфейсы приложения APIs (***A**pplication **P**rogramming **I**nterfaces*). Все это объединяется в *Java Software Development Kit* (сокращенно ***JDK***).

На следующем рисунке представлен обзор компонентов.



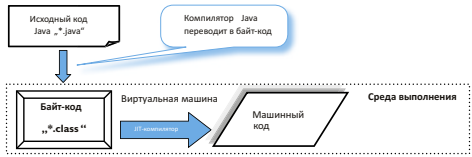
Большинство приложений может быть реализовано с помощью различных APIs. Отдельные APIs отвечают при этом за следующие сферы:

- **Base (lang и util):** Сборник классов для базового функционала: обработка строк, математические операции, форматированный вывод или обработка массивов.
- **Network:** С помощью этих классов можно реализовать сетевое программирование, например, через TCP-соединение и использование сокетов.
- **JDBC (J**ava **D**atabase **C**onnectivity**)**: С помощью этих классов идет подключение к базам данных. Также они являются основой распределенных приложений.
- **Swing и AWT (A**bstract **W**indow **T**oolkit**)**: Эти классы предоставляют компоненты для разработки графических интерфейсов пользователя.

1.1.4 Компиляция программ Java

Исходный код Java переводится непосредственно не в исполняемый файл, а в один из видов промежуточных кодов (байт-код). Затем этот промежуточный код воспроизводится средой выполнения Java. При этом так называемая виртуальная машина ***JVM*** (**J**ava **V**irtual **M**achine) переводит промежуточный код во внутренний, который является исполняемым кодом на соответствующей платформе. Актуальные виртуальные машины основаны на умных *Just-in-time*-компиляторах **(JIT)**, таких как ***HotSpot***, оптимизация которых позволяет очень быстро исполнять программы Java.

На рисунке представлен схематичный процесс компиляции:

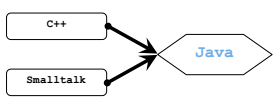


1.2 Язык Java

1.2.1 Развитие языка Java

Основа Java была заложена в начале 90-х годов группой разработчиков во главе с *Джеймсом Гослингом*[2]. Команда работала над новой технологией, которая должна была предоставить язык программирования с соответствующей средой выполнения для любого электроприбора. Первую версию этого современного и объектно-ориентированного языка Гослинг назвал ***OAK***. По юридическим причинам позже необходимо было переименовать *OAK*. Было выбрано имя ***Java*** (Ява это не только остров, но и название ароматного сорта кофе, используется во многих странах как синоним кофе). Java синтаксически сильно опирается на язык C++, но не переняла некоторые концепции C++ (множественное наследование). Поэтому, в сравнении с C++ Java более простая для изучения. Проблема резервирования памяти в C++ была решена в Java с помощью разработки программы ***garbage collector***. Так, стирание зарезервированной памяти осуществляется автоматически и дает большую разгрузку разработчику.

Веб-программирование совершило большой прорыв благодаря Java – во все традиционные браузеры (как *Firefox* или *Internet Explorer*) встроена Java. Благодаря так называемым апплетам Интернет-приложения становятся просты в реализации. Благодаря *Java Enterprise Edition* (***Java EE***-приложение) можно реализовать современные трехуровневые ***Three-Tier***-приложения[3], а с помощью технологии *Java Server Pages* (***JSP***) веб-сайты можно создавать достаточно быстро.



Java «наследует» некоторые концепции от полностью объектно-ориентированного языка Smalltalk и синтаксически сильно опирается на гибридный язык C++.

Язык ***Smalltalk*** был одним из первых объектно-ориентированных языков программирования (наряду с *Simula-67*) и уже располагал такими концепциями, как ***MVC*** (***M****odel* ***V****iew* ***C****ontroller*), который очень успешно применяется и в Java. Также концепция *garbage collector* уже использовалась в языке Smalltalk.

1.2.1 Характеристики языка Java

Следующие характеристики отличают язык Java:

- Современный, объектно-ориентированный язык
- *«Немного»* более простой для изучения, чем язык C++ (кавычки не следует использовать)
- Кроссплатформенная концепция
- Быстрая и эффективная разработка программного обеспечения (десктоп-приложения, веб-приложения) при поддержке мощных программных интерфейсов APIs и библиотек классов
- Комфортное соединение с любой базой данных

[2] Джеймс Гослинг работал с 1984 по 2010 г. в компании «Sun Microsystems». В последние годы он был техническим руководителем отдела исследований и разработок.

[3] Three-Tier-приложение имеет три уровня, установленных, как правило, на трех разных вычислительных системах. Уровни состоят из клиентской и серверной части и базы данных.

1.3.1 Ключевые слова в Java

Язык Java имеет лексикон из примерно 50 зарезервированных слов – так называемых **ключевых слов**. Ключевые слова образуют основу программ в Java. В таблице представлены ключевые слова Java:

abstract	continue	for	new	switch
assert	default	*goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Ключевые слова *const* и *goto* (еще?) нельзя использовать в Java. Они зарезервированы и поэтому не обозначают переменные или объекты.

Значения отдельных ключевых слов последовательно объясняются в данном информационном блоке.

1.3.2 Процедурное, структурированное и объектно-ориентированное программирование в Java

В программировании различают различные парадигмы[4]. Существуют такие языки, как C, при помощи которых возможно только структурированное (и процедурное) программирование. Другие языки, такие как C++, поддерживают как структурное (и процедурное), так и объектно-ориентированное программирование. Язык Java же является полностью объектно-ориентированным языком. Несмотря на это, структурированное программирование имеет значение и в Java, поскольку в рамках объектной ориентированности необходимо также структурное программирование.

Для большего понимания кратко объясним эти понятия.

Структурированное программирование

Структурированное программирование отличают управляющие конструкции, например, выбор (`IF-ELSE`) или повторяющиеся циклы (`FOR`, `WHILE` и др.). Так программа получает четкую структуру. У истоков программирования было принято использование операторов перехода (`GOTO`) в программе. Программа становится очень сложной и подверженной ошибкам. Структурированные же программы имеют ясную структуру и более удобны в обслуживании.

Пример:

```
для Var := от 1 до 5 с шагом 1
Напишите на экране Var
```

```
1
2
3
4
5
```

Пример показывает повторение в так называемом псевдокоде[5]. Этот код описывает процесс выполнения программы, но без специального языка программирования. На примере переменная Var повышается на 1 до тех пор, пока не будет достигнуто значение 5. Каждое значение переменной выводится на экран.

Процедурное программирование

Процедурное программирование делит программы на небольшие единицы (процедуры или функции), ответственные за определенные задачи. Если эти процедуры написаны и протестированы один раз, они могут быть использованы снова и снова – это экономит время разработки и дает большую возможность считывания программы.

[4] Слово „парадигма" заимствовано из греческого языка и имеет значение «образец» или «пример».

[5] Псевдокод – это вид языка, описывающий процесс выполнения программы. Псевдокод отличается тем, что он более близок к естественному языку, чем к языку программирования. Программу, написанную в псевдокоде, можно легко перевести на язык программирования.

Пример:

```
ПРОЦЕДУРА ВЫВОД
   НАПИШИТЕ НА ЭКРАНЕ «ПРИВЕТ»
КОНЕЦ

ДЛЯ Var := ОТ 1 ДО 5 С ШАГОМ 1
ЗАПУСК Вывод
```

```
Привет
Привет
Привет
Привет
Привет
```

Пример в псевдокоде демонстрирует процедуру под названием `Вывод`. Эта процедура включает оператор, который пишет слово «Привет» на экране. Уже известное повторение из вышеприведенного примера применяется 5 раз и каждый раз вызывает процедуру `Вывод`. Так, на экране 5 раз написано слово «Привет».

Объектно-ориентированное программирование

Цель объектно-ориентированного программирования – представить объекты реального мира в программе. Таким образом, формулировка задачи из любой области (деловые процессы, научные исследования и др.) может быть реализована в программах более удобным способом, чем в других парадигмах программирования.

В центре объектно-ориентированного программирования стоит **класс**, из которого затем создаются конкретные объекты. Эти объекты имеют определенные свойства (атрибуты) и так называемые методы, с помощью которых можно, к примеру, изменить эти свойства.

Пример:

```
КЛАСС Клиент
         Имя
         Телефон
КОНЕЦ
СОЗДАЙТЕ ОБЪЕКТ K1 для клиента
K1.Имя   := "Майер"
K1.Телефон := "123456"

НАПИШИТЕ НА ЭКРАНЕ K1.Имя и K1.Телефон
```

```
Майер
123456
```

В примере определен класс `Клиент`. Из этого класса могут быть созданы конкретные объекты, как `K1` (для клиента 1). Свойствам объекта (имя, телефон) можно присвоить значения. В данном примере объект `K1` получает имя «Майер» и номер телефона «123456». Далее имя и телефон объекта отображаются на экране.

1.3.3 Составляющие программы Java

Программа Java состоит из последовательности конечного множества однозначных операторов[6], которые создаются с помощью ключевых слов и самостоятельно выбранных названий для определенных элементов, таких как классы или объекты. Дополнительно программа Java может содержать операторы, не относящиеся к самой программе, а управляющие ее созданием. Следующий пример демонстрирует первую простую программу Java:

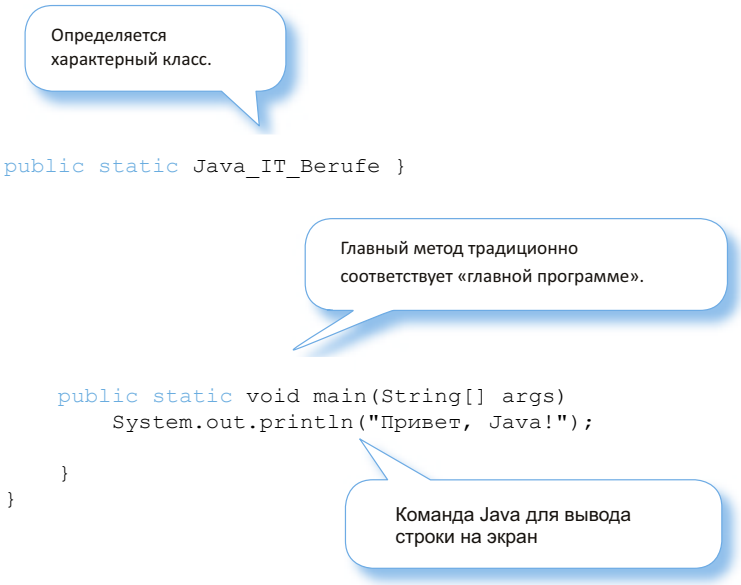
```
package java_it_berufe;
```

Так определяется характерный пакет с названием «`java_it_berufe`».

```
import java.io.*;
```

Команда `import` обеспечивает интеграцию библиотек, в данном случае, библиотек ввода-вывода.

[6] Конечную последовательность однозначных операторов компьютера называют алгоритмом.



Из вышеприведенного примера становится ясно, что даже простая программа Java имеет относительно сложную структуру. Это связано с тем, что Java является полностью объектно-ориентированным языком и поэтому необходимо всегда определять класс. Данная структура будет последовательно разобрана в следующих главах.

2 Первая программа Java

2.1 Создание проекта Java

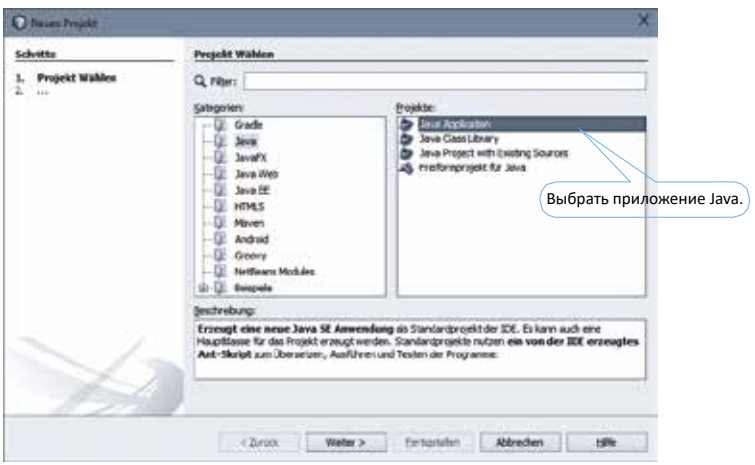
Интегрированная среда разработки *NetBeans IDE 8.0.1* – это комфортная среда для разработки программ Java. Особенно радует то обстоятельство, что среда доступна бесплатно в Интернете. Программа Java состоит из одного или нескольких файлов с исходным кодом. Эти файлы собраны в одном проекте. *NetBeans* предлагает множество проектов. Ниже перечислены самые важные:

- Java-приложение (создает базовое приложение)
- Библиотека классов Java (создает модель библиотеки Java)
- Веб-приложение Java (генерирует пустое веб-приложение)
- Приложение Java-Enterprise (создает модель распределенного приложения)

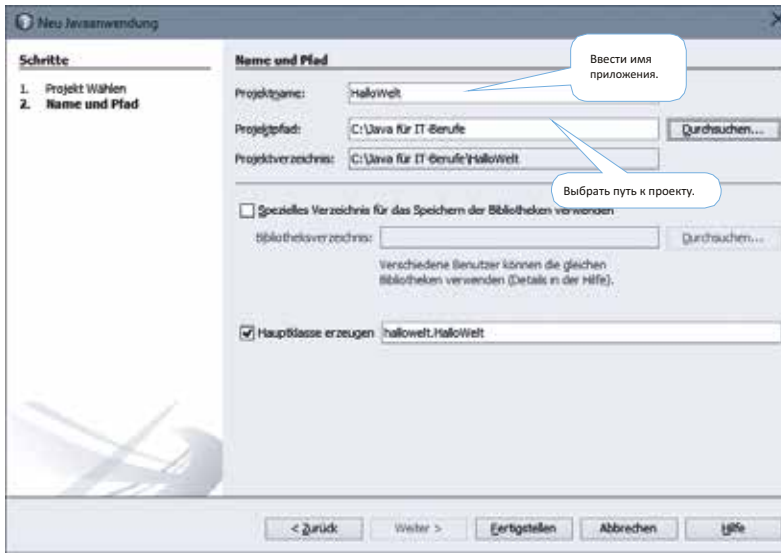
В этой книге используется преимущественно проектная форма: **Java-приложение.** Она позволяет разрабатывать как простые консольные приложения, так и GUI-приложения. В конце книги будет рассмотрен новый проект: **проект Android** – однако для этого необходимо провести некоторую подготовку.

Составление нового проекта:

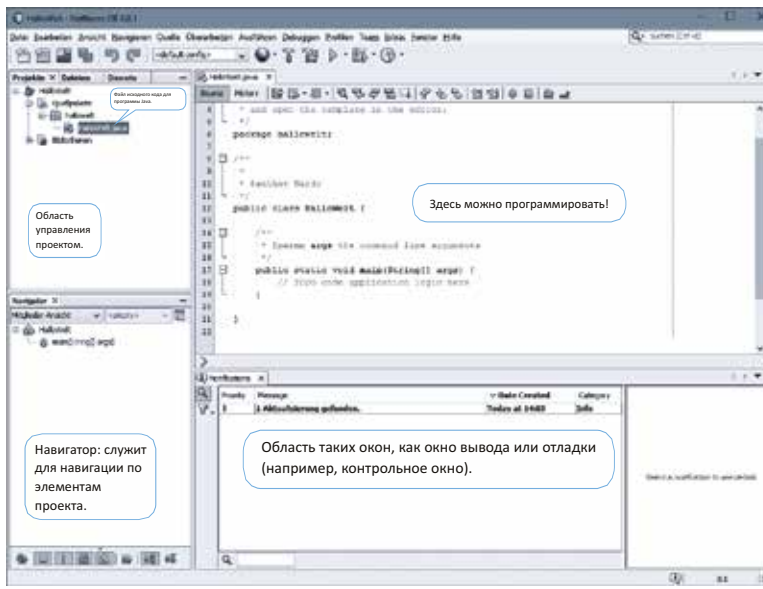
- Запустите *NetBeans DIE 8.0.1.*
- Выберите пункт меню Файл → Новый проект.



После подтверждения Далее следует ввести имя и выбрать путь к проекту.



Далее можно создать проект, нажав кнопку Завершить.



Среда разработки создала проект с выбранным именем (здесь «HalloWelt»). Можно создавать любое количество других проектов. Внутри проекта создан файл с исходным кодом «HalloWelt.java». Дополнительно отображается используемая библиотека. В созданном файле с исходным кодом уже имеется основная структура, представляющая отлаженную программу Java – но без функционала.

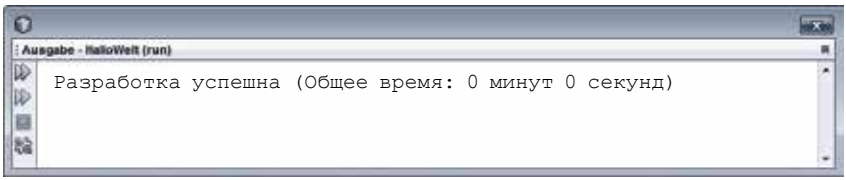
Выполнение программы Java

Для компиляции и выполнения программы в *Netbeans* существуют разные возможности:

- Пункт меню: Выполнение → Проект (HalloWelt) выполнение
- Пункт меню: Отладка → Проект (HalloWelt) отладка
- Комбинация клавиш: F6 (Выполнение) или STRG + F5 (Отладка)

Позже, уже имея некоторые знания, Вы будете использовать преимущественно отладку для анализа сложных программ. Для новичков же абсолютно достаточно вышеописанной процедуры действий.

После запуска (Выполнение) первой программы открывается следующее окно:

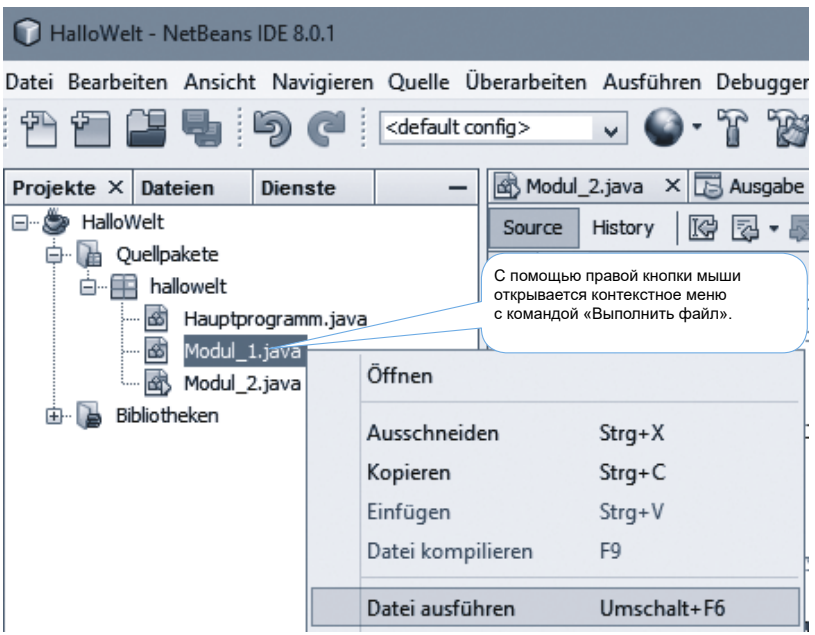


Примечание

Окно вывода в среде разработки свернуто по умолчанию. Его можно развернуть щелчком правой кнопкой мыши (как показано выше).

Альтернатива: Исполнение отдельных файлов Java

Альтернативой вышеописанного способа действий, при котором всегда выполняется весь проект, является исполнение отдельных файлов Java. Это экономит время, когда за один этап тестирования необходимо проверить несколько файлов Java, но выполнить весь проект еще нельзя. На следующем рисунке показано исполнение отдельных файлов:



2.2 Первая программа Java

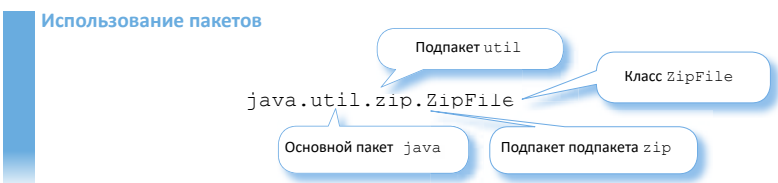
2.2.1 Основная структура Java

Первый простой пример программы Java уже был представлен в первой главе. Теперь рассмотрим эту основную структура подробнее, так как она является исходной базой для всех остальных программ Java.

В принципе, в начале можно было бы ограничиться главным методом (и его содержанием). До глав о методах и понятии классов в Java интерес представляет только этот главный метод, поскольку до этого все программы Java полностью написаны в этом методе. Однако самые важные компоненты все равно будут кратко освещены в следующих подразделах. Далее информационном блоке они будут рассмотрены подробнее.

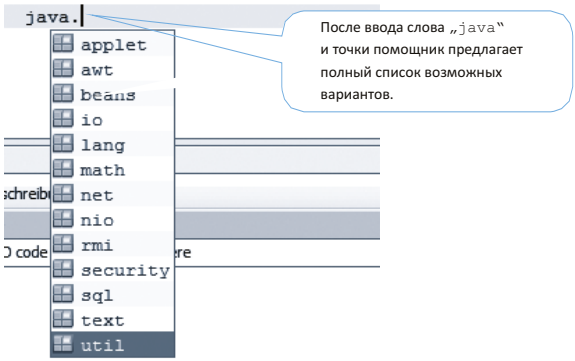
2.2.2 Пакеты

При разработке программ Java требуется много различных компонентов, например, чтобы писать на экране, считывать с клавиатуры или обращаться к базе данных. Эти компоненты доступны в форме библиотеки классов. Для придания структуры библиотеке все классы, необходимые, например, для вывода на экран и ввода данных с клавиатуры сводятся в характерном пакете. Другие классы, необходимые, например, для чтения и письма, также собираются в характерный пакет. Пакеты в свою очередь могут содержать другие пакеты (подпакеты), так образуется рациональная структура. Основной пакет называется ***java***. В нем находятся самые важные классы и другие пакеты. Доступ к отдельным подпакетам или классам пакета осуществляется через так называемый точечный оператор (то есть точку), как показано в следующем примере:



Примечание

Среда разработки NetBeans удобна для пользователя тем, что она имеет помощника, поддерживающего ввод команд Java и поиск по пакетам и классам. При вводе имени пакета или объекта помощник пытается предложить подходящее продолжение. Как правило, этот инструмент очень помогает при разработке.



Команда импорт

Эта команда упрощает использование пакетов. Если для пакета указана команда `импорт`, то можно напрямую обратиться ко всем компонентам пакета.

Пример: без команды `import`

Выбор класса `ZipFile` из пакета `java.util.zip`:

```
java.util.zip.ZipFile einFile;
```

Пример: с командой `import`

Выбор класса `ZipFile` из пакета `java.util.zip`:

```
import java.util.zip.*;
ZipFile einFile;
```

Знак «*» означает, что все компоненты пакета должны быть интегрированы.

Теперь можно иметь доступ напрямую к классу, не вводя полное имя пакета.

2.2.3 Класс «Hello, world!» и главный метод main

Язык Java это полностью объектно-ориентированный язык. Поэтому каждая программа Java состоит минимум из одного класса. В новом Java-приложении среда разработки автоматически создает класс, который называется так же, как и проект – в данном случае «Hello, world!». Внутри класса есть так называемый статический метод `main`. Этот метод выполняется при запуске программы, то есть, это главная программа. Поэтому этот метод нельзя переименовать, потому что иначе компилятор не сможет найти «главную программу”. До изучения глав книги о методах и классах все операторы Java написаны в рамках этого статического метода `main`.

Примечание

До настоящего момента не имеет значения, если понятия класса и статического метода были не совсем понятны. Они могут стать понятными только в ходе следующих глав. Вполне достаточно, если присутствует ориентирование в компонентах программы Java в целом. С помощью концентрации на содержании метода `main` достаточно просто изучить основы программирования Java в следующих главах.

2.2.4 Вывод на экран

Теперь рассмотрим еще одну возможность первой программы – письмо на экране. Таким образом, реализуется первая возможность коммуникации между пользователем и программой. Исходная база это снова основная структура, которая создается средой разработки автоматически при новом проекте:

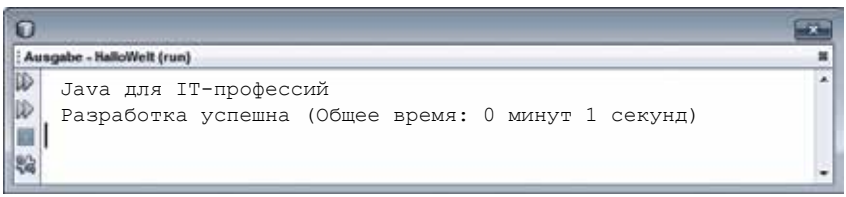
```
package helloworld;

public class helloworld{

public static void main(String[] args) {

  System.out.println("Java для IT-профессий");
```

Метод `println` статического объекта `out` пишет строку на экране и автоматически осуществляет разбивку на строки.

Примечание

Класс `System` относится к пакету `java.lang`. Этот пакет не нужно связывать командой `import`. Он ***автоматически***[1] встроен в каждый проект.

После запуска появляется следующий вывод на экран:



Что такое цепь символов?

Цепь символов это последовательность конечного количества символов, то есть букв, цифр и специальных знаков, которые приводятся в кавычках.

[1] Пакет `java.lang` предусмотрен разработчиками Java как базовый, чтобы он всегда был интегрирован.

2.2.5 Важные правила программы Java

После первых примеров простых программ Java можно закрепить основные правила:

- Каждая программа Java имеет минимум один класс и один главный метод `main`.
- Операторы главного метода заключаются в фигурные скобки { } – как и определение класса.
- Каждый оператор в Java (например, `System.out.println("…");`) оканчивается точкой с запятой.
- С помощью команды `import` легче обратиться к компонентам пакета.
- Java пишется прописными и строчными буквами!

2.3 Основополагающие конвенции в Java

После знакомства с первыми программами Java необходимо изучить другие аспекты, важные при разработке программы.

В связи с этим возникают следующие вопросы:

- Как создаются имена в Java (для переменных и др.)?
- Как пустые строки, разбивки строк или пропуски интерпретируются в исходном коде?
- Как можно писать комментарии в исходном коде?

2.3.1 Идентификатор (имя) в Java

Как и во всех языках программирования, в Java есть имена для переменных, констант, методов и классов. Эти самостоятельно выбранные или определенные имена подчиняются определенным правилам, которые обязательно необходимо соблюдать:

- Первым символом должна быть буква (также разрешено нижнее подчеркивание «_»).
- Остальная часть может состоять из цифр и букв.
- Идентификатор не должен совпадать с ключевым словом Java.

Примеры:

`zahl`	действительный идентификатор
`_zahl`	действительный идентификатор
`2mal7`	недействительный идентификатор (первым симфолом является цифра)
`break`	недействительный идентификатор (ключевое слово в Java)
`zahl 1`	недействительный идентификатор (пробел после слова «zahl»)

Соглашение об использовании имен Java

Обозначение идентификаторов предоставлено программисту. Однако целесообразно с самого начала придерживаться определенной конвенции в программе. Здесь поможет так называемый *СтильВерблюда (CamelCase)*: он предполагает, что идентификатор всегда начинается со строчной буквы. Если имя состоит из нескольких компонентов, то следующий компонент начинается с прописной буквы. Имена классов же всегда начинаются с прописной буквы. На следующих примерах эти правила показаны более наглядно:

Имена переменных

- `однаПеременная`
- `многоМашин`
- `новыйОбъект`

Имена методов

- `присвоитьЗначение ( )`
- `открытьФайл ( )`
- `закрытьОкно ( )`

Имена классов

- ПервыйКласс
- Сотрудник
- ДиалоговоеОкно

Примечание:

Имена пакетов всегда должны писаться только с маленькой буквы, а постоянные значения всегда с прописной (подробнее об этом позже).

2.3.2 Разделительный знак

Между разными командами и выражениями программы Java должен быть разделительный знак – точка с запятой. Несколько связанных команд (как при методе `main`) объединяются фигурными скобками.

Между ключевыми словами в Java и характерными идентификаторами должен стоять либо пробел, либо оператор для различия (напр. символ добавления «+»).

При этом конец строки, табулятор и несколько пробелов интерпретируются как пустые символы или разделительные знаки.

Пример:

Все три программы идентичны по своему функционалу. Они отличаются только порядком исходного текста:

Программа 1:

```
package helloworld;
public class helloworld{

        public static void main (String [ ] args)
                    System.out.println("Пробелы и Co.");
}
}
```

Между ключевым словом (здесь: `void`) и идентификатором (здесь: `main`) должен быть разделительный знак (здесь пробел).

Команда пишется в одной строке, чтобы вывести на экран, и оканчивается точкой с запятой.

Программа 2:

```
package helloworld;
public class helloworld{

 public static void main (String [ ] args)
 System.out.println("Пробелы и Co.");

                ;
        }
}
```

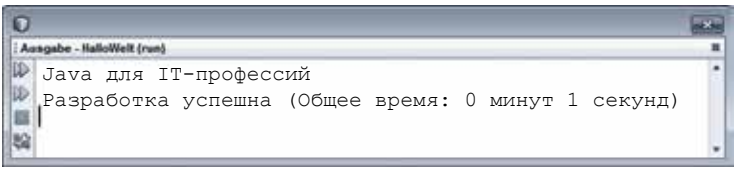
Вся команда делится на три строки, чтобы вывести на экран.

Программа 3:

```
package helloworld; public class helloworld {public static void
main(String[]args){System.out.println(«Пробелы и Co.»);}}
```

Вся программа пишется просто последовательно.

Видно, что первый вариант более читабельный, чем второй. Третий вариант представлен только для наглядности – программа Java не должна так выглядеть. Однако во всех трех случаях после запуска текст одинаково выводится на экран:



2.3.3 Комментарии в Java

Каждый программист знает, как важны комментарии в программе, особенно если через несколько месяцев нужно изменить исходный код, и никто уже не помнит, какие значения были у переменных или методов. Комментарии игнорируются при переводе компилятора, они служат только для понимания исходного кода.

Комментарий должен быть заключен в рамку из последовательности символов «/*» и «*/». В среде разработки *NetBeans* комментарии отображаются серым цветом. Комментарии могут быть на несколько строк, но не на несколько уровней. Комментарии, которые пишутся только в одну строку, начинаются с символов «//». Специальные комментарии ***javadoc*** начинаются с символа «/**» и со «звездочки» в каждой строке. Они также заканчиваются символом «*/».

Пример:

Программа Java с комментариями:

```
//Создается пакет
package helloworld;
//Пакет java.util импортируется
import java.util.*;

public class helloworld{

   /**
   *      Краткое описание метода…
   *      @param args
     *    Объясняются аргументы метода main
   */


   public static void main(String[] args) {
   /* Здесь идет программирование */
  }
}
```

Стандартный комментарий начинается с символов «//».

Комментарий `javadoc`

Комментарий на несколько строк

Примечание:

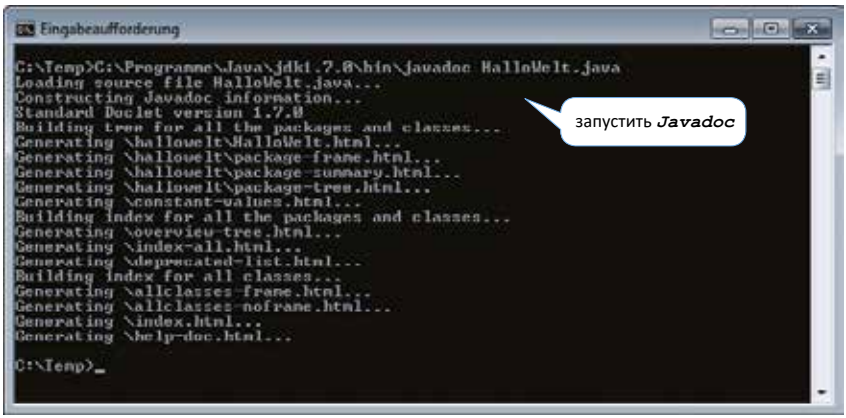
Комментирование исходного кода важно именно для повторного использования и четкой структуры исходного кода. Однако этим нельзя злоупотреблять. Естественные значения не нужно дополнительно комментировать. Важнее краткие и точные, не перегруженные объяснения. В начале исходного кода можно создать заголовок для комментария, который будет передавать информацию через исходный код или программу.

Пример:

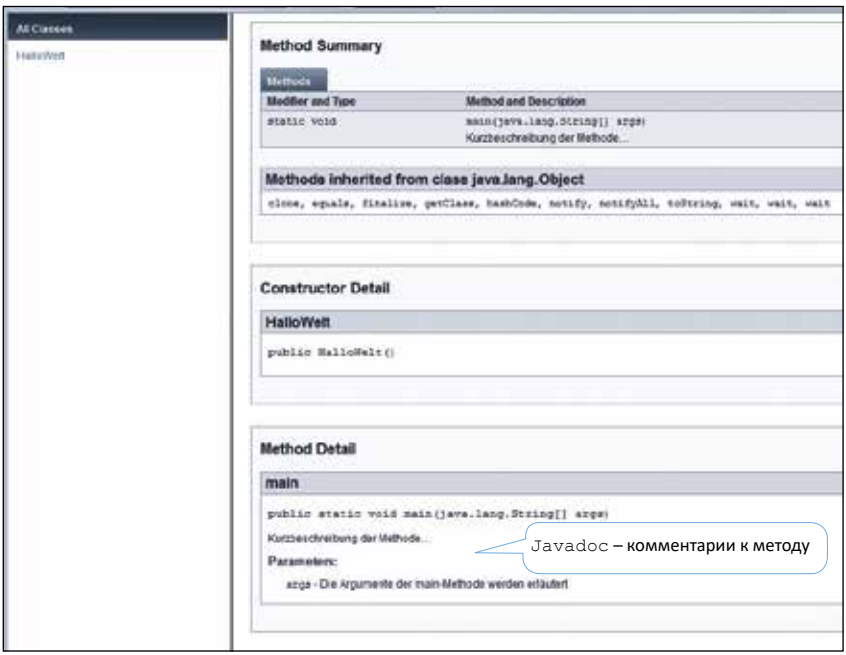
```
/**
* Создан 10.08.2015
* Последние изменения от 12.08.2015
* @автор Дирк Харди
*/
```

Комментарии с javadoc:

Комментарии, созданные в вышеприведенных примерах в стиле ***javadoc***, можно использовать с помощью программы ***javadoc*** для автоматически созданной документации.



После запуска программы ***javadoc*** автоматически создаются структурированные HTML-документы:



2.4 Типы данных и переменные

Базовая функция программ состоит в хранении и обработке данных. Мало программ ограничивается простой функцией вывода на экран. Собственно программирование начинается с возможности считывать значения через клавиатуру и соответственно их обрабатывать. В следующих главах освещаются основы этих процессов.

2.4.1 Переменные в Java

В программах Java переменные служат для того, чтобы хранить значения. Они являются символом-заполнителем для любого значения. Например, для расчета процентов можно использовать переменную, чтобы сохранить результат расчетов. Переменные имеют имя, которое должно быть действительным идентификатором.

В некоторых языках программирования (напр. BASIC) существуют переменные, которые могут сохранять любые значения (целочисленные значения, числа с плавающей запятой, цепи символов и др.).

В Java необходимо предварительно определить, какой вид значений должна сохранять переменная. При этом речь идет о типе данных переменной. Тип данных точно определяет диапазон и вид значений. Также есть области определения переменных. Некоторые переменные действуют локально (то есть ограничены местом) – например, только в одном методе. Некоторые переменные могут быть действительными для нескольких методов. Эти переменные создаются в соответствующем классе. Однако полное представление о действии можно получить только после проработки концепции класса. До этого переменные рассматриваются преимущественно в методе `main`.

Примеры переменных:

Имя переменной	Функция переменной
`счетчик`	Переменная для подсчета прохождения программы
`конечныйКапитал`	Сохраняет стоимость капитала для делопроизводственного расчета
`случайноеЗначение`	Сохраняет случайное значение

Переменные сохраняют значения в зависимости от функции. Однако не каждая переменная может сохранять любое значение. Для этого переменные имеют специализацию – они всегда сохраняют только определенные значения. Переменная имеет специальный тип данных.

2.4.2 Простые типы данных

В Java существуют разные типы данных для разных форматов данных. В математике существует множество натуральных чисел (положительные целые числа), множество целых чисел (положительные и отрицательные целые числа) и действительные числа (десятичные дроби). Для этих типов чисел в Java есть соответствующие типы данных (рациональные и комплексные числа в этом отношении не играют роли). Дополнительно существуют специальные типы данных для сохранения отдельных символов, цепей символов или истинных значений (логических типов). В следующей таблице представлены отдельные типы данных и их области определения:

Тип данных	Описание	Размер в байтах	Диапазон значений
`byte`	Служит для сохранения целочисленных значений.	1	От – 128 до 127
`char`	С помощью этого типа данных можно сохранять юникоды-набор символов.	2	юникод-набор символов
`short`	Служит для сохранения положительных и отрицательных целочисленных значений.	2	– 32768...32767
`int`	Служит для сохранения положительных и отрицательных целочисленных значений.	4	– 2147483648... 2147483647
`long`	Служит для сохранения положительных и отрицательных целочисленных значений.	8	– 9223372036854775808... 9223372036854775807
`float`	Служит для сохранения чисел с плавающей запятой.	4	– 3,402823E+38... 3,402823E+38

Тип данных	Описание	Размер в байтах	Диапазон значений
`double`	Служит для сохранения чисел с плавающей запятой.	8	– 1,797693134862E+308... 1,797693134862E+308
`boolean`	Может сохранять логические типы.	1	true или false
`String`	Служит для сохранения цепей символов. ВНИМАНИЕ: Строковый тип данных это класс, который только похож на простой тип данных.	По требованию	Все символы из юникод-набора символов

Типы значений и ссылок

Все простые типы данных относятся к так называемым **типам значений** в Java. Только строковый тип данных является типом ссылок. Для выявления разницы между типами значений и ссылок рассмотрим их подробно в главе о классах.

2.4.3 Объявление переменных

С помощью простых типов данных и соответствующих идентификаторов (имен) в Java может быть создана переменная. Для этого сначала необходимо создать тип данных и затем (отделив пробелом) идентификатор переменной. Следующие примеры демонстрируют объявление некоторых переменных:

Примеры переменных:

```
//Создать переменную типа int:
int i;

//Создать переменную типа double:
double d;

//Создать переменную типа char:
char c;

//Создать переменную типа String:
String s;
```

Инициализация переменных:

При объявлении переменных сразу может быть осуществлена инициализация. Так переменным присваивается значение. Это происходит с помощью оператора присваивания «=».

ВНИМАНИЕ:

Перед первым использованием переменной после ее объявления ее необходимо инициализировать.

Примеры объявлений с инициализацией:

```
// Создать переменную типа int и инициализировать:
int i = 10;

// Создать переменную типа double и инициализировать:
double d = 1.25;
```

Инициализация с помощью литерального значения

Инициализация с помощью литерального значения

```
// Создать переменную типа char и инициализировать:
 char c = 'x';

//Переменная
String s = "Цепь символов";
```

Простые символы в одинарных кавычках!

Цепи символов в двойных кавычках!

Литералы

Постоянные значения в исходном тексте называются **литералами.** Вышеприведенные значения инициализации являются литералами. Литералами могут быть численные значения от 10 или 1.25 (точка обозначает десятичный разделитель) или строки символов, заключенные в двойные кавычки. Простые же символы оформляются одинарными кавычками.

Для продвинутых пользователей:

Для математических или делопроизводственных расчетов важно знать, на скольких позициях работает тип данных. Тип данных `float` может работать на 6 позициях, а `double` – на 15. Если сохранить следующее число с помощью переменной по типу `float`, его нельзя будет распознать, так как точность равна 6.

123.456**789** ⟷ 123.456**999**

2.4.4 Операции с простыми типами данных

Арифметические операторы (+, -, /,*) известны с уроков математики. С помощью них можно складывать, вычитать, делить и умножать. Естественно, эти операции можно проводить и с помощью переменных в Java. С помощью оператора присваивания «=» значения и результаты расчета присваиваются переменным. На следующих примерах показано использование этих переменных:

Пример 1:

```
public class ПростыеТипыданных {
public static void main(String[] args) {

    int значение1 = 10;
    int значение2 = 20;
    int сумма = 0;
    int продукт = 0;
    сумма = значение1 + значение2
    продукт = значение1 * значение2;
```

Создаются четыре целых переменных с соответствующими инициализациями.

Рассчитывается и присваивается сумма и результат первых двух переменных.

Объявление и «говорящие» имена

Как видно из примера, переменные создаются в начале метода. Хоть это и не обязательно, однако это повышает читабельность исходного текста и поэтому необходимо применять. При выборе идентификатора следует использовать «говорящие» имена (как `сумма` и `продукт`). Так исходный текст также будет более читабельным и понятным.

Пример 2:

```
public class ПростыеТипыданных {
                    public static void main(String[] args) {

                            double x = 1.5;
                            double y = 2.5;
                            double quotient = 0;
```

```
                quotient = x / y;
            }
}
```

Число с плавающей запятой x делится на число с плавающей запятой y и присваивается результат переменной quotient.

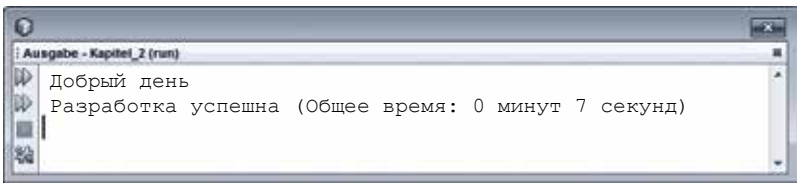
Пример 3:

```
public class ПростыеТипыданных {
        public static void main(String[] args) {

            String s1 = "Добрый";
            String s2 = "день";
            String сцепление = ""


            сцепление = s1 + " " + s2;


            System.out.println(сцепление);
        }
}
```

В переменной сцепление появилась цепь символов: «Добрый день»

Цепи символов могут быть сцеплены, то есть связаны друг с другом, с помощью плюс-оператора.

После запуска вышеуказанной программы вывод на экран может выглядеть так:



Примечание:

Кроме плюс-оператора для цепей символов больше нет других арифметических операторов.

2.4.5 Постоянные переменные

С помощью ключевого слова final можно создавать так называемые постоянные переменные. Постоянные переменные объявляются только один раз и сразу инициализируются. После этого значение постоянной переменной нельзя изменить. Логичным образом такие константы используются для постоянных значений, которым необходимы «говорящие» имена. Например, так могут быть представлены физические или математические постоянные.

Постоянные имена переменных должны оформляться прописными буквами.

Пример:

```
final double PI = 3.14;
final double УСКОРЕНИЕ ПАДЕНИЯ = 9.81;

УСКОРЕНИЕ ПАДЕНИЯ = 10.81;
```

ОШИБКА: присваивание невозможно

3 Ввод и вывод в Java

3.1 Система вывода в Java

В предыдущей главе уже был показан простой вывод на экран. Работа программ зависит от взаимодействия с пользователем. Возникает вопрос, как можно лучше оформить вывод на экран и как пользователь сможет вводить значение для программы (ввод с клавиатуры). Для этого существуют статические методы класса `System` и метод класса `BufferedReader`.

3.1.1 Вывод переменных

Уже знакомый метод `println()` может наряду со строками символов выводить также и содержимое переменных. Дополнительно существует метод `printf()`, который особенно удобен для программиста на C. Следующие примеры демонстрируют использование методов:

Пример 1: Вывод с помощью `println()` и плюс-оператора

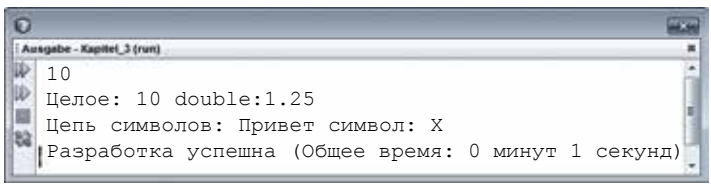
```
public class Вывод {

public static void main(String[] args) {
int i = 10;
double d = 1.25;
String s = "Привет";
char c = 'x';

System.out.println(i)
System.out.println("целое:" + i + " double: " + d);
System.out.println("цепь символов:" + s + " символ:" + c);
System.out.println();
}
}
```

Вывод содержимого переменной i.

Плюс-оператор связывает строку символов с содержимым переменных

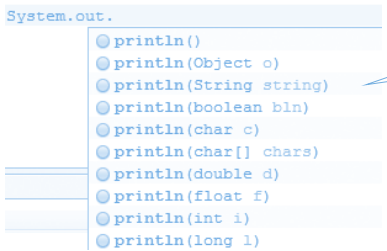
С помощью плюс-оператора содержимое переменных автоматически преобразуется в цепи символов и может быть связано с другими строками и записано на экране.

После запуска вывод на экран выглядит так:



Для продвинутых пользователей:

Метод `println()` может вывести содержимое любых переменных, потому что для каждого типа данных есть свой вариант исполнения метода, который реагирует на особенности типа данных. Эта техника называется перегрузкой методов и будет подробнее рассмотрена в теме методов в Java.



Пример 2: Вывод с помощью `printf()` и символов-заполнителей

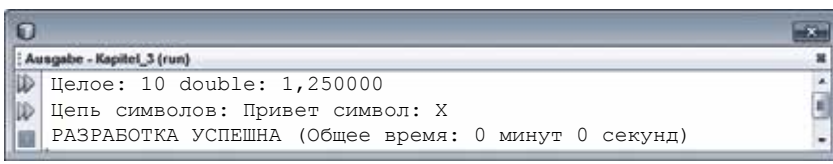
```
public class Вывод {
public static void main(String[] args) {

int i = 10;
double d = 1.25;
String s = "Привет";
char c = 'x';

System.out.printf("Integer: %1$d / double: %2$f %n", i, d);

System.out.printf("цепь символов: %1$s / символы:%2$c",s, c);

   }
}
```

После знака процента «%» создается символ-заполнитель для содержимого переменной.
У каждого символа-заполнителя есть номер (индекс). Первый номер начинается с единицы. С помощью знака «%n» производится разрыв строки.

Переменные вводятся в порядке символов-заполнителей и отделяются запятыми.

После знака доллара «$» вводится формат переменных.
Возможны следующие форматы:
- `d` десятичное число (целочисленное значение)
- `f` число с плавающей запятой (десятичная дробь)
- `c` символ (Charakter)
- `s` цепь символов (String)

После запуска вывод на экран выглядит как в первом варианте:



3.2 Ввод с консоли

3.2.1 Ввод цепи символов

Ввод символов с клавиатуры может быть осуществлен с помощью метода `readLine()`. Этот метод считывает входной поток с клавиатуры и сохраняет его в цепи символов. Далее эта цепь символов может быть присвоена переменной `String`. Однако для этого объект ввода должен быть использован по типу `BufferedReader`. Понимание о создании и использовании этого объекта ввода может прийти позже, в главах о классах и методах – до этого он просто будет использован. Также «главный метод» должен быть подготовлен к поглощению ошибок. Это происходит путем введения обработки ошибок (`throws IOException`). Данную проблему мы также рассмотрим подробно позже. На следующем примере показано применение метода `readLine()`:

Пример: ввод с

```
import java.io.*;

public class Ввод {

    public static void main(String[] args) throws IOException

String s;

BufferedReader ввод lesen = new BufferedReader (new BufferedReader(new
                            InputStreamReader(System.in));

System.out.println("Пожалуйста, введите текст: ");

s = ввод lesen.readLine()
System.out.println("Ввод: " + s);
    }
}
```

Пакет `java.io` необходимо интегрировать.

Введение обработки ошибок `IOException` готовит метод к ошибкам ввода.

Создается объект для ввода с консоли.

Метод `readLine()` объекта ввода применяется для ввода цепи символов. Далее эта цепь символов присваивается переменной String.

После запуска программа ожидает ввода с клавиатуры. Ввод подтверждается кнопкой RETURN и затем пишется на экране.

```
Ausgabe - Kapitel_3 (run)
Пожалуйста, введите текст:
Привет
Ввод: Привет
РАЗРАБОТКА УСПЕШНА (Общее время: 0 минут 4 секунд)
```

3.2.2 Конвертация ввода

Метод `readLine()` всегда сохраняет ввод в строке символов. Поэтому требуется произвести преобразование (конвертацию), если, например, необходимо ввести целое число или число с плавающей запятой с клавиатуры. Для такого преобразования

существуют специальные классы – так называемые классы оболочки ***(Wrapper)***. Эти классы представляют собой объектно-ориентированную форму простого типа данных. Например, классы располагают статическим методом, который может преобразовать соответствующее числовое значение. Следующий пример демонстрирует преобразование цепи символов в целое число и число с плавающей запятой:

Пример:

```
import java.io.*;

public class Ввод {
public static void main(String[] args) throws IOException {

   String s;
   int i;
   float f;
   BufferedReader ввод = new BufferedReader(new
                         InputStreamReader(System.in));

   System.out.println("Пожалуйста, введите текст: ");
   s = einlesen.readLine();
```

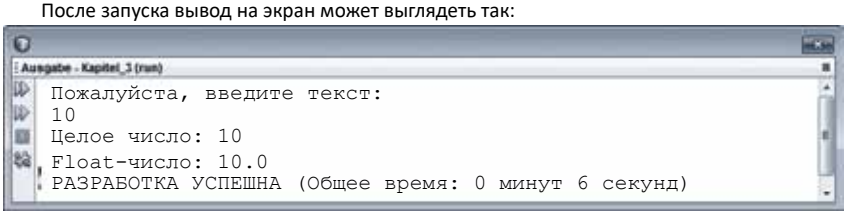
Wrapper-класс `Integer`

```
   i = целое.parseInt(s);
   f = Float.parseFloat(s);
```

Статический метод `parseFloat()` преобразует строку символов в значение Float.

```
   System.out.println("Целое число: " + i);
   System.out.println("Float-число: " + f);

  }
}
```

После запуска вывод на экран может выглядеть так:



Другие классы оболочки и методы их конвертации:

- `Boolean.parseBoolean(...);`
- `Byte.parseByte(...);`
- `Short.parseShort(...);`
- `Long.parseLong(...);`
- `Double.parseDouble(...);`

Классы предлагают соответствующий метод для всех простых типов данных. Разумеется, возможно преобразование не только цепи символов в число, но и числа в цепь символов или чисел разных типов данных между собой, как показано на примере:

Пример:

```
String цепь символов
float f = 10.5f;
int i = 5;
```

ВНИМАНИЕ: Литералы с плавающей запятой имеют тип `double`. Поэтому к литералам типа `float` нужно добавить «`f`».

```
Цепь символов = Float.toString(f);
```

Преобразование числа с плавающей запятой в строку символов.

```
Цепь символов = Integer.toString(i);
```

Преобразование целого числа в строку символов.

Примечание:

Конвертация (преобразование) типов данных будет рассмотрена подробнее в главе об операторах. Наряду с вышеприведенными примерами эксплицитного (явного) преобразования существуют другие возможности для имплицитного (автоматически производящегося) преобразования типов данных.

4 Операторы в Java

Операторы являются очень важной составляющей языка программирования. Без операторов не может быть написана ни одна программа. От системы ввода/вывода еще можно было бы отказаться при необходимости, но от использования операторов – нет.

4.1 Арифметические операторы

4.1.1 Простые типы данных и их арифметические операторы

Ранее мы уже говорили об арифметических операторах (так называемые основные арифметические операторы) для разных типов данных. Операторы используются так, как мы к этому привыкли на уроках математики. Однако следует различать разные типы данных, поскольку операторы ведут себя по-разному в зависимости от типа данных.

Тип данных по типу чисел с плавающей запятой (float и double):

Основные арифметические операторы применяются как обычно. Числа данного типа можно складывать, вычитать, умножать и делить. Результатом операций также являются числа по типу плавающей запятой.

Пример:

```
float a = 1.2f;
float b = 10.45f;
float c;
c = a + b;     // Переменная c имеет значение 11.65
c = a / b;     // Переменная c имеет значение 0.1148…
```

Примечание:

Литералы типа чисел с плавающей запятой всегда имеют тип данных `double`. Чтобы переменной `float` можно было присвоить число с плавающей запятой, к `float` следует добавить суффикс «`f`».

Типы данных по типу целых чисел (byte, short, int, long):

Основные арифметические операторы также применяются как обычно. Числа данного типа можно складывать, вычитать, умножать и делить. Результатом операций также являются числа по типу целых чисел.

Пример:

```
int x = 1;
int y = 2;
int z;
z = x + y;        // Переменная c имеет значение 3
z = x / y;        // Переменная c имеет значение 0
```

Операция деления целого числа имеет остаток, но не имеет знаков после запятой. Знаки после запятой «сокращаются». Остаток деления можно определить с помощью другого оператора – оператора модуля (см. следующий подраздел).

4.1.2 Оператор модуля

Этот оператор выводит остаток деления целого числа. Например, число 25 делится на 7 только с остатком. Остаток равен 4. Именно этот остаток выводит оператор модуля «%».

Пример:

```
int a = 25;
int b = 12;
int c;
c = a % b;        //c имеет значение 1
c = b % a;        //c имеет значение 12
```

Эта операция поначалу кажется странной, так как b меньше, чем a. Однако логика оператора модуля однозначна.
12 делится на 25 0 раз и остается остаток 12.

Примечание: Использование оператора модуля

Оператор модуля всегда применяется тогда, когда необходимо внедрить математические алгоритмы. Например, номера счетов и БИК имеют так называемые контрольные разряды. В частности, эти контрольные разряды рассчитываются с помощью оператора модуля.

4.1.3 Операторы инкремента и декремента

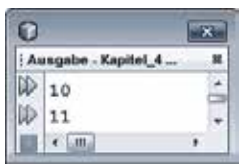
В программировании часто бывает, что значение переменной необходимо увеличить или уменьшить на 1, например, переменных, подсчитывающих определенные процессы или операции. Для этого существуют специальные операторы, которые увеличивают или уменьшают переменную на 1. Это оператор инкремента «++» и оператор декремента «--».

ВНИМАНИЕ:

Важна позиция операторов. Существует префиксная и постфиксная нотация. Это означает, что оператор пишется до и после переменной. Это показано на примерах:

Постфиксная нотация

```
int x = 10;
System.out.println(x++);
System.out.println(x);
```



Значение x увеличивается, однако это действует только в следующем операторе. При выводе x еще имеет значение 10, только после этого 11.

Префиксная нотация

```
int x = 10;
System.out.println(--x);
System.out.println(x);
```

Значение x уменьшается именно в этом же операторе. При выводе x уже имеет значение 9; далее, разумеется, тоже 9.

4.2 Реляционные и логические операторы

4.2.1 Реляционные операторы

Реляционные операторы это операторы сравнения. Они служат для того, чтобы сравнивать два значения между собой. Результатом этого сравнения является логический тип (`true` или `false`).

Существуют следующие операторы:

Оператор	Значение
<	меньше
<=	меньше – равно
>	больше
>=	больше – равно
==	равно
!=	не равно

Примеры:

```
int x = 10;
( x < 20 )        дает логическое значение    true
( 5 >= x )        дает логическое значение    false
( x == 10 )       дает логическое значение    true
( x != 11 )       дает логическое значение    true
```

Примечание: Сравнение цепей символов всегда делается методом equals

Цепи символов (тип `String`) в Java можно проверять с помощью оператора равенства «==». Однако оператор проверяет только то, ссылаются ли переменные String на ту же область памяти. Как правило, это так, если двум или более переменным String присвоен один и тот же литерал `String`. Однако если для одной переменной String зарезервирована новая память, то оператор равенства больше не работает. На следующем примере показана эта проблема:

Сравнение литералом работает правильно.

```
String s1 = "ABC";
if (s1 == "ABC") System.out.println (String содержит 'ABC'");
```

Создается другая переменная `String` и получает новую область памяти, но то же содержимое. Эту новую форму создания `new` мы подробно рассмотрим в главе ***понятие классов.***

```
String s2 = new String("ABC");
```

Оператор if рассмотрим в следующей главе.

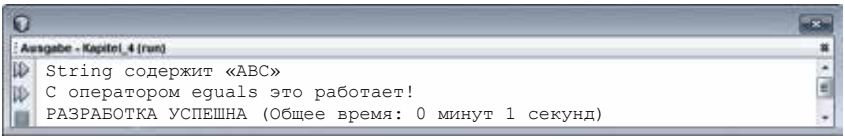
```
if (s1 == s2) System.out.println("с оператором == это работает!");
```

Это сравнение теперь больше не работает!

```
if (s1.equals(s2)) System.out.println("С оператором equals это работает!");
```

Метод equals сравнивает содержимое обеих переменных Strings и выводит `true` или `false`.

После запуска появляется следующий вывод на экран:



4.2.2 Логические операторы

Логические операторы связывают логические состояния (`true` или `false`) друг с другом, результатом также является логическое значение (`true` или `false`).

Пример:

Значение должно быть в ограниченном диапазоне. Для этого два сравнения связываются логическим оператором.

```
int x = 10;

System.out.println( (x > 0) && (x < 100) );
```

`true` или `false`

`true` или `false`

Логический оператор И `&&` связывает оба состояния. Только если оба состояния `true`, результатом также будет `true`.

Существуют следующие операторы:

Оператор	Значение
&&	И
||	ИЛИ
!	ОТРИЦАНИЕ (НЕ)

При связке И результат будет только тогда `true`, если оба оператора `true`, в противном случае будет `false`. При связке ИЛИ результат будет `true`, если хотя бы один (или оба) операнда `true`. Операнды это выражения слева и справа от операторов.

Примеры:

```
int x = 10;
int y = 20;
( x < 20 ) && ( y > x )     дает логическое значение    true
!( x > 20 )                 дает логическое значение    true
( x > 20 ) || ( x > y )     дает логическое значение    false
```

Операнд 1 Операнд 2

Ниже представлены таблицы связок (таблицы истинности) для отдельных операторов:

И	*true*	*false*
true	**true**	false
false	false	false

ИЛИ	*true*	*false*
true	**true**	false
false	false	**false**

ОТРИЦАНИЕ	Результат
true	false
false	true

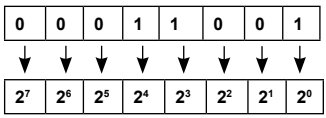
4.3 Битовые и другие операторы

4.3.1 Логические битовые операторы

Все значения переменных в Java в конечном итоге сохраняются в форме нулей и единиц. Тип данных целых чисел `byte`, например, имеет размер 1 бит и необходимо 8 бит, чтобы сохранить ноль или единицу. Поэтому такие числа представлены в так называемой двоичной (бинарной) системе. В двоичной системе счисления есть два числа: ноль и единица. С помощью них можно составлять числа.

Пример:

```
byte x = 25;
Битовое представление числа 25:
```



Двоичная система

$0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3$
$+ 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
$= 0 + 0 + 0 + 16 + 8 + 0 + 0 + 1 = 25$

Битовые операторы работают с двоичной запоминающей матрицей. Они обрабатывают по двоичным разрядам, то есть бит за битом. **Однако битовые операторы работают только с целочисленными значениями!**

Существуют следующие операторы:

Оператор	Значение
&	побитовое И
|	побитовое ИЛИ
^	побитовое исключающее ИЛИ
~	побитовое ОТРИЦАНИЕ

Примеры:

```
short x = 11;     0000000000001011
short y = 9;      0000000000001001

(x & y)           0000000000001001
(x | y)           0000000000001011
(x ^ y)           0000000000000010
(~x)              1111111111110100
```

Биты связываются по отдельности оператором И.

Биты связываются по отдельности оператором ИЛИ.

Все биты связываются по отдельности оператором **исключающего ИЛИ.** Это значит, что результат будет `true` (1) тогда, когда только один операнд `true` (1).

Все биты отрицаются, то есть переворачиваются.

Примечание: Использование битовых операторов

К примеру, IP-адреса в адресном пространстве сетевого оборудования собираются с помощью так называемых подсетей. Соответствующие IP-адреса идентифицируются с помощью побитовой связки И маской подсети.

4.3.2 Битовые операторы сдвига

Операторы сдвига << и >> (и >>>) также работают в двоичной матрице и сдвигают двоичную комбинацию целочисленных значений на неограниченное количество позиций вправо или влево. На примере показан этот сдвиг битов.

Примеры:

```
short x = 11;     0000000000001011
x = x << 2;       0000000000101100

short x = 11;     0000000000001011
x = x >> 2;       0000000000000010
```

Биты сдвигаются на две позиции влево. Справа добавляются нули.

Биты сдвигаются на две позиции вправо. Слева добавляются нули.

Примечание:

С математической точки зрения сдвиг битов влево на x позиций означает умножение числа на 2^x. Соответственно сдвиг вправо означает деление целого числа на 2^x.

Для продвинутых пользователей:

Отрицательные числа представлены в так называемом дополнительном двоичном коде. Это означает, что все биты соответствующего положительного числа сдвигаются, и дополнительно к ним добавляется 1. Это связано с внутренней возможностью представления чисел. Поэтому сдвиг битов отрицательных чисел происходит по-другому. Например, при сдвиге вправо слева добавляются не нули, а единицы.

```
short x = -5;  1111111111111011     Дополнительный код
x = x >> 2;    1111111111111110      -2 в дополнительном коде

Альтернативно: den оператор >>>     (добавляет нули слева)
x = x >>> 2;   0011111111111110     Положит.число: 1073741822
```

4.3.3 Преобразование типов оператором CAST

Переменной данных long можно легко присвоить значение другой переменной целого числа. Это присваивание включает так называемое имплицитное (автоматическое) преобразование типа данных. На следующих строках присвоения представлены возможные имплицитные преобразования:

Примеры:

```
byte b = 10;
char c = 'a'
long l;
double d

l = b;
l = c;
d = l;
d = c;

d = 1
l = d;
```

При преобразовании символа целочисленного значения используется юникод символа – в данном случае l получает значение «97»

При преобразовании символа в число с плавающей запятой также используется юникод символа – в данном случае d получает значение «97.0».

ВНИМАНИЕ: Эта конвертация не может быть имплицитной, так как возможна потеря данных.

На вышеприведенных примерах видно, что имплицитное преобразование возможно тогда, когда оно не влечет потерю данных. Если, несмотря на возможную потерю данных,

необходимо произвести конвертацию, существует возможность эксплицитного преобразования – с помощью так называемых операторов CAST. Эксплицитное преобразование происходит путем указания желаемого типа данных (в скобках) перед преобразуемым значением или преобразуемой переменной.

Примеры:

```
byte  b;
char  c;
long l = 97;
double d = 98.25;
b = (byte)l;
System.out.println("Результат: " + b);
c = (char)l;

System.out.println("Результат: " + c);

l = (long)d;

System.out.println("Результат: " + l);

c = (char)d;

System.out.println("Результат: " + c);
```

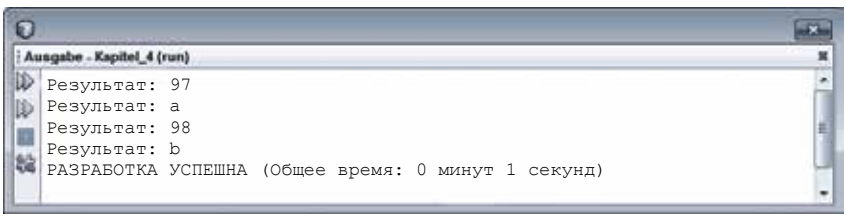
Значение long преобразуется в значение byte.

Значение long преобразуется в значение char.

Значение double преобразуется (с потерей данных) в значение long.

Значение double преобразуется (с потерей данных) в значение char.

После запуска преобразования выглядят на экране так:



ВНИМАНИЕ:

Вышеприведенные примеры демонстрируют, что возможны преобразования целочисленных значений и чисел с плавающей запятой (с потерей данных). Однако невозможно преобразовать логические типы в целочисленные значения, как привыкли это делать, например, программисты C/C++.

4.3.4 Присваивание и связанное присваивание

Оператор присваивания «=» нам уже известен из предыдущих глав. Он работает правильно. Тем не менее, имеет смысл еще раз подробно изучить этот оператор. Слева от оператора присваивания всегда должно стоять так называемое **левое значение**. Оно подразумевает переменную.

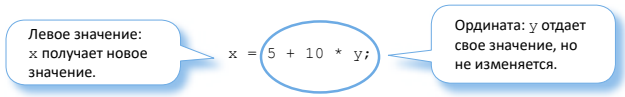
Пример:

```
int x;
x = 5;    //корректное присваивание
5 = x;    //некорректное присваивание. 5 не левое значение.
```

С правой стороны от оператора присваивания стоит так называемая **ордината**. Это значение может быть переменной или выражением.

Пример:

```
int x;
int y = 10;
```



Примечание:

Переменная, находящаяся в ординате, не изменяет свое значение. Она лишь отдает свое значение для присвоения.

Связанные присваивания

Кроме простого присваивания может быть присваивание, связанное с другим оператором. Так выражение можно сократить. Всегда ли это рекомендуется, остается не решенным вопросом. Читабельность исходного кода от этого не повышается.

Примеры:

Обычное выражение	Связанное присваивание
`x = x + 10;`	`x + = 10;`
`y = y / 5;`	`y /= 5;`

Следующие операторы могут быть связаны:

`+ , - , * , / , % , & , | , ^ , << , >> , >>>`

4.4 Ранг оператора

Из уроков математики известно, как действует порядок вычисления. Этот принцип также действует в Java. Например, мультипликативный оператор имеет больший приоритет, чем оператор сложения. Также все остальные операторы имеют свои приоритеты. Так образуется последовательность обработки выражения. Все рассмотренные до настоящего момента операторы перечислены в следующей таблице с их рангом (приоритетом).

Ранг	Описание	Оператор
14	Точечный оператор	.
14	Индексные скобки	[ ]
14	Запуск метода	( )
13	Инкремент/Декремент	++ и --
13	Конвертация (оператор CAST)	(ТИП)
13	Логическое отрицание	!
13	Побитовое отрицание	~
13	Знак	+ и -
12	Умножение	*
12	Деление	/
12	Оператор модуля	%

Ранг	Описание	Оператор
11	Сложение	+
11	Вычитание	-
10	Битовые операторы сдвига	<<, >>, >>>
9	Меньше (меньше-равно)	< (<=)
9	Больше (больше-равно)	> (>=)
9	Тип времени прохождения цикла	istanceof
8	Равно	==
8	Не равно	!=
7	Побитовое И	&
6	Побитовое исключающее ИЛИ	^
5	Побитовое ИЛИ	|
4	Логическое И	&&
3	Логическое ИЛИ	||
2	Условный оператор	? :
1	Присваивание	=
1	Связанные присваивания	+= -= *= /= %= &= |= ^= >>= <<=

Ниже приведен схематичный пример, как компилятор перевел бы выражение. При этом учитываются приоритеты операторов.

Пример:

```
int x = 0;
boolean b = false;

System.out.println(!b && x++ > 1 || x << 1 > 1);
                   ( true && 1 > 1  || x << 1 > 1 )
                   ( true && 1 > 1  ||    2  > 1 )
                   ( true && false  ||      true )
                        ( false     ||      true )
                              ( true )
```

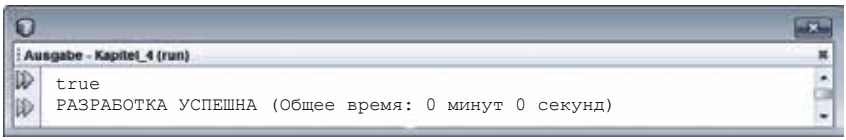
Приоритет 13: ! и ++

Приоритет 9: <<

Приоритет 4: &&

Приоритет 3: ||

После запуска экран подтверждает произведенный выше «расчет»:



Примечание:

Проблему приоритетов можно избежать, если в скобках указать последовательность выполнения операций.

5 Селекция и итерация

Селекция (выбор) и итерация (повторение) это две очень важные конструкции языка программирования. В предыдущих главах некоторые проблемы уже были решены без применения этих конструкций, однако большие и постоянно усложняющиеся программы не могут обходиться без селекции и итерации.

5.1 Селекция

5.1.1 Представление выбора с помощью блок-схемы программы

Формулирование задачи:

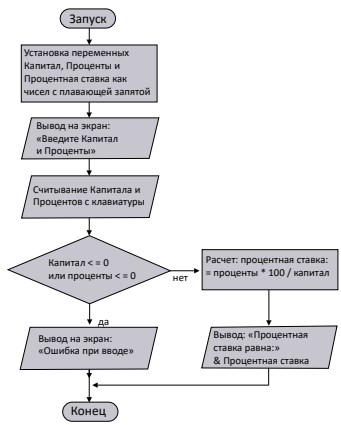
Необходимо рассчитать процентную ставку при заданном капитале и процентах.

Формула: $$\text{Процентная ставка} = \frac{\text{Проценты } 100}{\text{Капитал}}$$

Программа может выдать ошибку, если значение капитала равно нулю (деление на ноль не разрешается). Также не будет смысла, если для капитала или процентов заданы отрицательные значения.

Возможное решение:

Программа распознает, если заданы нулевые или отрицательные значения, и в таком случае не производит расчет. Проблема сначала фиксируется схематически блок-схемой программы PAP[1]. Далее идет реализация в Java.



Такой вид представления называется **блок-схемой программы PAP.** Символы соответствуют стандарту DIN 66001. Это одна из нескольких возможностей представления алгоритма (см. также приложение).

[1] Блок-схема программы была создана с помощью программы PapDesigner. Это программное обеспечение было разработано специально для обучения в области IT и доступно для бесплатного скачивания в Интернет.

5.1.2 Одиночный выбор с помощью оператора if

Возможность выбора (селекция) в Java реализуется с помощью ключевого слова `if`. Оператор `if` это одиночный выбор, так как при выполнении условия выполняется только одна или несколько команд. Если условие не выполняется, то ничего не происходит.

Синтаксис в Java:

```
Оператор if ( условие );
```

Если условие выполняется (логично `true`), выполняется команда.

или

```
if ( условие );

Оператор_1;
    :
Оператор_N;
}
```

Если условие выполняется, выполняется любое количество команд.

Примечания:

► Условие стоит в простых скобках. Несколько взаимосвязанных команд оформляются в фигурные скобки.

► Что такое условие? Условие это выражение, имеющее логическое состояние (`true` либо `false`). Условием может быть, например, сравнение.

► **После условия нет точки с запятой!**

Примеры одиночного выбора с помощью оператора if:

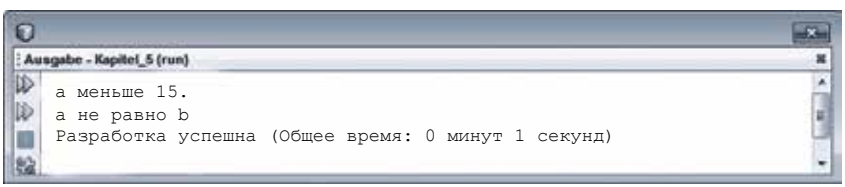
```
int a = 10;
int b = 15;

if ( a < 15 )System.out.println("a меньше 15.");

if ( a != b ) {


System.out.println("a не равно b.");
System.out.println();
}

if ( a > b ) System.out.println("a больше b.");
```

Условие выполнено: Команда выполняется.

Условие выполнено: Команды выполняются.

Условие не выполнено: Команда не выполняется.

После запуска экран выглядит так:



5.1.3 Двойной выбор с помощью оператора if-else

В некоторых случаях следует иметь альтернативу, если условие является неверным. В таких случаях можно применять так называемый двойной выбор с помощью оператора if-else.

Синтаксис в Java:

```
Оператор if ( условие ); else Оператор;
```

или

```
if ( условие ) {
    Оператор_1;
        :
    Оператор_N;
}
else {
    Оператор_1;
        :
    Оператор_N;
}
```

Если условие не выполняется, то один или несколько команд выполняются после else.

Примеры двойного выбора с помощью оператора if-else:

```
int a = 20;
int b = 20;
```

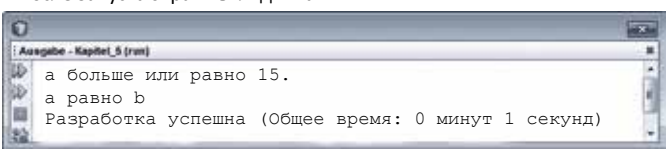
Условие не выполнено: Выполняется команда else.

```
if (a < 15)System.out.println ("a меньше 15.");
else System.out.println("a больше или равно 15.");
```

Условие не выполнено: Выполняются команды else.

```
if (a != b)
System.out.println("a не равно b.");
System.out.println();
}
else {
System.out.println("a равно b.");
System.out.println();
}
```

После запуска экран выглядит так:



Использование блок-схемы программы из исходной ситуации

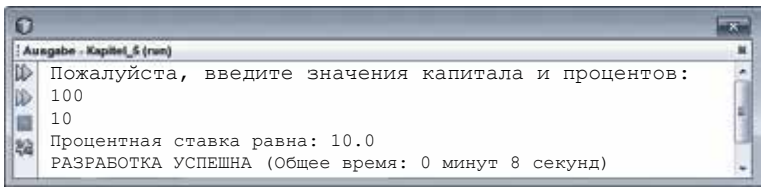
С помощью двойного выбора формулирование задачи теперь также можно использовать из исходной ситуации:

```
package kapitel_5;
import java.io.*;
public class выбор {
  public static void main(String[] args) throws IOException {
     ввод BufferedReader   = new BufferedReader(new
                    InputStreamReader(System.in));
     double капитал;
     double процентная ставка;
     double проценты;
```

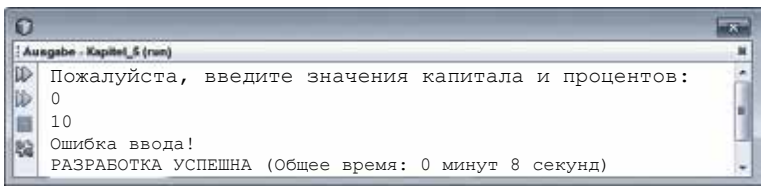
```
System.out.println("Пожалуйста, введите значения капитала и процентов:");

капитал = Double.parseDouble(ввод.readLine());
проценты = Double.parseDouble(ввод.readLine());

if (капитал <= 0 || проценты <  0)
System.out.println("Ошибка ввода!");

}
else {
      Процентная ставка = проценты * 100 / капитал;
      System.out.println("Процентная ставка равна: " +
      процентная ставка);
      }
   }
}
```

Применение выбора

После запуска экран выглядит так:



или так:



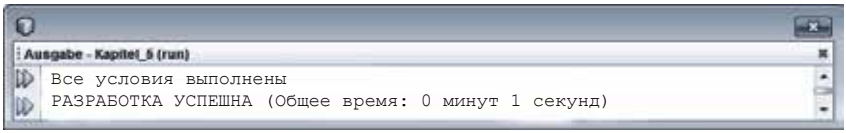
5.1.4 Ветвление с помощью операторов if и if-else

Разумеется, после оператора выбора снова может стоять выбор, так как сам выбор это не что иное, как действующая команда. Ветвление может быть любой глубины. В крайнем случае, глубина ветвления когда-то может быть ограничена проблемой области памяти.

Пример:

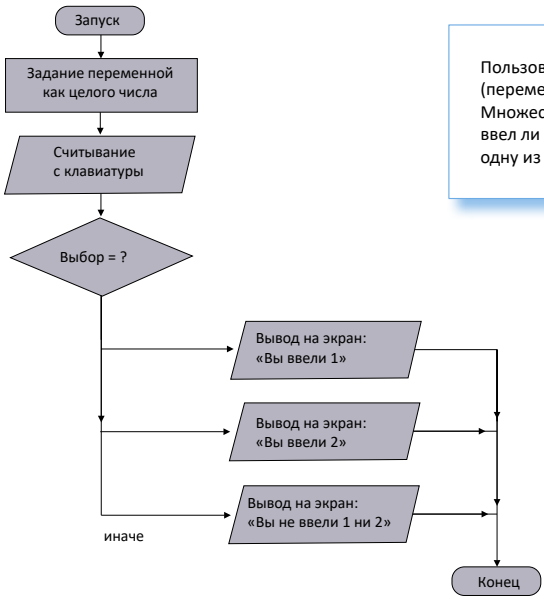
```
int a = 10;
int b = 20;
int c = 1;
if ( a < 15 )
    if ( b > 10 )
     if ( c != 0) System.out.println("Все условия выполнены");
       else System.out.println("Условие 3 не выполнено");
    else System.out.println("Условие 2 не выполнено");
else System.out.println("Условие 1 не выполнено");
```

После запуска экран выглядит так:



5.1.5 Множественный выбор с помощью оператора switch

В некоторых случаях необходимо запросить несколько значений переменной. Это могло произойти, например, вследствие ветвлений с помощью оператора if. Однако в этом случае удобнее использовать множественный выбор с помощью оператора switch. Оператор switch проверяет целочисленную переменную (также char) на определенные значения. Так, целенаправленно выполняются команды. В блок-схеме программы множественный выбор представлен следующим образом:



Пользователь может ввести число (переменную на выбор) с клавиатуры. Множественный выбор определяет, ввел ли пользователь цифру 1, 2, или ни одну из этих цифр.

Синтаксис в Java:

Переменная **var** должна быть целым числом (или выражением, которое имеет целочисленный результат).

```
switch ( var ) {

        case значение_1:
             оператор _1;
                :
             оператор _N;
        break;
        case значение_2:
             оператор _1;
                :
             оператор _N;
        break;
```

Если **var** соответствует значению_1, команды выполняются до `break`.

Если **var** соответствует значению_2, команды выполняются до `break`.

```
        :
        case значение_N:
             оператор_1;
                  :
             оператор_N
        break;

        default:
             оператор_1;
                  :
             оператор_N;
        break;
    }
```

Если **var** соответствует *значению_N*, команды выполняются до break.

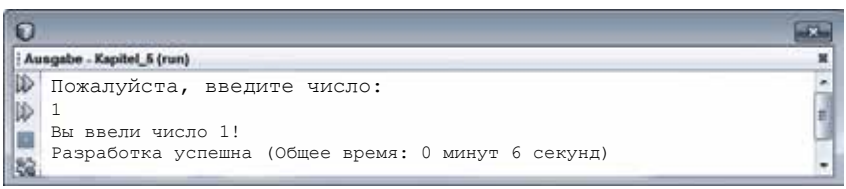
Если **var** не соответствует ни одному из вышеуказанных значений, команды default выполняются до break. Оператор default всегда должен стоять в конце.

break можно пропустить, если это последняя операция.

С помощью оператора switch может быть выполнен пример из блок-схемы программы:

```
public class выбор {
   public static void main(String[] args)throws IOException{

        int выбор;
        ввод BufferedReader = new BufferedReader(new
                                  InputStreamReader(System.in));

    System.out.println("Пожалуйста, введите число (выбор):");
    выбор = Integer.parseInt(ввод.readLine());

    switch (выбор){
          case 1:
                System.out.println("Вы ввели число 1!");
          break;
          case 2:
                System.out.println("Вы ввели число 2!");
          break;

          default:
                System.out.println("Ни 1 ни 2 не введено!");
          break;
    }
  }
}
```

После запуска экран выглядит так:



Примечание:

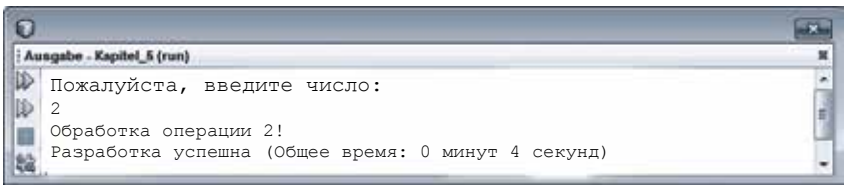
Пропуск оператора `break` приводит к тому, что все остальные операции обрабатываются друг за другом. Это связано с тем, что оператор `switch` проверяет только один раз, введена ли точка входа, затем выполняет все остальные операции до следующего оператора `break`.

Пример:

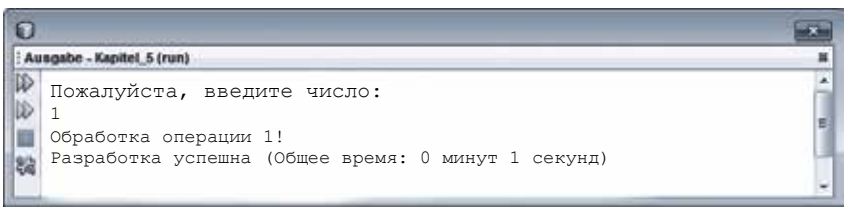
```
switch (выбор) {
        case 1:
              System.out.println("Обработка операции 1!");

        case 2:
              System.out.println("Обработка операции 2!");
        break;

        default:
          System.out.println("Обработка операции default!");
        break;
}
```

Сейчас здесь нет оператора `break`.

После запуска экран выглядит так:



или так:

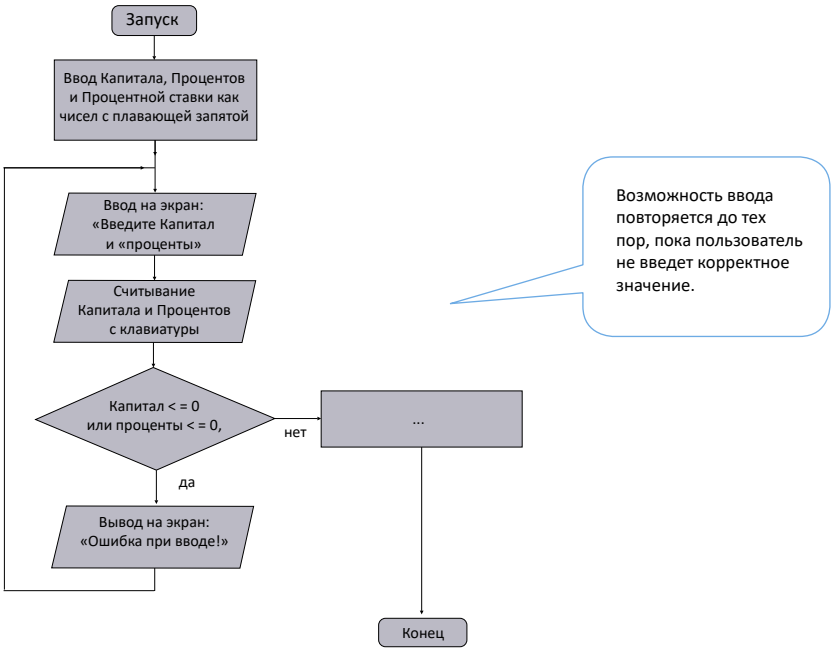


5.2 Итерационные циклы с постусловием и предусловием, циклы со счетчиком

Формулирование задачи:

Программа расчета процентов из подраздела 5.1.1 не должна производить расчет, если заданный капитал равен нулю. В этом случае было бы целесообразно, если бы программа дала возможность ввести заново корректное значение.

В программе PAP это может быть представлено так:



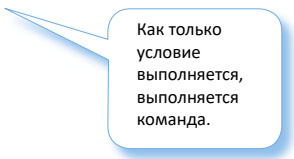
В программе Java есть три возможности достичь такого повторения с помощью так называемых циклов (итераций). Существует цикл `do-while`, цикл `while` и цикл `for`. Еще один вид цикла (цикл `for-each`) мы рассмотрим в главе о массивах.

5.2.1 Цикл do-while

Цикл `do-while` это повторение одного или нескольких операторов до тех пор, пока условие не будет выполнено. Структура условия здесь такая же, как у оператора `if`. Цикл `do-while` называется циклом с постусловием, так как проверка условия осуществляется в конце цикла. Тело цикла (команды внутри цикла) будет выполняться минимум один раз.

Синтаксис в Java:

```
do    команда;    while (условие);
```



Конечно, также действительны:

```
do {
        Команда_1;
        Команда_2;
         :
        Команда_N;
}
while (условие);
```

Выполнение нескольких команд

Примеры цикла do-while:

Необходимо вывести цифры от 1 до 4 на экран:

```
int x = 1;
do {

System.out.println(x);
x = x + 1;
}
   while (x < 5);
```

Ввод повторяется до тех пор, пока вводится слово «Hallo».

```
String s;
ввод BufferedReader = new BufferedReader(new
                      InputStreamReader(System.in));
do {

s = ввод.readLine();

} while (s.equals("Hallo"));
```

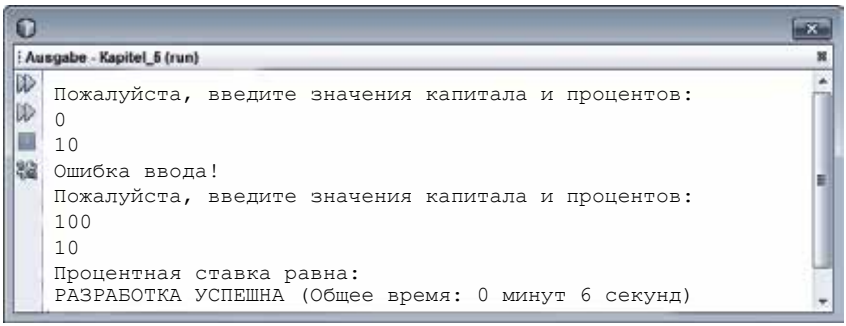
Цепи символов проверяются на равенство с помощью метода `equals`.

С помощью цикла `do-while` теперь можно запускать блок-схему программы из задачи. Некорректный или бессмысленный ввод повторяется.

```
public class Iteration {
   public static void main(String[] args) throws IOException {
   ввод BufferedReader= new BufferedReader(new
                        InputStreamReader(System.in));
        double капитал;
        double процентная ставка;
        double проценты;
        do {
        System.out.println("Пожалуйста, введите значения капитала и
        процентов:");
        капитал = Double.parseDouble(ввод.readLine());
        проценты = Double.parseDouble(ввод.readLine());
        if (капитал <= 0 || проценты <= 0){
             System.out.println("Ошибка ввода!");
        }
        else {
             процентная ставка = проценты * 100 / капитал;
             System.out.println("Процентная ставка равна: " +
                                процентная ставка);
{
```

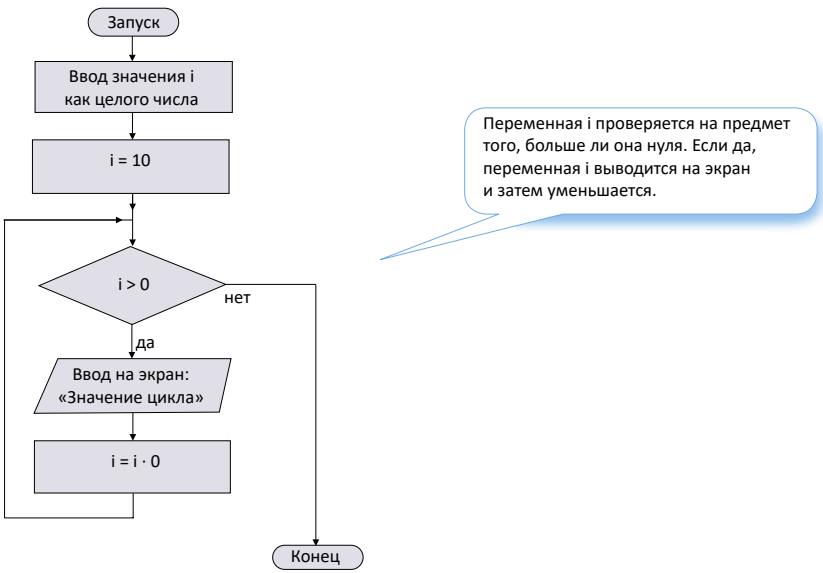
```
} while (капитал <= 0 || проценты <= 0);

   }
}
```

После запуска экран выглядит так:



5.2.2 Цикл while

Цикл `while` – это также повторение одного или нескольких операторов до тех пор, пока условие не будет выполнено. Однако цикл `while` называется **циклом с предусловием**, так как условие проверяется сразу в начале цикла. Таким образом, тело цикла может быть не выполнено ни разу (если условие ложно). На следующей блок-схеме показан цикл с предусловием:



Синтаксис в Java:

```
while       (условие)       команда;
```

Как только условие выполняется, выполняется команда.

Конечно, также действительны:

```
while        (условие) {

             Команда_1;
             Команда_2
             :
             Команда_N;
```

Выполнение нескольких команд

Пример цикла while:

Вышеприведенная блок-схема цикла с предусловием реализуется с помощью цикла `while`.

```
int i;
i = 10;
while (i > 0) {
        System.out.println("Значение цикла: " + i);
        i = i - 1;
}
```

5.2.3 Цикл for

Цикл `for` называется циклом со счетчиком. Как правило, переменная изменяет свое значение от заданного начального значения до конечного значения с некоторым шагом.

Синтаксис в Java:

```
for (инициализация ; условие ; шаг ) команда;
```

Конечно, также действительны:

```
for (инициализация ; условие ; шаг) {
      команда_1;
      команда_2;
      :
      команда_N;
{
```

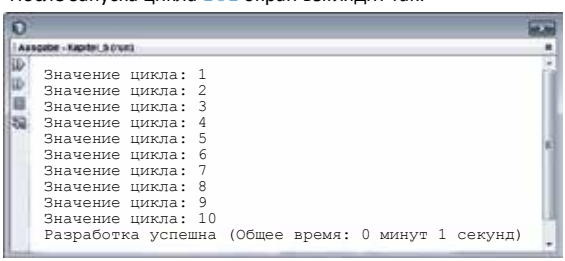
Пример цикла for:

Цикл начинается со значения 1 и заканчивается значением 10. Шаг переменной равен 1.

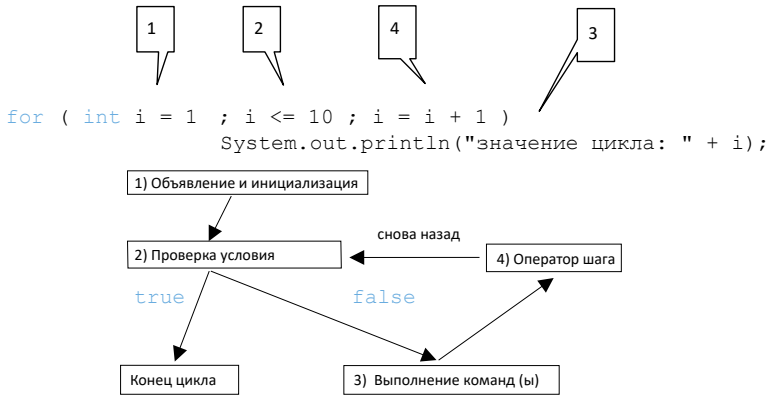
Ключевое слово `for` | Инициализация | Условие | Шаговый оператор

```
for (int i = 1 ; i <=10; i = i + 1 );
                System.out.println("значение цикла: " + i);
```

После запуска цикла `for` экран выглядит так:



Очень важно также понимать временное прохождение цикла for, так как от этого зависит корректная работа цикла.



Однако цикл for может намного больше, чем «просто» считать: по необходимости части цикла можно пропускать или добавлять несколько команд или условий в одну часть.

Так, самый простой цикл выглядит так:

```
for ( ; ; );
```

Бесконечный цикл

Это действительный оператор, который не обязательно применять – это так называемый бесконечный цикл, программа «зависает». Однако, компиляторы распознают данную проблему.

Следующий пример демонстрирует универсальность цикла for:

```
for (int i = 10; i>0 ; System.out.println(i--)) ;
```

Этот цикл считает значения переменной i от 10 до 1 и выдает соответствующие значения. Шаговая часть при этом не имеет шагового оператора, а выводит на экран переменную i с одновременным декрементированием. В теле цикла нет операторов.

Примечание:

Переменная цикла for, как правило, объявляется и инициализируется в части инициализации. Переменная for действительна только в рамках цикла (локального действия). После цикла for переменную уже нельзя использовать.

```
for (int i = 1; i <= 10; i = i + 1)
            System.out.println("Значение цикла: " + i);


i = 10;
```

Сообщение об ошибке компилятора:
Cannot find symbol
Symbol: variable

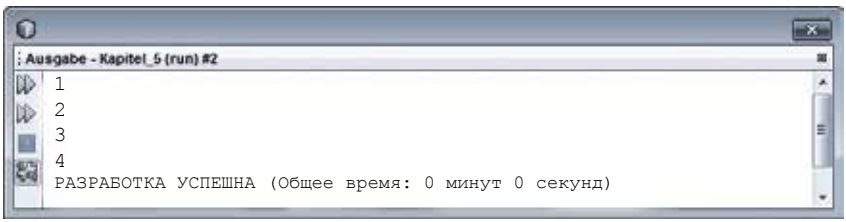
5.2.1 Досрочный выход и пропуск итерации

Все циклы могут быть прерваны независимо от проверки условия. Это происходит с помощью ключевого слова `break`. С помощью ключевого слова `continue` может быть выполнен пропуск итерации.

Пример использования break:

```
for (int i = 1; i <= 10 ; i++) {
      if ( i == 5 ) break;
      System.out.println(i);
}
```

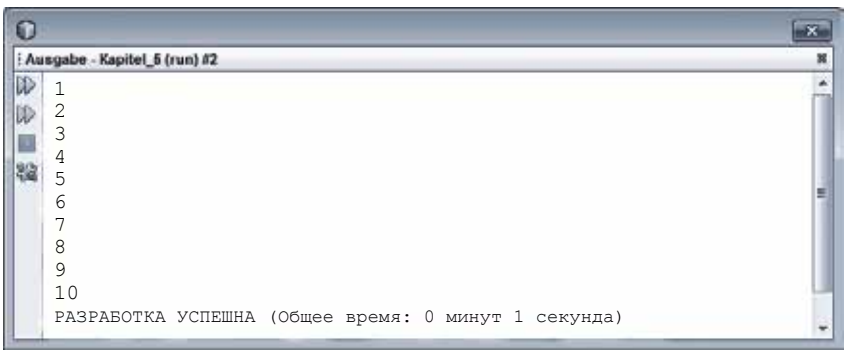
Цикл с ключевым словом `break` выглядит так:



Пример использования continue:

```
for (int  i = 1; i <= 10 ; i++) {
      if ( i == 5 ) continue;
      System.out.println(i);
}
```

Цикл с ключевым словом `continue` выглядит так:



6 Понятие классов в Java

Язык Java является полностью объектно-ориентированным. Однако в предыдущих главах мы не рассматривали объектно-ориентированные темы, а говорили об основах структурного программирования. Это было необходимо, чтобы подготовить базу для других тем. Тем не менее, некоторые аспекты уже были объектно-ориентированы, но были описаны пока лишь поверхностно, что было достаточно для выполнения программы. В этой главе начнется объектно-ориентированное программирование в Java. Под ним понимается специальный вид программирования, который старается применять определенные данные максимально приближенно к реальности. В центре объектно-ориентированного программирования стоят **объект** и **класс**. Класс можно рассматривать как структурный план, с помощью которого образуются объекты. Теперь подробнее рассмотрим понятия объекта и класса.

Что такое объект?

Объект это репрезентация реального или виртуального, четко обозначенного предмета или понятия с использованием программных средств. Объект охватывает все аспекты предмета с помощью атрибутов (свойств) и методов.

Что такое атрибуты и методы?

Атрибуты это свойства объекта. Они полностью описывают предмет. Атрибуты защищены от внешних манипуляций (это называется инкапсуляция). Методы описывают операции, которые можно проводить с объектом (или его атрибутами). Снаружи доступ к атрибутам производится через методы.

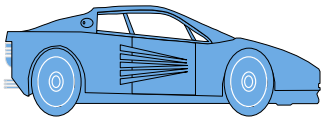
Что такое класс?

Под классом понимается описание структурного плана для объекта с использованием программных средств. Из класса могут производиться объекты (образовываться, создаваться путем инстанцирования).

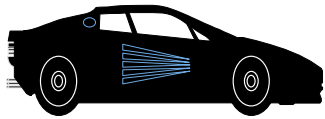
Приведем наглядные примеры для этих несколько абстрактных, но важных определений.

Пример:

Эти гоночные автомобили являются конкретными объектами. Они имеют такие атрибуты, как цвет, мощность в кВт и объем двигателя.

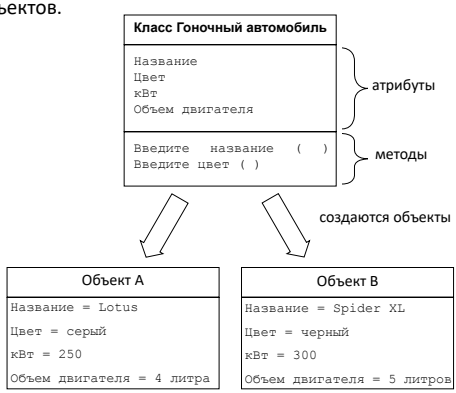


Название:	Lotus
Цвет:	синий
кВт:	250
Объем двигателя:	4 литра



Название:	Spider XL
Цвет:	черный
кВт:	300
Объем двигателя:	5 литров

Оба гоночных автомобиля имеют одинаковые атрибуты. Они отличаются только величиной атрибутов. Например, Spider XL имеет большую мощность, чем Lotus. Можно сказать, что оба автомобиля произведены с помощью одного и того же плана производства. Структурный план, лежащий в их основе, можно обозначить как **класс** `гоночный автомобиль`. Следующее представление классов и объектов соответствует общему представлению классов и объектов.



Примечания

Объектно-ориентированность и новые понятия образуются абсолютно абстрактно в самом начале и почти невозможно представить, как запрограммировать новое программное обеспечение объектно-ориентированно. Здесь поможет только одно: пошаговое изучение аспектов объектно-ориентированного программирования **(ООП)** и их применение на конкретных примерах. Хорошая объектно-ориентированная разработка программ также сильно связана с опытом.

Наряду с измененным пониманием программирования, ООП также имеет преимущества перед структурным или процедурным программированием. Это, к примеру, инкапсуляция данных в объектах или наследование. Инкапсуляция данных означает, что доступ к атрибутам объекта проходит под контролем. Этот контролируемый доступ осуществляется через методы объекта. Так, например,

предотвращается случайное описание важного атрибута объекта с ошибочным значением. Наследование экономит программисту много времени, потому что он может передавать однажды написанные классы другим классам по наследству. Однако полное понятие ООП станет действительно ясно тогда, когда мы рассмотрим главы о понятии классов, наследовании и полиморфизме.

6.1 Первый класс в Java

В этой главе речь идет о конкретном применении класса в Java. Сначала будет описана общая структура класса. При этом на переднем плане стоят, прежде всего, атрибуты и их доступ. Это непосредственно связано с важным аспектом ООП – инкапсуляцией. Далее будут освещены принципы действия и структура методов.

6.1.1 Структура класса в Java

Класс в Java инициируется с помощью ключевого слова **`class`**. Внутри класса (оформленного фигурными скобками) существуют атрибуты и методы, снабженные модификатором доступа (private, public или protected). Отдельные модификаторы имеют свое влияние.

Модификатор доступа private:

Ко всем атрибутам (и методам), отмеченным этим модификатором, нет внешнего доступа. Доступ может осуществляться только с помощью соответствующих (открытых) методов.

Модификатор доступа public:

Ко всем методам (и атрибутам), отмеченным этим модификатором, есть внешний доступ. Эти элементы также называют внешним интерфейсом класса. Коммуникация с классом (или объектом этого класса) производится через этот интерфейс (элементы `public`).

Модификатор доступа protected:

Этот модификатор работает на внешний доступ, как и модификатор `private`, но имеет другую функцию, связанную с темой наследования. Пока рассмотрим только два первых модификатора.

Синтаксис в Java:

Класс может также иметь модификатор доступа `public`. С ним он доступен и для других пакетов. Если класс задан без модификатора `public`, тогда он доступен только в своем пакете.

```
[public] class Название {
        [ атрибуты ]
          [ методы ]
}
```

Можно создавать неограниченное количество атрибутов.

Можно создавать неограниченное количество методов. Дополнительно существуют специальные методы (конструкторы и деструктор, подробнее о них позже).

Первый пример класса

```
package kapitel_6;

class ПервыйКласс {

  public int x = 10;
  private String s = "Привет";
  int без Модификатора = 10;
```

В файле всегда может быть только «основной класс», который называется так же, как и файл. Поэтому другие классы (как `ПервыйКласс`) не могут иметь модификатор `public`.

Определяется класс `ПервыйКласс`. В классе есть три атрибута. Один атрибут „`public`“, второй атрибут „`private`“, и третий атрибут без модификатора.

```
public class Основнойкласс {
   public static void main(String[]

      ПервыйКласс объектСсылка;
      объектСсылка = new ПервыйКласс

      объектСсылка.x = 20;

            объектСсылка.s = "новый";
            объектСсылка. безМодификатора = 10;

   }
}
```

Создается объект класса.

Доступ к атрибуту public работает.

Этот доступ запрещен, так как s является частным атрибутом!

Доступ к атрибуту без модификатора также работает!

В этом первом примере следует пояснить некоторые аспекты:

► Определяется новый класс, но не внутри «основного класса» (это также может иметь смысл, подробнее об этом позже). Имя нового класса на выбор (см. Конвенции для имен переменных).

► Создание объекта нового класса происходит примерно так же, как создание переменной простого типа данных. Однако вместо типа данных используется имя класса. В принципе, класс это вновь созданный (пользовательский) тип данных. На первом этапе создается так называемая ссылка на класс:

```
ПервыйКласс объектСсылка;
```

Создать ссылку

► Далее этой ссылке может быть присвоен конкретный объект в оперативной памяти. С помощью оператора new такой объект создается в оперативной памяти и присваивается ссылке:

```
объектСсылка = new первыйКласс ();
```

Создать объект new в оперативной памяти и присвоить ссылке.

► Доступ к атрибуту объекта осуществляется через точечный оператор.

```
объектСсылка.x = 20;
```

Точечный оператор

Доступ к атрибутам Public осуществляется напрямую. Однако атрибуты public противоречат главному принципу ООП – также см. следующее пояснение.

► К атрибутам Private не может быть внешнего доступа. После запуска программы появляется следующая ошибка компилятора:

```
Ошибка:
s has private access
```

```
объектСсылка.s = "новый";
```

Строка ошибки

Данная ошибка обращает внимание на то, что была попытка доступа к частному атрибуту. Компилятор это предотвращает, так как частные атрибуты не имеют внешнего доступа. Это соответствует главному принципу объектно-ориентированного программирования – инкапсуляции. Конечно, нельзя решить проблему тем, что предусмотреть модификатор доступа `public` для всех атрибутов, таким образом этот главный принцип был бы нарушен. Вместо этого необходимо разрабатывать другие подходящие механизмы для контролируемого доступа к атрибутам. С помощью методов, перечисленных в следующем подразделе, эту проблему можно решить.

Примечание:

Атрибуты, заданные без модификатора, (как атрибут `безМодификатора` в вышеприведенном примере), в принципе, ведут себя как частные атрибуты. Однако для классов внутри пакета доступ предоставлен так, как если бы атрибут был объявлен открытым (`public`).

6.1.2 Типы значений и ссылок

До настоящего момента переменные простого типа данных (типы значений) просто создавались и могли быть использованы. Это было связано с тем, что эти переменные хранились в определенной области памяти – памяти **STACK**[1]. При введении классов в Java в игру вступает новый тип, а именно тип ссылки. Все объекты, образуемые от классов, хранятся в другой области памяти – **HEAP**. Теперь, чтобы получить доступ к объекту, должна быть создана ссылка на объект. Это может быть выполнено в два этапа (см. пример выше) или в один этап, как на примере:

```
ПервыйКласс объектСсылка = new ПервыйКласс ();
```

Ссылка класса `ПервыйКласс`

С помощью оператора `new` создать объект значение в динамичной памяти HEAP и присвоить ссылке

Программа garbage collector

Все переменные типа значений сохраняются в STACK и затем автоматически удаляются, когда становятся недействительными. В отличие от них объект сохраняется в динамичной памяти HEAP и удаляется только тогда, когда больше нет ссылки на этот объект, так как на один и тот же объект может существовать много ссылок. Это удаление производится с помощью так называемого «сборщика мусора» **garbage collector**. Это механизм распознает объекты без ссылок и удаляет их из памяти. В других языках программирования, например в C++, программист должен был самостоятельно производить это удаление, что приводило большому количеству ошибок.

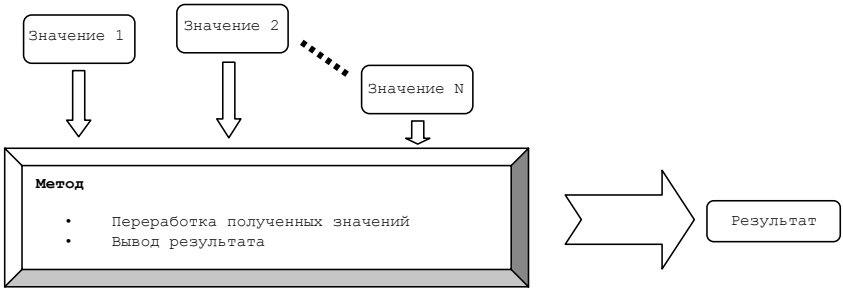
6.2 Методы в Java

Из первых примеров класса было понятно, что доступ к атрибутам должен осуществляться через механизм, который дополнительно предлагает возможности контроля. Например, было бы бессмысленно присвоить атрибуту *PS* гоночного автомобиля отрицательное значение. В этом случае метод должен был бы предотвратить это бессмысленное присвоение.

6.2.1 Структура метода

С технической точки зрения метод это не что иное, как функция. Он запускается и выполняет определенную функцию. Наглядно метод можно представить как аппарат, который получает ввод (значения) и производит результат.

[1] Память STACK это определенная область памяти, используемая для локальных (ограниченно действующих) переменных. Она работает по принципу LIFO (Last in first out). Область динамического распределения памяти, напротив, это область, где предоставляется место для объектов.



Во всех предыдущих примерах методы уже использовались (интуитивно). Самым важным при этом был статический метод `main`. Этот метод вызывается и выполняется при запуске программы. В нем видно, что в рамках метода код программируется так же, как и в предыдущих главах.

Обзор самых важных свойств метода:

► Методы имеют идентификатор (имя), который создается так же, как у переменных. После идентификатора всегда стоит пара круглых скобок (либо пустые, либо с параметрами).

► Методы имеют так называемое тело, в котором метод программируется. Тело заключается в фигурные скобки.

► Методы могут принимать неограниченное количество значений (параметров).

► Методы могут возвращать значение.

Первый простой пример метода

```
class Person {

        private String имя


    public void initName() {

            имя = "Кайзер";
    }

}

public class Основнойкласс {
     public static void main(String[] args)

          Лицо однолицо = new Person

              одноPerson.initName();
     }
}
```

Класс Person применяется по отношению к любому лицу. Для удобства сначала создается только один атрибут (имя).

Метод `initName()` инициализирует частный атрибут `name` для класса Person.

Создается объект класса.

Метод `initName()` запускается с помощью точечного оператора.

На примере видно, что метод `initName()` может быть запущен объектом класса Person. Это связано с тем, что метод является «открытым». Сам метод типа `void`. Это значит, что метод не возвращает значение на место запуска – то есть тип данных `void` не доступен для возврата (подробнее об этом позже). Метод также имеет пару круглых скобок. Поэтому он не принимает значения (подробнее об этом также позже). Так, на примере выше показана самая простая форма метода.

В следующем примере добавляется еще один метод, который выводит имя на экран:

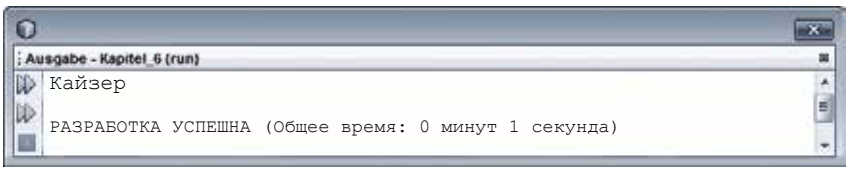
```
class Person {
      :
      :
      public void напишитеИмя() {
             System.out.println(имя);
             System.out.println();
      }
}
```

Метод напишитеИмя() выводит имя на экран.

```
public class Основнойкласс {
       public static void main(String[] args) {

              Person одноPerson = new Person
              одноPerson.initName();
              одноPerson.напишитеИмя();
   }
}
```

Запуск методов путем ввода имени и пустых скобок

После запуска экран выглядит так:



6.2.2 Возвращаемое значение метода

Метод напишитеИмя() из примера выше выводит имя на экран. Однако теперь имя лица должно быть присвоено другой переменной String. Для этого метод должен иметь возможность возвращать значение. Это может происходить путем следующей адаптации:

Тип данных возвращаемого значения метода

```
public String возвратИмя() {
       return имя;
}
```

Тип данных возвращаемого значения метода

Возврат значения с помощью return

С помощью данного метода имя может быть присвоено переменной String:

```
public class Основнойкласс {
   public static void main(String[] args) {

       Person одноPerson = new Person(
       String Имя;
       одноPerson.initName();
       одноимя = одноPerson.возвратИмя ();
       System.out.println(одноИмя);
```

Метод возвращает имя.

Вышеуказанное присваивание идет потому, что после запуска метода он возвращает значение и затем это значение стоит на месте запуска метода:

```
одноИмя = одноPerson.возвратИмя ();

одноИмя =       "Кайзер";
```

После запуска

В целом структуру метода с возвращением значения можно описать так:

```
Модификатор тип данных возвращаемого значения идентификатор ()
{
      оператор_1;
      оператор_2;
      :
      оператор_N;

      return значение;
}
```

Тип данных возвращаемого значения и возвращаемое значение должны совпадать.

На следующем примере показан метод, который имеет тип данных возвращаемого значения `double`, однако возвращает `String`. Разумеется, эти значения не совпадают.

```
public double плохой
       String возврат = "Привет";
       return возврат;

}
```

Ошибка компилятора:

```
incompatible types
required:double
found:java.lang.String
```

Примечания:

- При возврате типа значений с помощью `return` создается копия возвращаемого значения и передается на место запуска.
- При возврате типа ссылок с помощью `return` возвращается ссылка.

6.2.3 Локальные переменные

Переменные, созданные в методе, действительны только в рамках метода. Если запускается один метод, происходит переработка этих переменных и по окончании они «удаляются» (то есть они имеют локальное действие). Объекты также могут создаваться в методе. Они удаляются с помощью процесса **garbage collector**, когда на них больше не существует ссылки. Если необходимо обеспечить «выживание» объекта, то следует вернуть ссылку на объект. На следующем примере показана эта проблема:

```
package глава_6;
class Person { ... }
class тест {
      public Person возвратPerson() {
            Person одноPerson = new Person();
            return одноPerson;
      }
      public void локальныеПеременные();
            int x = 10;
            double d = 1.25;
      }
}
```

Этот метод производит объект из динамичной памяти с локальной ссылкой. Однако ссылка возвращается методом.

Этот метод создает две локальные переменные в памяти STACK. После вызова методов эти переменные снова удаляются.

```
public class Основнойкласс {
        public static void main(String[] args) {

    Test одинTest =  new test()
    Person новоеPerson = одинTest.возвратPerson();

    новоеPerson.initName();
    новоеPerson.напишитеИмя();

    одинTest.локальныеПеременные();
    }
}
```

С помощью метода `возвратPerson()` создается лицо и присваивается ссылка `новомуPerson`. Даже после вызова объект-`Person` действителен, так как на него есть ссылка.

Объект-лицо может быть использован далее

Метод вызывается, и вызываются две локальные переменные. После вызова локальные переменные удаляются.

Примечание

Все переменные, которые уже были созданы в методе `main`, естественно, локального действия. Однако так как метод `main` в принципе является «основной программой», переменные сохраняют свое действие в течение всего процесса.

6.2.4 Передаваемые параметры метода

Переменная может не только возвращать значение, но и принимать его. Это происходит с помощью так называемых параметров, которые могут быть заданы в круглых (прежде пустых) скобках метода. Многие параметры отделяются запятыми.

В целом структуру метода можно описать так с помощью возвращаемого значения и параметров:

```
Модификатор тип данных возвращаемого значения идентификатор
(тип парам_1,тип парам_2,…) {

        оператор_1;

        оператор_2;
        :
        оператор_N;

        return значение;
}
```

В параметрах (или передаваемых переменных) сохраняются значения, которые передаются методу. У каждого параметра есть тип данных и имя. Параметры отделяются запятыми.

Следующая программа демонстрирует использование параметров:

```
class Person {

        private String имя;
        private double вес;

public void задайтеИмя(String имяПараметр) {

        имя = имяПараметр;
}
```

Метод `задайтеИмя()` класса `Person` может перенимать `String` и присваивать атрибуту имя.

Метод задайтеВсеЗначения () класса Person может перенимать String и значение double и присваивать их соответствующим атрибутам.

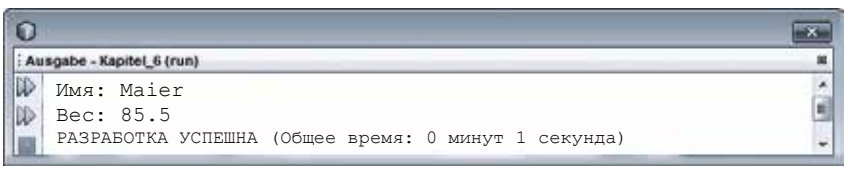
```
public void задайтеВсеЗначения (String имяПарам,double весПарам){
        задайтеИмя(имяПараметр);
        вес = весПараметр;
    }

    public String возвратИмя() {
           return имя;
    }
    public double возвратВес() {
           return вес;
    }
}
```

Существующий метод задайтеимя() прост в применении. Методы того же класса, конечно, могут запускаться в методах.

Запускаются методы и передаются значения

```
   public class Основнойкласс {
          public static void main(String[]args){
          Person одноPerson = new Person();
          одноЛицо.задайтеИмя("Кайзер");
          однолицо.задайтеВсеЗначения("Майер",85.5);
          System.out.println("имя: " + одноPerson.возвратИмя());
          System.out.println("вес: " + одноPerson.возвратВес());
      }
   }
```

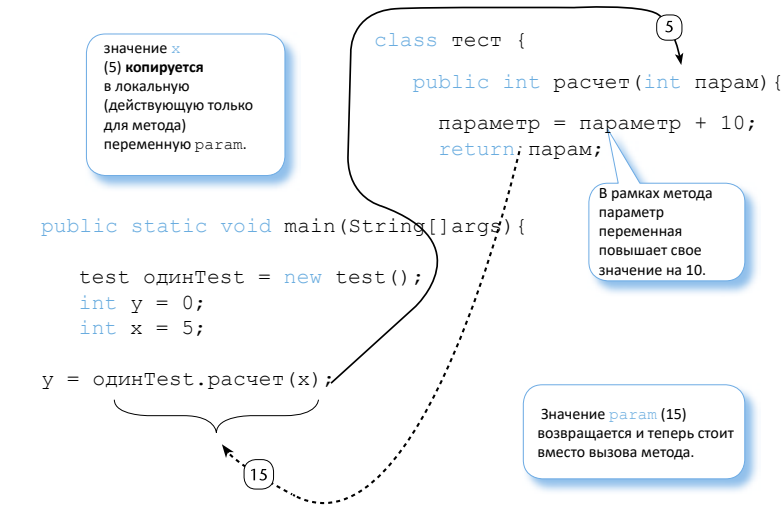
После запуска экран выглядит так:



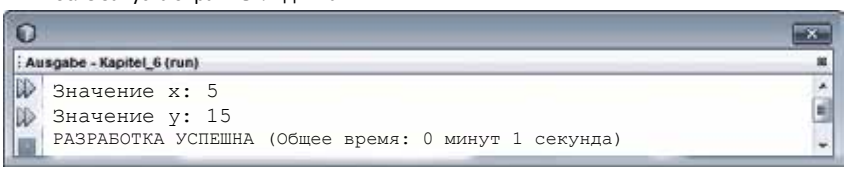
Примечание

Метод может иметь неограниченное количество параметров. Параметры – это не что иное, как локальные переменные, в которых сохранены значения, которые передаются методу при запуске. С параметрами можно работать так же, как и со всеми остальными локальными переменными метода.

Понимание передаваемых параметров и возвращаемых значений метода очень важно, поэтому еще раз рассмотрим их связь на графике:



После запуска экран выглядит так:



Переменная y изменила свое значение. Она получила возвращаемое значение метода. **Переменная x не изменила свое значение.** Она только предоставила свое значение для копирования. Этот вид передачи называется **вызовом по значению**. В отличие от других языков в Java есть только этот вид передачи – в C++ или C# параметр может также передаваться как ссылка.

Передача параметров типов значений

С типом ссылок передача параметров на первый взгляд происходит несколько иначе, чем с типом значений. Когда передается тип ссылок, ссылка копируется и таким образом как параметр, так и переданная ссылка ссылаются на тот же самый объект в памяти.
На следующем примере показано отличие от передачи типа значений:

```
class ссылка {

   private int x;

   public int gibwert ()}
      return x;
   }
```

Класс Ссылка – простой пример передачи ссылки. Можно сохранить значение целого числа.

```
public void задайтеЗначение(int параметр) {
        x = параметр;
    }
}
class ссылкаПараметр {

   public void передача_1(ссылка параметр){

          парам.задайтеЗначение(10)
}
public void передача _2(ссылка параметр){


        параметр = new ссылка();
        параметр.задайтеЗначение(30);
    }
}

public class основнойКласс {
      public static void main(String[] args) {

          ссылкаПараметр одинТест = new ссылкаПараметр();
          ссылка однаСсылка = new ссылка();


        однаСсылка.задайтеЗначение(0);


        одинТест.передача_1(однаСсылка);
        System.out.println(однаСсылка.возврат());


одинТест.передача_2(однаСсылка);
System.out.println(однаСсылка.возврат());

    }
}
```

Класс ссылкаПараметр исполняет метод, который имеет ссылку на параметр.

Переданная ссылка вызывает метод задайтеЗначение().

Параметру ссылки динамично присваивается новый объект и затем вызывается метод задайтеЗначение().

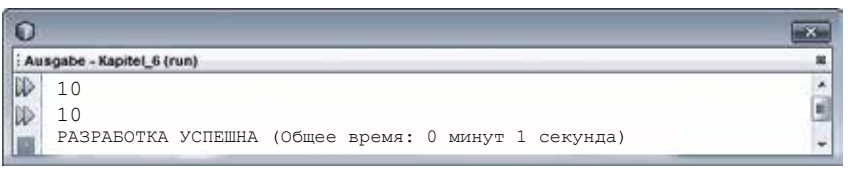
Инстанциируются объекты классов ссылкаПараметр и ссылка.

Передача методу передача_1()

Объект ссылка получает новое значение (ноль).

Передача методу передача_2()

После запуска экран выглядит так:



По вызовам видно, что передача ссылки хоть и допускает изменения, однако нельзя инстанцировать новые объекты, которые затем могут быть присвоены переданной ссылке.

Примечание

Хотя передача типа ссылки происходит по принципу ***call by value***, возможно изменение объекта, ссылка которого была передана. Этим такая передача похожа на ***call by reference***, хотя это и не настоящий вызов ссылки.

6.2.5 Перегрузка методов

Перегрузка методов означает, что методы имеют то же самое имя и выполняют похожие функции, однако для разных передаваемых параметров. Перегрузка это важное свойство, которое применяется, прежде всего, у конструкторов (см. следующую главу).

Пример:

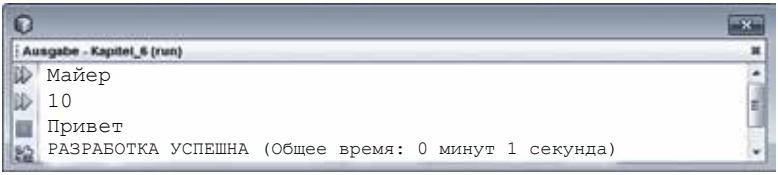
Необходимо написать методы, которые выводят передаваемые параметры на экран. Для разных типов данных исполняется собственный метод.

```
class Person {
      private String имя = "Майер";
      public String возвратИмя() {
              return имя;
      }
}
class перегрузка {
 public void вывод(Person PersonПараметр) {
   System.out.println(PersonПараметр.возвратИмя());
}
public void вывод (int intParam) {
       System.out.println(intParam);
}
public void вывод (String stringParam) {
       System.out.println(stringParam);
}
}
public class Основнойкласс {
public static void main(String[] args) {
          перегрузка одинTest = new перегрузка ();
          лицо одноPerson = new лицо();
          одинTest.вывод(одноЛицо)
          одинTest.вывод (10);
          одинTest.вывод("Привет");
}
}
```

Три метода с одинаковым идентификатором, но разными типами параметров

В зависимости от передаваемых данных компилятор распознает корректный метод.

После запуска экран выглядит так:



Преимущество:

Программист может назвать похожие функции одним именем (метода) – так программирование упрощается и становится более структурированным.

Примечание:

Перегрузка методов не ограничена – однако компилятору всегда необходимо однозначно распознавать, какой из методов следует использовать. Следующие методы перегружены некорректно:

```
public int метод() { return 10; }
public double метод() { return 10.5; }
```

Различия типа возвращаемого значения **недостаточно.**

6.2.6 Дополнительная информация о методах

Вся предыдущая информация о методах освещала скорее техническую сторону (структура, возвращаемое значение и параметры). Следующие примечания дополнят эту информацию и общее впечатление о возможностях методов.

Проверка присваивания атрибутов

Важная функция методов – это присваивание и возвращение значений атрибутов. Если атрибуты получают значение только через методы, то можно установить, что не образуются бессмысленные значения атрибутов. Это может быть очень важно, так как бессмысленные значения атрибутов могут дать сбой всей программы.

Пример:

Атрибут `ps` объекта гоночного автомобиля проверяется на целесообразность. Только если передаваемый параметр ограничен, устанавливается новый атрибут.

```
public void задайтеps(int psПараметр) {

      if (psПарам < 1 || psПараметр > 3500) ps = 1;
      else ps = psПараметр;
}
```

Выгрузка повторяющихся функций в методы

Наряду с функциями так называемых методов Get и Set, как на примерах выше, методы, конечно, имеют и множество других функций. Так, например, имеет смысл выгружать постоянно повторяющиеся части программы в метод и тем самым обеспечивать к ним постоянный доступ. Если эти части программы должны быть не в открытом доступе, а только внутри класса, то можно предложить частный метод.

Пример:

Во многих методах класса значение должно вводиться с клавиатуры до тех пор, пока ввод не будет ограничен и тем самым корректен. Теперь эта программа ввода может выгружаться в частный метод и использоваться другими методами.

```
private int ввод()       throws IOException {

int ввод;
BufferedReader ввод = new BufferedReader
                      (new InputStreamReader(System.in));

do {

   System.out.println("Пожалуйста, введите значение (>=0)");
   ввод = Integer.parseInt(ввод.readLine());

   }
   while (ввод < 0

   return ввод;
}
public void другойМетод() throws IOException {
int x = ввод ();
int y = ввод ();
int z = ввод ();
}
```

Постоянно повторяющаяся функция выгружается в частный метод.

Метод ввод используется другими методами ввода.

Примечание:

Любой метод класса может быть вызван любым методом (не статическим) того же класса. Внутри класса также не имеет значения, является ли метод `private`, `protected` или `public`. За пределами класса это, конечно, имеет значение.

6.3 Другие элементы классов

6.3.1 Конструкторы и деструктор

Конструкторы

Конструкторы являются особыми методами класса, которые нельзя вызвать с внешней стороны, а необходимо получить при инстанцировании объектов. Конструкторы важны при «конструировании» объекта. Как правило, они перенимают функции по инициализации. Это могут быть присваивания к атрибутам или установка соединения с базами данных, а также открытие файлов (подробнее об этом позже). Конструкторы имеют следующие свойства:

- Конструкторы называются именем класса.
- Конструкторы не могут быть вызваны в явном виде.
- Конструкторы не имеют типа данных возвращаемого значения и тем самым возвращаемого значения
- Их перегрузка может производиться с любой частотой.
- Конструктор без параметров называется **стандартным конструктором**.
- Конструктор с параметрами называется **конструктором с параметрами**.

В следующей программе показаны разные конструкторы и их вызов при инстанцировании объектов.

```
class контакт {

      private String имя;
      private String телефон;

      public контакт () {
         имя = "ПУСТОЙ";
         телефон = "ПУСТОЙ";
      }
```

Класс `контакт` должен представлять простой контакт с именем и телефоном.

Стандартный конструктор инициирует атрибуты строкой символов «ПУСТОЙ».

```
    public контакт (String имя = имяПараметр; телефон = "ПУСТОЙ";
    }

    public контакт (String имяПараметр, String телефонпарам) {
       имя = имяПараметр;
       телефон = телефонПараметр;
    }

    public void вывод()
    {
       System.out.println("Имя: " + имя);
       System.out.println("Телефон: " + телефон);
       System.out.println();
    }
}

    public class другиеЭлементы {
      public static void main(String[] args) {

          контакт первыйКонтакт = new контакт();
          первыйКонтакт.вывод();


          контакт второйКонтакт = new контакт("Майер");
          второйКонтакт.вывод();


          контакт третийКонтакт = new контакт("Майер","123456");
          третийКонтакт.вывод
       }

    }
```

Первый конструктор с параметрами перенимает параметр для имени контакта.

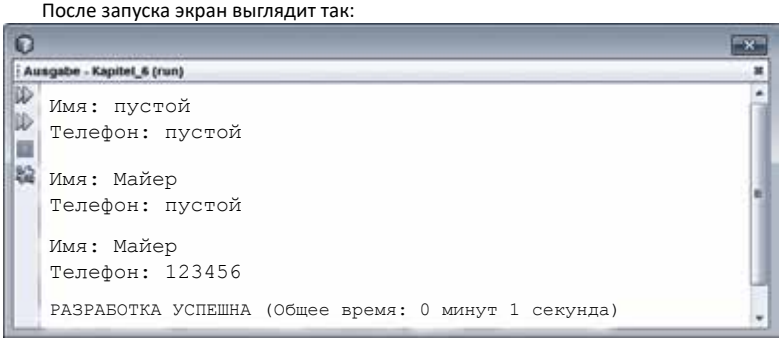
Второй конструктор с параметрами перенимает два параметра для имени и номера телефона контакта.

Неявный вызов стандартного конструктора путем ввода пустых скобок

Явный вызов первого конструктора с параметрами путем ввода параметра

Неявный вызов второго конструктора с параметрами путем ввода двух параметров

После запуска экран выглядит так:



Примечание:

Конструкторы, как правило, имеют модификатор `public`, чтобы их можно было вызвать при инстанцировании. Если же пишется частный конструктор, то инстанцирование предотвращается, как показано на примере:

```
class контакт {


      private String имя;
      private String телефон;

      private контакт() {
              имя = "ПУСТОЙ";
              телефон = "ПУСТС
      }
  :
  :
}

public class другиеЭлементы {
       public static void main(String[] args) {

            контакт первыйКонтакт = new контакт();
       }
}
```

Теперь стандартный конструктор – частный.

Инстанцирование невозможно!

Компилятор реагирует следующим сообщением об ошибке:

```
Ошибка: no suitable constructor found for Kontakt()
```

Приватные конструкторы находят свое применение, например, при использовании ***Singleton*** (шаблона проектирования). Задача приватного конструктора в том, чтобы объект класса нельзя было инстанцировать. Тогда единственной возможностью инстанцировать объект будет вызов специального метода, дополнительно ответственного за то, чтобы всегда был только один объект класса.

Деструктор

Деструктор это метод, который вызывается тогда, когда объект становится недействительным – то есть именно тогда, когда работает процесс ***garbage collector*** и удаляет объект. Деструктор, как и конструкторы, не имеет типа возвращаемого значения. Также деструктор не может перенимать параметры. Деструктор имеет стандартное имя `finalize()`. Деструктор можно вызвать в явном виде (в отличие от других языков программирования). В языке программирования типа C++ деструктор был очень важен, к примеру, для стирания зарезервированной памяти. В Java эти функции перенимает процесс ***garbage collector***, однако, несмотря на это, деструктор важен для проведения таких работ по расчистке, как отсоединение базы данных прежде, чем объект будет удален.

Пример деструктора в контакт-классе:

```
class контакт {

        private String имя;
        private String телефон;

        protected void finalize()
            System.out.println("Привет, это деструктор для" +имя)

    :
    :
}
   public class другиеЭлементы {
          public static void main(String[] args) {
```

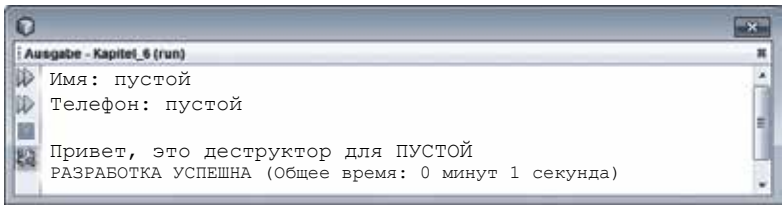
Деструктор – метод `finalize`

Метод `protected` – подробнее об этом в теме о наследовании

```
        контакт одинКонтакт = new контакт();
        одинКонтакт.вывод();
        одинКонтакт.finalize();
    }
}
```

Явный вызов деструктора

После вызова экран выглядит так:



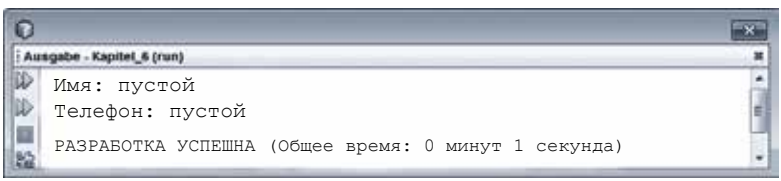
Примечание:

Когда объект становится недействительным, он автоматически удаляется процессом ***garbage collector***. Далее процесс ***garbage collector*** вызывает деструктор объекта. Однако когда процесс завершится, возможно, деструктор уже не будет вызван, как показано на примере:

```
public class другиеЭлементы {
        public static void main(String[] args) {

            контакт одинКонтакт = new контакт();
            одинКонтакт.вывод();
            одинКонтакт = null;

    }
}
```

Ссылка на контакт сбрасывается в `null` – так объект больше не имеет ссылки и процесс ***garbage collector*** должен удалить объект из памяти.

После запуска экран выглядит так:

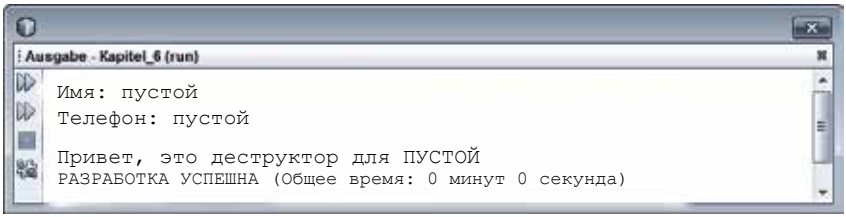


На примере показано, что деструктор не был вызван. Однако есть возможность заранее вызвать процесс ***garbage collector*** в явном виде – с помощью статического метода `System.gc()`:

```
public class другиеЭлементы {
    public static void main(String[] args) {

            контакт одинКонтакт = new контакт();
            одинКонтакт.вывод();

            одинКонтакт =
            System.gc();
    }
}
```

Явный вызов процесса ***garbage collector!***

После запуска экран выглядит так:



Внимание:

Нет гарантии, что процесс ***garbage collector*** будет работать сразу после вызова. Может быть и так, что вызывать нужно будет несколько раз. Поэтому, как правило, не имеет смысла полагаться на процесс ***garbage collector*** и деструктор, а внедрять собственный метод, который возьмет на себя работы по расчистке и т. п. Один такой метод называется `dispose()`. Метод `dispose` рассмотрим подробнее позже.

6.3.2 Ссылка this

Внутри метода есть ссылка, которая ссылается на экземпляр объекта, с которого был вызван метод. Таким образом может быть установлено целенаправленное соединение с атрибутами и методами экземпляра объекта и одинаковые имена не являются проблемой.

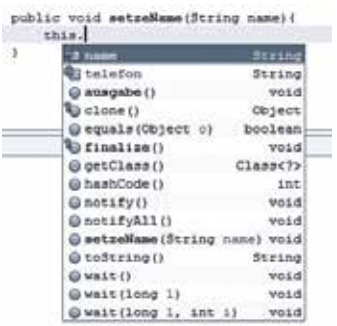
Пример: Выделение одинаковых имен параметров и атрибутов

```
class контакт {

   private String имя;
   private String телефон;

   public void задайтеИмя(String имя) {
          this.имя = имя;
   }
}
```

Опережение ссылки `this` снимает многозначность `name` – `this.name` является атрибутом, а `name` – параметром.

Примечание:

Пока параметры или локальные методы-переменные не перекрывают имя атрибута, ссылку `this` можно не использовать. Однако очень удобно использовать ссылку `this`, так как после ввода «`this.`» помощник предлагает все доступные элементы класса.



Помощник выдает список всех атрибутов и методов.

6.3.3 Статические элементы класса

При инстанцировании каждый объект получает отдельную память для своих атрибутов. Возможно параллельное существование неограниченного количества объектов. Однако иногда есть смысл в том, чтобы был общий атрибут для всех объектов. Атрибут распределяется между объектами. Такой атрибут называется **статическим атрибутом класса** или сокращенно атрибутом класса. Такой **атрибут позволяет**, к примеру, считать количество экземпляров объекта (объекты) класса. Метод также может быть статическим. Может быть вызван и без конкретного объекта, только при введении класса. Некоторые из этих практических методов уже были введены в начале: метод `main` или такие методы конвертации, как `parseInt()`. На следующем примере показан счетчик экземпляров объекта с помощью статических элементов:

Пример:

```
class контакт {

   private String имя;
   private String телефон;
   private static int счетчик экземпляров объекта = 0;

   public контакт () {
   имя = "ПУСТОЙ";
   телефон = "ПУСТОЙ";
   счетчик экземпляров объекта ++;
}

   public контакт(String имяПараметр
   имя = имяПараметр; телефон = "ПУСТОЙ";
   счетчик экземпляров объекта ++;
}

   public контакт (String имяПараметр, String телефонПараметр){
   имя = имяПараметр;
   телефон = телефонПараметр
   счетчик экземпляров объекта ++;
}


    public static int задайтеКоличествоЭкземпляров() {
           return счетчикЭкземпляров объекта;
}

public class другиеЭлементы {
   public static void main(String[] args) {

      контакт первый = new контакт();
      контакт второй = new контакт ("Майер");
      контакт третий = new контакт ("Майер","123456");

   int количество = контакт.задайтеКоличествоЭкземпляров();
   System.out.println("Количество объектов: " + количество);
```

Создать атрибут класса с ключевым словом `static`

Считать новый объект

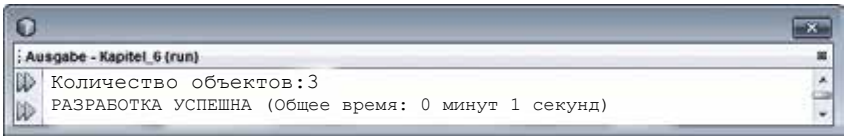
Считать новый объект

Считать новый объект

Создать статический метод с ключевым словом `static`

Вызвать статический метод

После запуска экран выглядит так:



Примечания:

► Согласно инкапсуляции целесообразно определять статический атрибут как `private`. Считывание статического атрибута происходит с помощью статического метода.

► Программисты языков C и C++ установили, что в Java нет глобальных методов и функций. Статические методы могут частично заменять такие глобальные элементы.

► Классы также могут быть статическими, однако для этого они должны создаваться как внутренние классы (подробнее об этом в теме GUI-программирование).

6.3.4 Константные элементы класса

В главе о простых типах данных уже было введено ключевое слово `final`, которое объявляет переменную постоянной. Атрибуты класса также могут быть объявлены постоянными. В некоторых случаях даже имеет смысл объявлять атрибут не только постоянным, но и статическим. На следующем примере показано применение таких атрибутов.

Пример:

```
class постоянные {
   private final int MAXWERT = 100;

   public static final double PI = 3.14;

}
```

Постоянный частный атрибут `MAXWERT`, задающий, к примеру, верхнюю границу для внутренних расчетов

Статический, постоянный и открытый атрибут `PI`, использующийся в математических расчетах, доступен всегда (даже без объекта класса `постоянные`)

Примечание:

Не только атрибуты, но и классы и методы могут быть объявлены постоянными (подробнее об этом в теме о наследовании).

6.4 Перечисляемые типы

6.4.1 Простые перечисления

Под простыми перечислениями понимают объединение нескольких постоянных значений в один блок. Только тогда переменные таких перечислений способны сохранять эти заранее заданные постоянные значения. Это имеет значение, когда речь идет об ограниченном количестве постоянных значений. На следующем примере показано простое перечисление для черного, желтого и красного цветов:

Пример

```
[public] enum цвета { ЧЕРНЫЙ, ЖЕЛТЫЙ, КРАСНЫЙ }
```

Тип перечисления также может быть `public` (как при определении класса).

Ключевое слово `enum` инициирует перечисление.

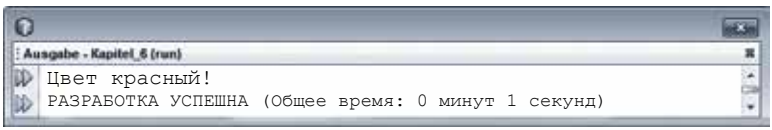
Переменные для цветов приводятся через запятую.

```
public class другиеЭлементы {
        public static void main(String[] args) {
                цвета одинЦвет = цвета.черный;

                одинЦвет = цвета.КРАСНЫЙ;

                if (одинЦвет == цвета.КРАСНЫЙ)
                     System.out.println("Цвет красный!");

                else
                     System.out.println("Другой цвет!");
        }
}
```

Создать переменную перечисления

Присвоить переменной новое значение перечисления

После запуска экран выглядит так:



Оператором `if` переменная `одинЦвет` была проверена на соответствие значению КРАСНЫЙ. Соответствие подтверждено благодаря предыдущему присвоению, и на экран выдается соответствующее сообщение.

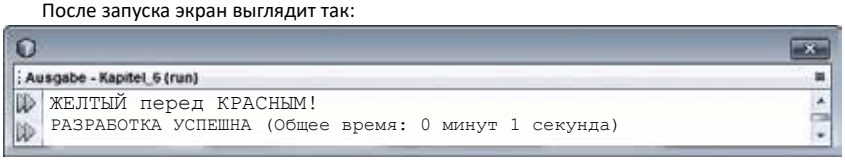
Примечание:

Переменные перечисления можно сравнивать между собой. При этом учитывается последовательность объявления постоянных, как показано на примере:

```
Цвета первыйЦвет = цвета.КРАСНЫЙ
Цвета второйЦвет = цвета.ЖЕЛТЫЙ

if (первыйЦвет.compareTo(второйЦвет) < 0)
          System.out.println("КРАСНЫЙ перед ЖЕЛТЫМ!");
else
          System.out.println("ЖЕЛТЫЙ перед КРАСНЫМ!");
```

Метод `compareTo()` сравнивает значения цветов и возвращает либо –1, 0, либо 1, если позиция `первогоЦвета` меньше, равна или больше позиции `второгоЦвета`.

После запуска экран выглядит так:



6.4.2 Классы перечислений

Тип `enum` можно использовать не только как простое перечисление, но и вводить как вид класса. При этом «`enum`-класс» обязательно имеет частный конструктор, и к постоянным значениям могут быть добавлены значения с дополнительной информацией. Далее эту информацию можно считывать с помощью специальных методов, как показано в примере.

Пример:

```
enum Столицы {

    Берлин ("Столица Германии!"),
    Лондон ("Столица Англии!"),
    Париж ("Столица Франции!");


  private String описание;

  private столицы(String описание)  {
          this.описание = описание;
   }
   public String задайтеОписание(){
          return описание;
     }
}

   public class другиеЭлементы {
       public static void main(String[] args){

            Столицы город;


            город = столицы.ЛОНДОН;

            System.out.print (город.задайтеОписание());

}
```

Постоянные значения получают дополнительную информацию (в этом примере – цепь символов).

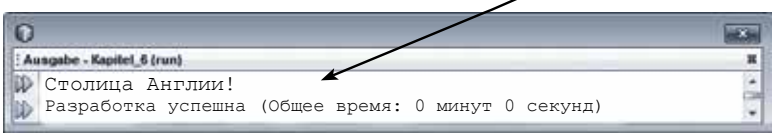
Постоянные значения также отделяются запятой, после последнего значения ставится точка с запятой.

Частный конструктор сохраняет дополнительную информацию к постоянному значению в частном атрибуте.

С помощью открытого метода `задайтеОписание()` можно вызвать дополнительную информацию к значению.

Частный конструктор отвечает за присваивание дополнительной информации для значения `ЛОНДОН`.

После запуска экран выглядит так:

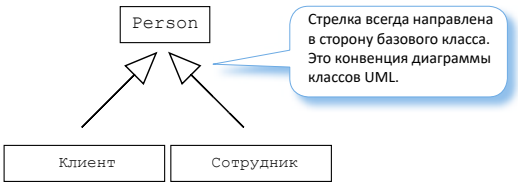


7 Наследование в Java

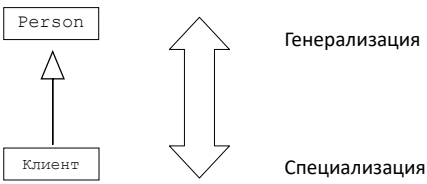
Концепт наследования является центральным в ООП. Во-первых, с помощью наследования ситуации из «реального» мира легче использовать в языке программирования, во-вторых, можно повторно использовать уже существующий код программы (в форме классов). Так достигается большая эффективность и надежность в разработке программного обеспечения благодаря уже существующим и проверенным программным кодам. При наследовании речь идет о так называемой **фактической реляции.**

Пример:

Классы `Клиент` и `Сотрудники` наследуют описание базового класса `Person`. Клиент и Сотрудники являются одним лицом (фактическая реляция). Класс Клиент и класс Сотрудники теперь имеют все элементы базового класса (с определенными ограничениями, об этом позже). Когда, например, у класса `Person` есть атрибут имя, то его наследует и класс клиент, и класс сотрудник.



Классы `Клиент` и `Сотрудник` – особые классы `Person`. Класс `Person` – это обобщение классов Клиент и Сотрудник. Поэтому также говорят о генерализации и специализации.



Примечание:

Класс, от которого идет наследование (`Person`), как правило, называется базовым, надклассом или суперклассом. Класс, который наследует (`Клиент`), называется производным классом, подклассом или субклассом.

7.1 Наследование в Java

7.1.1 Простое наследование

Когда наследование всегда идет только от одного класса, речь идет о ***простом наследовании***. Но простое наследование не означает, что несколько классов не могут наследоваться друг за другом. Следующие примеры демонстрируют простое наследование.

Пример: Простые наследования

Примечание:

Множественное наследование[1] (например, в языке C++) в Java невозможно. Тем не менее, есть возможность симулировать множественное наследование, в котором имплементируются так называемые интерфейсы (подробнее об этом позже в этой главе).

7.1.2 Применение наследования в Java

Наследование применяется относительно просто путем ввода основного класса после ключевого слова extends.

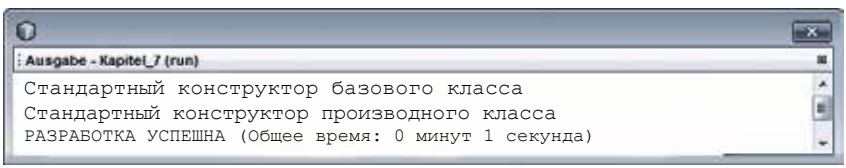
Синтаксис в Java:

```
class основной {
  public основной () {
    System.out.println("Стандартный конструктор базового класса");

    }
}
```

После ключевого слова `extends` вводится основной класс.

```
class наследование extends

     public наследование(){

System.out.println("Стандартный конструктор производного класса");
}
}
```

После запуска экран выглядит так:



На экране видно, что при инстанцировании объекта класса Наследование вызывается не только стандартный конструктор класса Наследование, но и стандартный конструктор базового класса – в первую очередь конструктор базового класса.

[1] Под множественным наследованием следует понимать параллельное наследование от неограниченного количества других классов.

Явный вызов конструкторов базового класса

Из примера выше ясно, что стандартный конструктор базового класса вызывается в неявном виде. В некоторых случаях – прежде всего, когда существует несколько конструкторов, целесообразно иметь возможность явного вызова конструктора базового класса. Это происходит с помощью ключевого слова `super`, как показано на примере:

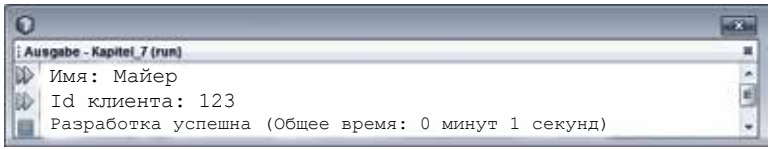
Пример:

```
class лицо {

      private String имя;
      public лицо() {
            имя = "ПУСТОЙ";
    }

      public лицо(String имяПараметр) {
            имя = имяПараметр;
    }

      public String задайтеИмя() {
            return имя;
    }
 }

class клиент extends Person {

   private int IDклиента;


public клиент()
       super();
       IDклиента = 0
}

public клиент(String имяПараметр, int IDклиентаПараметр) {
Вызов параметра конструктора

      super(имяПараметр);
      IDклиента = клиент
}

public void вывод() {
      System.out.println("имя: " + задайтеИмя());
      System.out.println("ID клиента: " + IDклиента);
   }
}

public class наследование {
    public static void main(String[] args) {
       клиент одинКлиент = new клиент();
       клиент второйКлиент = new клиент("Майер", 123);
       второйКлиент.вывод();
   }
}
```

Явный вызов конструктора базового класса с помощью `super()` – но стандартный конструктор базового класса всегда вызывается в неявном виде.

Вызов параметра конструктора с помощью `super(имяПараметр)` и передача имени конструктору базового класса

После запуска экран выглядит так:



В основном методе инстанцируются два клиента. Первый клиент инстанцируется с помощью стандартного конструктора. Так происходит также неявный вызов конструктора базового класса, и атрибуты получают значение «ПУСТОЙ» или «0». Второй клиент вызывается конструктором параметра, и переданное имя просто передается далее конструктору базового класса и параметра. Так атрибуты второго клиента получают значения «Майер» и «123».

7.1.3 Доступ к атрибутам

В производном классе `Клиент` все открытые элементы базового класса могут быть использованы так, как будто они были созданы в классе. Однако прямой доступ к частным элементам закрыт и он возможен только через методы, показанные на примере:

Пример:

Метод `вывод()` в вышеприведенном примере должен выводить как ID клиента, так и имя объекта клиента. При этом следующий вариант невозможен:

```
public void вывод() {
        System.out.println("имя: " + имя);
        System.out.println("IDклиента: " + IDклиента);
}
```

Прямой доступ к наследуемому атрибуту имя **НЕ** возможен!

Исполнение метода Get `задайтеИмя()` в классе `Person` решает эту проблему, однако существует еще другая (более удобная) возможность – с помощью модификатора `protected`. Атрибут имя в классе лицо просто следует обозначить как `protected`.

```
class Person {

    protected String имя;
    :
    :
}
public void вывод() {
      System.out.println("имя: " + имя);
      System.out.println("ID клиента: " + ID клиента);
}
```

Обозначить атрибут как protected

Прямой доступ к наследуемому атрибуту имя **ТЕПЕРЬ** возможен!

Примечание:

Ко всем атрибутам, объявляемым как `protected`, может быть прямой доступ в производных классах, однако внешне они защищены как частные элементы. Как вариант, атрибут может быть создан и без модификатора. Доступ в производном классе также был бы возможен, но и во всех остальных классах этого пакета, как показано на примере:

```
class A {

        String безмодификатора;
}
```

Атрибут создается без public, private protected.

```
    class B extends A {
          public B() {
            безМодификатора = "ПУСТОЙ";
    }
}

    class C {
            A однаСсылка = new A();
            public C() {
                        однаСсылка.безМодификатора= "ПУСТОЙ";

    }
}
```

Класс **B** производный от **A** и поэтому может иметь прямой доступ к атрибуту.

Класс **C** инстанцирует объект класса **A**.В отличие от наследования это отношение называется ***Has***-отношением (или в профессиональном языке UML ассоциацией).

Прямой доступ через объект возможен.

7.1.4 Финальные классы

Если необходимо предотвратить наследование класса, то базовый класс следует определить как `final`. В таком случае наследование другими классами этого класса будет невозможно.

Пример:

```
final class нетНаследования { … }
class нужноНаследование extends нетНаследования { … }
```

Ошибка компилятора:
`cannot inherit from нетНаследования`

7.2 Полиморфизм

Только язык программирования, использующий полиморфизм, может называться объектно-ориентированным. Иначе это просто язык программирования, ***основывающийся на объектах.*** Общее значение полиморфизма в том, что ссылка во время исполнения программы может указывать на разные объекты, и, тем не менее, корректные методы объекта можно вызвать с помощью данной ссылки. Это объяснение, которое кажется немного сложным, теперь следует пояснить на конкретных примерах. Для этого на первом этапе класса `java.lang.Object` (основа всех классов) обсуждаются присваивания ссылок ссылкам в рамках иерархии наследования. В дальнейшем суть полиморфизма можно объяснить более подробно.

7.2.1 Класс Объект

В Java есть важный класс – `Объект`. Этот класс можно рассматривать как базовый класс всех классов. **Каждый класс (даже собственные классы) по умолчанию производится от класса, это не требуется задавать.** Поэтому каждый тип ссылки (или ссылку) можно сконвертировать в объект класса `Объект`.

Пример:

```
Объект objCсылка;
клиент одинКлиент = new

objCсылка = одинКлиент;
```

Создать ссылку типа `object`

Объект (или ссылка) класса `Клиент` присваивается объекту или ссылке.

Упаковка

Переменные и типы значений (а также литералы) также могут быть преобразованы в тип ссылки. При этом происходит так называемая ***автоупаковка***. Тип значений, сохраненный в памяти стека, становится объектом, который сохраняется в памяти HEAP. Так в динамичной памяти резервируется место и там сохраняется содержание типа значений. На следующем примере показана ***автоупаковка***.

Пример:

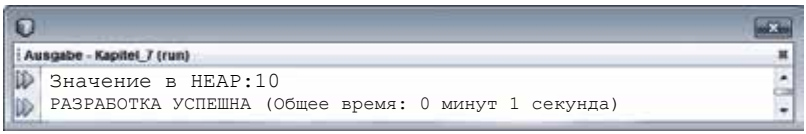
```
Odject однаСсылка
int x = 10;
однаСсылка = x;
x = 20;

System.out.println("Значение в HEAP: " + objСсылка);
```

Автоупаковка: В памяти HEAP сохраняется объект целого числа

Переменная x получает новое значение.

Сохраняет значение 10

После запуска экран выглядит так:



Примечание:

На первый взгляд кажется удивительным, что ссылку `objСсылка` можно просто вывести на экран и с корректным значением (то есть 10). Это связано с тем, что класс `объект` исполнил метод `toString()`, который также должен быть исполнен производными классами. Этот метод возвращает содержание объекта как строку символов. Если при выводе на экран будет выведена только ссылка, то этот метод `toString()` будет вызван по умолчанию. Значение целого числа было по умолчанию упаковано с помощью `автоупаковки` в объект класса `целое`, и этот класс, конечно, предлагает метод `toString()`. Класс `целое` – это так называемый ***класс-оболочка*** (подробнее об этом позже).

Классы-оболочки

К каждому простому типу данных Java предлагает ***класс-оболочку***. Объекты этих классов должны предлагать объектно-ориентированные противоположности к простым типам данных. Так, значения простых типов данных могут быть сохранены и в памяти HEAP и управляться соответствующей ссылкой. В принципе, в вышеописанной *автоупаковке* используются такие классы-оболочки для упаковки значений простых типов данных и сохранения в памяти HEAP.

Обзор классов-оболочек:

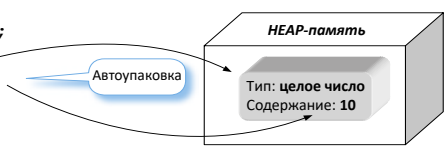
Тип данных	Класс-оболочка
byte	Byte
char	Character
short	Short
int	Integer

Тип данных	Класс-оболочка
long	Long
float	Float
double	Double
boolean	Boolean

Пример ***автоупаковки*** теперь следует рассмотреть подробнее:

Пример:

```
object однаСсылка;
int x = 10;

однаСсылка = x;
```



Также возможно использование ссылки типа целого числа:

Пример:

```
Integer целСсылка;
int x = 10;

intCсылка = x;
```

Автоупаковка

Возможно также явное создание объектов:

Пример:

```
int x = 10;
Integer intCсылка = new Integer(x);
```

Явная упаковка

Примечание:

Классы-оболочки уже использовались при конвертировании цепи символов в другие типы данных – например, с помощью статического метода `Integer.parseInt()`.

Распаковка

Распаковка имеет противоположное значение ***упаковки***. Тут тип ссылки (или объект) конвертируется в тип значения. Для этого можно использовать метод ***класса-оболочки*** или ***автораспаковки.***

Пример:

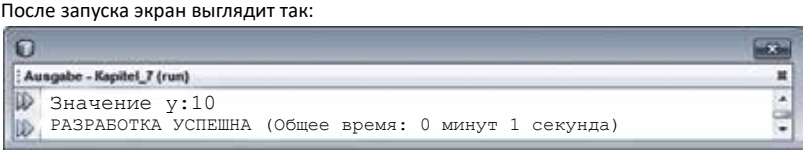
```
int x = 10;
целое intCсылка = new Integer (x);

int y = intCсылка.intValue();
// Альтернативно: int y = intCсылка;

целСсылка = 20;

System.out.println("Значение y: " + y);
```

Распаковка методом `intValue()`

Автораспаковка

Новая упаковка

После запуска экран выглядит так:



После ***распаковки*** переменная y имеет значение 10. Даже если объект целого числа получает новое значение, переменная y по-прежнему имеет значение 10. ***Распаковка***, таким образом, копирует значение из объекта и присваивает его переменной y – ссылка (или объект) и переменная не зависят друг от друга.

7.2.2 Присваивание в рамках иерархии наследования

В рамках иерархии наследования объекты (или ссылки) могут присваиваться производным классам объектов базового класса (или общим классам). Это имеет смысл, так как объекты производного класса имеют всю информацию (значения), которую должен иметь базовый объект класса. В обратном порядке это не будет иметь смысла.

Пример:

Исходной базой является иерархия наследования лицо, клиент и сотрудник.

```
class Person {
        :
        :
}
```

```
class клиент extends Лицо {
      private int IDклиента;
     :
}

class сотрудник extends лицо {
      private String отдел;
       :

public String задайтеОтдел() {
       return отдел;
     }
}

public class полиморфизм {
       public static void main(String[] args) {

         лицо одноЛицо = new лицо();
         клиент одинКлиент = new клиент();
         сотрудник одинСотрудник = new сотрудник();

         одноЛицо = одинКлиент;
         одноЛицо = одинСотрудник;

         одинКлиент = одинСотрудник;
         одинСотрудник = одинКлиент;
         одинКлиент = одноЛицо;
         одинСотрудник = одноЛицо;
     }
}
```

Клиент производное от Лицо.

Сотрудник также производное от Лицо.

Присваивание в порядке.

Присваивание не в порядке – отсутствие данных

Ошибка компилятора: incompatible types

Примечание:

Присваивание Клиента или Сотрудника Лицу происходит нормально. Однако специальные атрибуты и методы больше не могут быть вызваны Клиентом или Сотрудником, так как ссылка будет только для Лица. Например, следующий доступ будет **невозможен**:

Нет доступа!

```
System.out.println("отдел: " + одноЛицо.задайтеОтдел
```

Метод `задайтеОтдел()` определен в классе Сотрудника и не может быть доступен по ссылке Лица. Данная проблематика также будет рассмотрена в следующем подразделе.

7.2.3 Перезапись методов

Перезаписанные[2] методы создаются в базовом классе и, как правило, заново исполняются в производном классе (перезаписываются). Теперь, если объект производного класса присваивается объекту базового класса, во время исполнения с помощью перезаписанного метода может быть вызван правильный метод. Следующий пример поясняет это сложное объяснение.

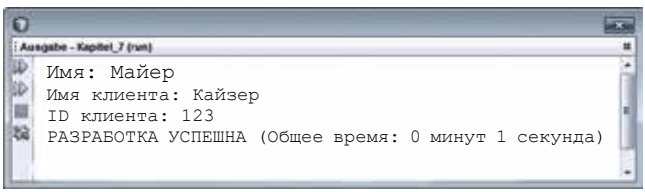
[2] Эти методы также называют *виртуальными*. В других языках программирования виртуальные методы должны быть особо обозначены – в Java это не требуется.

Пример:

Как в классе Лицо, так и в производном классе Клиент есть метод `вывод()`, который выводит все данные объекта на экран.

```
class лицо {
    :
    public void вывод() {
           System.out.println("Имя: " + имя);
           System.out.println();
    }
}
class клиент extends Лицо {
    :
    public void вывод() {
           System.out.println("Имя клиента: " + имя);
           System.out.println("ID клиента: " + IDклиента);
    }
}
public class полиморфизм {
     public static void main(String[] args) {

              лицо одноЛицо = new лицо("Майер");
              клиент одинКлиент = new клиент("Кайзер",123);
              одноЛицо.вывод();
              одинКлиент.вывод();
    }
}
```

После запуска экран выглядит так:



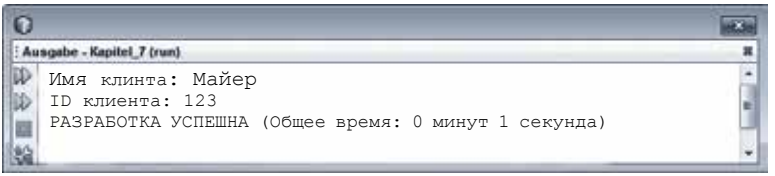
Методы вывода функционируют нормально – данные Лица и Клиента выводятся на экран. На следующем примере класс Клиент будет присваиваться по ссылке Лицо.

Объяснение примера:

Клиент присваивается по ссылке Лицо и метод `вывод()` используется повторно.

```
public class полиморфизм {
    public static void main(String[] args) {

           Person одноPerson = new Person("Майер");
           клиент одинКлиент = new клиент("Кайзер",123);

           одноPerson = одинКлиент;
           одноPerson.вывод();
    }
}
```

После запуска экран выглядит так:



Можно видеть, что правильный метод `вывод()` вызывается из класса клиента, хотя это ссылка Лица. С помощью перезаписи метода в производном классе в языке Java был применен полиморфизм.

После введения перезаписи теперь можно подробнее рассмотреть и функционирование метода `toString()`. Данный метод описан в базовом классе `Object`. Так как каждый класс неявно наследует класс `Object`, метод `toString()` может и должен перезаписываться в каждом классе для применения полиморфизма. Если предпринять вывод данных в классе Лица с помощью метода `toString()`, вывод Лица будет еще проще:

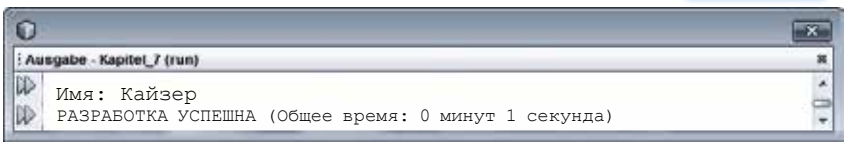
```
class лицо {
      :

      public String toString() {
            return "Имя: " + имя;
      }
}

System.out.println(одноЛицо.toString());
// Альтернативно: System.out.println(одноЛицо);
```

Метод `toString()` возвращает строку символов.

Вызвать метод `toString()`!

Неявный вызов `toString()`!

После запуска экран выглядит так:



Примечания:

• Со времен версии Java 5 существуют так называемые **аннотации**, которые может вносить программист рядом с Java-операциями и комментариями в исходном тексте. Эти аннотации должны передавать компилятору определенные задачи. Например, если необходимо перезаписать метод, то это можно отметить с помощью аннотации «`@Override`». Далее компилятор проверяет синтаксическую корректность перезаписи. Среда разработки `NetBeans`, к примеру, автоматически предлагает аннотацию, когда есть перезапись.

Пример:

```
class лицо {
      :
      public void вывод() {
            System.out.println("имя лица: " + имя);
            System.out.println();
      }
}
```

```
class клиент extends лицо
        :
        @Override
        public void вывод() {
                System.out.println("имя клиента: " + имя);
                System.out.println("ID клиента: " + IDклиента);
        }
}
```

аннотация @Override

• Если следует предотвратить полиморфизм в классе, то необходимо отметить метод только ключевым словом final. Так перезапись будет запрещена:

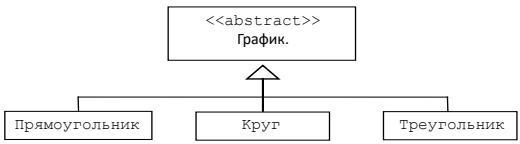
```
final void нетПолиморфизма() { … }
```

7.3 Абстрактные базовые классы

Абстрактный базовый класс должен стать основой для других классов, без возможности инстанцирования объекта от этого класса. Абстрактные классы имеют смысл тогда, когда для иерархии наследования необходима основа, но от основного класса не могут и не должны инстанцироваться смысловые объекты.

Пример:

Пример использования абстрактных базовых классов – иерархия классов для сохранения графических объектов (круги, треугольники, прямоугольники и т. д.). Должна быть возможность изображения каждого объекта на экране. Поэтому целесообразно делать набросок абстрактного базового класса – график с несколькими основными атрибутами и методами.



7.3.1 Абстрактный базовый класс

Класс становится абстрактным базовым классом с помощью ключевого слова abstract. Так ни один объект этого класса не может быть инстанцирован, как показано на примере.

Пример:

```
abstract class график {

    public void изобразить (){
      //TODO: перезаписать
    }
}
class прямоугольник extends график
        @Override
        public void изобразить(){
                System.out.println("Привет, я прямоугольник!");
        }
}
class круг extends график {
    @Override
    public void изобразить(){
            System.out.println("Привет, я круг!");
    }
}
```

Класс становится абстрактным.

Метод изображения перезаписывается и подстраивается под класс.

Инстанцирование невозможно – ошибка компилятора:
Grafik is abstract; cannot be instantiated

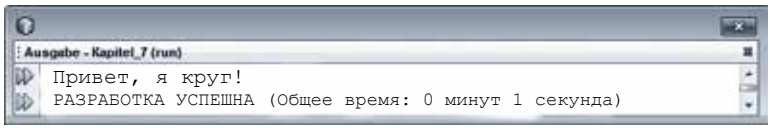
```
public class абстрактныеКлассы
    public static void main(String[] args)

                    //график графикСсылка = new график();
```

Ссылка возможна.

Присваивание объекта из иерархии наследования ссылке базового класса функционирует правильно

```
                    График графикСсылка = new изобразить();
                    графикСсылка.изобразить();
        }
}
```

После запуска экран выглядит так:



Примечание:

Методы абстрактного класса не должны исполняться, так как обычно производные классы исполняют методы. Поэтому такие методы должны быть обозначены ключевым словом `abstract`. Если класс получает хотя бы один абстрактный метод, то и класс тогда должен быть абстрактным.

Пример:

Метод не исполняется и поэтому должен быть ***абстрактным***.

```
abstract class график {

        public abstract void изобразить
}
```

7.4 Интерфейсы в Java

Принцип интерфейсов (сопряжения) имеет много общего с понятием абстрактных базовых классов. Интерфейс выглядит как класс, однако без какого-либо исполнения. Интерфейс только определяет, какие методы необходимо использовать. Класс, исполняющий интерфейс, должен также описывать методы. Класс может исполнять неограниченное количество интерфейсов.

7.4.1 Структура интерфейса

С помощью ключевого слова `interface` задается объявление интерфейса. Все методы только объявляются, не исполняются. Методы интерфейса **неявные**, **абстрактные** и **открытые**. На следующем примере приведены описания интерфейсов и исполнения в классе.

Пример

Интерфейс также может быть `public` (как класс).

Создание интерфейса.

```
interface Вывод {

        public abstract void вывод(object obj);
}
```

Создание другого интерфейса.

```
interface Ввод {
    public abstract Ввод объекта() throws IOException ;

}
```

Интерфейс объявляет неограниченное количество абстрактных методов. Модификаторы `public` и `abstract` могут быть убраны.

С помощью ключевого слова `implements` исполняется неограниченное количество интерфейсов.

```
class Программа консоли implements Вывод, Ввод {
```

Метод `вывод()` интерфейса вывод теперь должен быть исполнен.

```
    @Override
    public void вывод (object obj){
    System.out.println("содержание: " + obj.toString());
}
```

Также метод `вывод()` сопряжения Ввод должен быть исполнен.

```
@Override
public ввести объект() throws IOException {

    BufferedReader ввод  =
    new BufferedReader(new InputStreamReader(System.in));

    System.out.println("Пожалуйста, введите: ");
    return (объект)(ввод.readLine());
  }
}
public class интерфейсы {
   public static void main(String[] args) throws IOException {
```

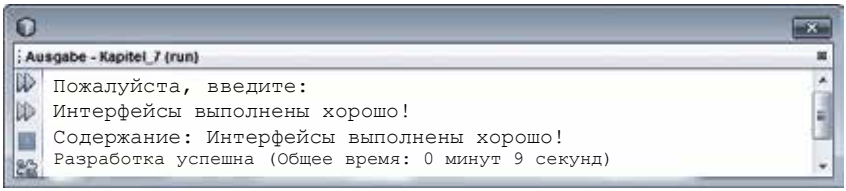
Объект класса инстанцируется программой консоли.

```
  //Альтернативно: Ввод con = new программа консоли();
  //Альтернативно: Вывод con = new программа консоли ();

  con.вывод(con.ввод());

  }
}
```

Объект использует имплементарные методы.

От интерфейсов не могут быть инстанцированы объекты, но они могут быть ссылками на объекты классов, которые имплементировали интерфейс.

После запуска экран выглядит так:



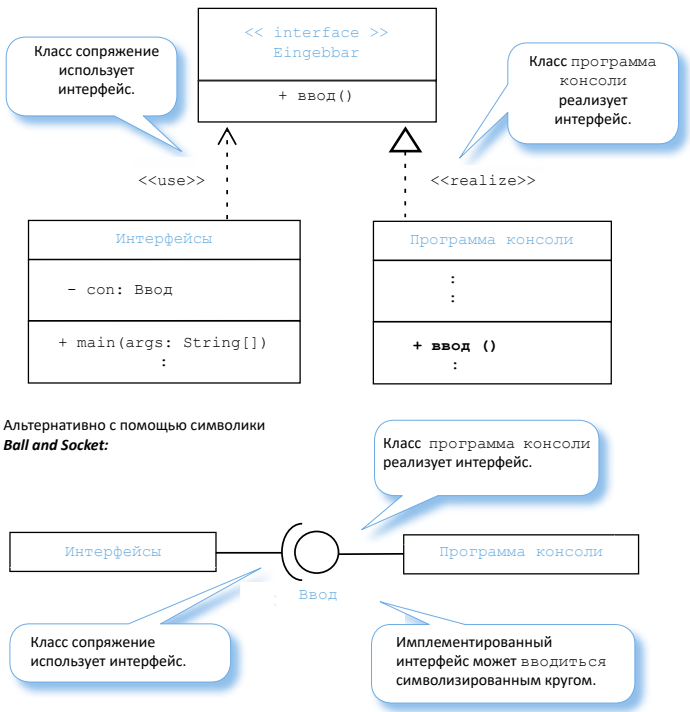
Примечание:

Имена интерфейсов обычно выбираются так, чтобы они выражали цель интерфейса. В предыдущем примере были названы интерфейсы `Ввод` и `Вывод`. Так, объекты классов, исполняющие названные интерфейсы, должны иметь эти характеристики, то есть иметь возможность ввода и вывода. Библиотеки Java предлагают множество интерфейсов, которые могут исполняться для соответствующих целей. В теме о массивах и сортировке интерфейс `Comparable` будет разработан подробнее.

Представление интерфейсов в UML

Наследование классов в UML было представлено в начале главы. Реализация интерфейсов обычно происходит с другой символикой. Следующий пример демонстрирует представление в UML предыдущего примера:

Пример:



8 Массивы в Java

Допустим, необходимо в одной программе сохранить десять значений типа int и далее работать с этими значениями. Для этого можно создать десять переменных типа int и выполнить последовательный ввод.

Пример:

```
package глава_8;
import java.io.*;

public class массивы {
    public static void main(String[] args) throws IOException {
      BufferedReader ввод  =
       new BufferedReader(new InputStreamReader(System.in));

       int значение1, значение2 ,значение3 , значение4, значение5;
       int значение6, значение7, значение8, значение9 , значение10;

       System.out.println("Ввод первого значения:");
       значение1 = Integer.parseInt(ввод.readLine());
       :
       System.out.println("Ввод последнего значения:");
       значение10 = Integer.parseInt(ввод.readLine());
  }
}
```

Это не очень эффективно: Не только потому, что много письменной работы, но и сложная реализация многих решений проблем. Например, поиск минимума введенных чисел был бы связан с затратами.

Пример:

```
int minimum; minimum=значение1;
if (minimum < значение2) minimum = значение2;
if (minimum < значение3) minimum = значение3;
if (minimum < значение4) minimum = значение4;
:
if (minimum < значение8) minimum = значение8;
if (minimum < значение9) minimum = значение9;
if (minimum < значение10) minimum = значение10;
```

Можно догадаться, с какими затратами это было бы связано, если бы необходимо было проработать не 10, а 1000 или 10000 значений. Для облегчения таких процессов в Java (и во всех других языках программирования) существуют так называемые массивы, так же называемые полями.

8.1 Одномерные и многомерные массивы

8.1.1 Одномерные массивы

Если необходимо (как в вышеописанных примерах к массивам) сохранить несколько значений одного типа данных, это можно сделать с помощью (одномерного) массива.

Вместо:

```
int значение1;
int значение2;

:
:
int значение10;
```

```
int []значения = new int[10];
```

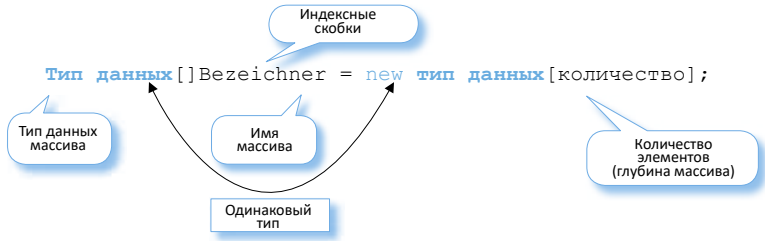
Таким образом, существует десять областей памяти для целых значений. К каждой отдельной области памяти есть доступ через имя массива (здесь значения) и так называемый индекс. Например, если нужно сохранить значение 100 в первом элементе массива, это может происходить так:

```
значения[0] = 100;
```

Индекс

Первая область памяти или первый элемент массива получает значение 100.

В целом, синтаксис массива можно представить так:



Пример:

Массив значения должен быть заполнен числами.

```
int[] значения = new int[10];
int i;
for (i = 0; i < 10; i++) значения[i] = i * 10;
```

Доступ к элементам массива осуществляется с помощью оператора **Индекс [ ]**. Если необходим доступ к специальному элементу, это происходит с помощью оператора Индекс и введения индекса.

После цикла значения массива такие:

Индекс i	0	1	2	3	4	5	6	7	8	9
Содержание: значения[i]	0	10	20	30	40	50	60	70	80	90

ВНИМАНИЕ:

В Java массив хоть и создается с десятью элементами, однако индекс проходит от 0 до 9. Это связано с внутренним сохранением элементов.

Конечно, массивы могут быть образованы с каждым типом данных.

Примеры других массивов:

```
class лицо {
   private String name;

   public лицо(){
          имя = "ПУСТОЙ";
   }


   public лицо(String n){
          имя = n;
   }
}
```

Простой класс Лицо для примера

```
char [] последовательность знаков = new char[100];
float [] измеряемые значения = new float[1200];
boolean [] истинные значения = new boolean [8];

лицо [] лица = new лицо[5];
```

Массивы простых типов данных

Массивы ссылок Лицо

Примечание:

Глубина массива может быть определена и с помощью переменной. Так можно динамично изменять необходимый размер массива, как показано на примере:

```
int количество;
int [] динамичныйМассив;

System.out.println("Сколько элементов?");
количество = целое.parseInt(ввод.readLine());
```

Глубина массива определяется во время исполнения. Однако переменная `Количество` должна быть проверена на смысловое значение.

```
if (количество > 0)
                динамичный массив new int [количество];
else
                System.out.println("Ошибочный ввод");
```

Инициализация массивов

При описании массива сразу может проходить инициализация, при которой значения для массива задаются в фигурных скобках. Так глубина массива рассчитывается автоматически из количества значений в фигурных скобках.

```
int[] числа = { 1, 2, 3, 4, 5 };

лицо[] лица =  { new лицо("A"), new лицо("B")};
```

Инициализация массива с Числами. Глубина массива равна 5.

Инициализация массива с лицами. Глубина массива равна 2.

Примечание:

При объявлении массива ссылки на первом этапе динамично создаются только ссылки. Далее объекты либо сразу инстанцируются путем инициализации, либо это необходимо делать позже в явном виде.

```
лицо[] лица = { new Слицо("A"), new Слицо("B")};
```

Массив ссылок лиц

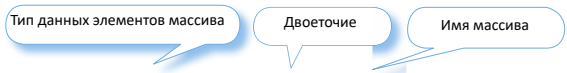
или

```
лицо[] лица = new лицо
лица[0] = new лицо("A");
лица[1] = new лицо("B");
```

8.1.2 Цикл for each

В Java был введен вид цикла, который также существует в некоторых других современных языках – цикл ***for each***. Этот цикл был введен специально для работы с массивами и имеет смысл только при использовании вместе с массивами. С помощью этого цикла массив может быть пройден очень просто элемент за элементом, без необходимости знать количество элементов массива.

Синтаксис цикла ***for each***:



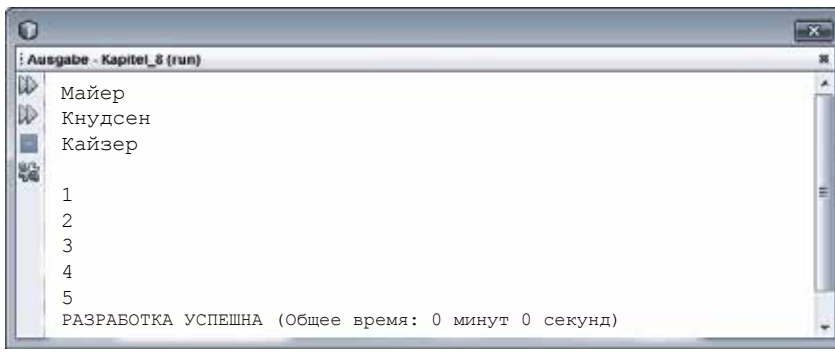

```
for (Тип данных Обозначение : Имя массива)

                    //Использование переменной обозначение
        }
```

На следующих примерах цикл ***for each*** показан на практике. Массив лиц и массив целых значений проходят элемент за элементом и выводятся на экран:

```
public class массив {
     public static void main(String[] args) {

        лицо[] лица = {   new лицо("Майер"),
                          new лицо("Кнудсен"),
                          new лицо("Кайзер") };

         int[] числа = {1, 2, 3, 4, 5 };

         for (лицо прохождение : лица) {

         System.out.println(прохождение);

         }

         System.out.println();

         for (int прохождение : числа) {

                  System.out.println(прохождение);
        }
    }
}
```

Массив проходит от начала до конца. Переменная `прохождение` каждый раз получает актуальный элемент массива.

После запуска экран выглядит так:



Массивы проходят шаг за шагом, и содержание выводится на экран.

Примечание:

Конечно, массивы можно обрабатывать и с помощью уже знакомых видов циклов. Цикл ***for each*** является лишь удобным дополнением – специальные доступы (например, использование каждого второго или третьего элемента) лучше осуществлять с помощью обычных циклов. На следующем примере противопоставляются три разных вида циклов: все циклы выводят содержание элементов массива на экран. При этом используется постоянный атрибут `length`, который сохраняет длину (глубину) массива.

```
int[] значения = { 10, 20, 30 }   Применить цикл do-while
int i = 0;

do {
   System.out.println(значения[i);
   i++;
} while (i < значения.length);

System.out.println("-----------");

for (i = 0; i < значения.length; i++) {
   System.out.println(значения[i]);
}
System.out.println("-----------");
for(int прохождение : значения) {
   System.out.println(прохождение
}

System.out.println();
```

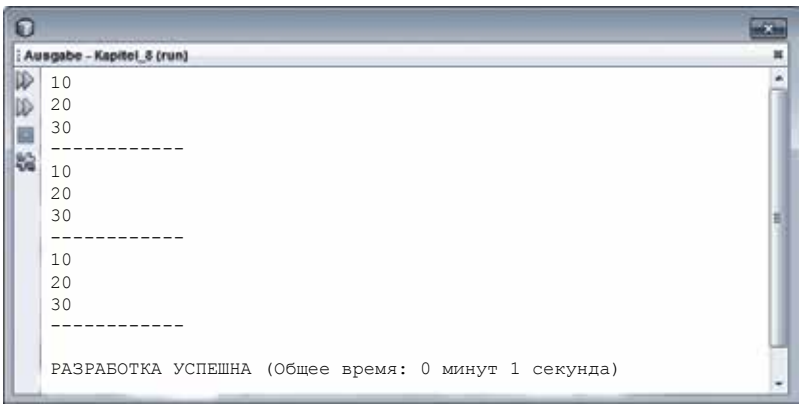
Применить цикл do-while

Использовать постоянный атрибут length

Использовать цикл for

Использовать цикл for each

После запуска экран выглядит так:



Все три цикла выводят содержание корректно.

8.1.3 Многомерные массивы

Понятие многомерных массивов сначала относительно сложное. Поэтому важно сначала получить представление об одномерных, двумерных и трехмерных массивах. Представление двумерных массивов показано таблице. Таблицы общепринятые – они состоят из строк и столбцов, которые отражают соответствующий индекс массива.

Пример:

Создается двумерный массив.

Определить первый размер – соответствует количеству строк таблицы.

Определить второй размер – соответствует количеству столбцов таблицы.

```
int [][] таблица = new int 3 [ 4 ];
```

В соответствии с размерами ставятся пустые скобки.

Двумерный массив представлен в таблице. Доступ к элементу массива осуществляется с помощью двойного индекса. Например, доступ к элементу из строки 0 и столбца 1 можно получить так:

таблица[0][1] = 3

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0		3		
Строка 1				
Строка 2				

Обычно многомерный массив может быть инициализирован напрямую. Таким образом также определяются размеры:

```
int [][] таблица = {        {5,3,7,2} ,
                            {8,6,9,1} ,
                            {2,7,3,4}
```

Три строки с тремя столбцами в каждой

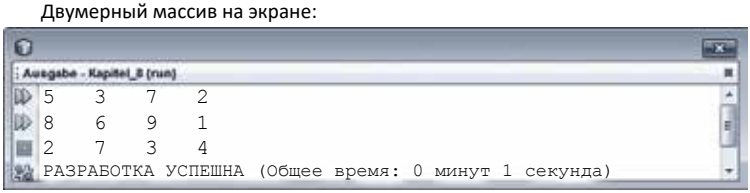
	столбец 0	столбец 1	столбец 2	столбец 3
строка 0	5	3	7	2
строка 1	8	6	9	1
строка 2	2	7	3	4

Размеры также могут быть считаны с помощью постоянного атрибута `length`, как показано на примере:

```
for (int i=0; i < таблица.length; i++) {
     for (int j=0; j < таблица[0].length; j++) {
          System.out.print(таблица[i][j] + "      ");
     }
     System.out.println();
}
```

Метод `print()` не разрывает строку после вывода.

Двумерный массив на экране:



Трехмерные массивы можно представить как сборник рабочих листов, расположенных друг за другом.

Пример:

Создается трехмерный массив.

```
float  [][][] таблицы = new float [3] [3] [4];
```

Трехмерный массив можно представить так:

		Лист 2	Столбец 0	Столбец 1	Столбец 2	Столбец 3
		Строка 0	1.5	3	12.33	25.3
	Лист 1	Столбец 0	Столбец 1	Столбец 2	Столбец 3	99.2
	Строка 0	1.5	7	12.33	45	10
Лист 0	Столбец 0	Столбец 1	Столбец 2	Столбец 3	45.55	
Строка 0	1.5	7	12.33	45	78.6	
Строка 1	124	99.99	453	67.89		
Строка 2	12	90.2	2727.5	22		

```
таблицы [1][1][3] == 45.55f
```

Рабочий лист | Строка | Столбец

После третьего расширения человеческое воображение заканчивается. Многомерные массивы с размерами больше трех невозможно конкретно представить как, например, таблицы, однако возможно целесообразное использование этих массивов.

Пример пятимерного массива:

Для психологического эксперимента создаются три разных группы по 15 участников. Каждый участник получает 10 анкет, в каждой по 12 вопросов. Каждый вопрос имеет 3 варианта ответа, из которых можно выбрать. Массив, отражающий данный эксперимент, может выглядеть так:

```
boolean [][][][][] эксперимент = new boolean [3][15][10][12][3];
```

Теперь нужно сохранить ответ 5-го участника из группы 2 из 7-й анкеты на 4-й вопрос. Три ответа были: `Да, Нет, Да`.

Для удобства ответ «Да» сохраняется с помощью логического типа `true`, ответ «Нет» с помощью логического типа `false`.

```
эксперимент [1][4][6][3][0] = true;
эксперимент [1][4][6][3][1] = false;
эксперимент [1][4][6][3][2] = true;
```

Массивы для массивов

Многомерные массивы, в принципе, представляют собой не что иное, как массивы для массивов с постоянной величиной. Однако в Java есть возможность не точно определять второй и больший размеры, а задавать переменные значения. Так получается многомерный массив, который имеет подмассивы разных размеров. Простой пример объяснит этот принцип.

Пример: Необходимо создать двумерный массив, четыре строки которого имеют разное количество столбцов (форма треугольника).

	Столбец 0	Столбец 1	Столбец 2	Столбец 3
Строка 0	1			
Строка 1	1	2		
Строка 2	1	2	3	
Строка 3	1	2	3	4

```
int [][] таблица = new int [4][];
```

Двумерный массив с **не**определенным вторым размером вывода.

```
for (int i=0; i < таблица.length; i++) {
        таблица[i] = new int[i+1];
```

Динамичное определение второго размера с помощью переменной цикла i

```
        for (int j=0; j < таблица[i].length; j++)
        таблица[i][j]= j+1;
}
```

Заполнение массива с помощью другого цикла

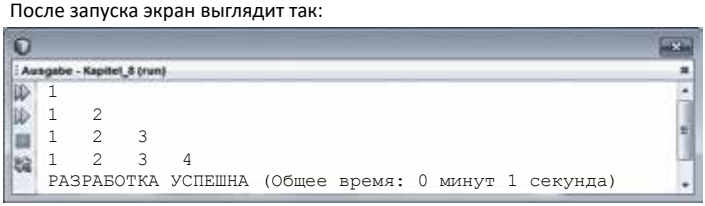
```
for (int i=0; i < таблица.length; i++) {
```

Считывание соответствующего размера

```
        for (int j=0; j < таблица[i].length; j++) {
             System.out.print(таблица[i][j] + "       ");
        }
        System.out.println();
}
```

Вывод массива треугольника

После запуска экран выглядит так:



8.1.4 Копирование массивов

Массивы в Java это типы ссылок. Поэтому присваивание массива другому массиву будет просто присваиванием ссылки. Однако если нужно сделать настоящую копию, необходимо либо копировать значения массивов в цикле шаг за шагом, либо использовать статический метод из класса `java.util.Arrays`. На следующем примере показаны оба способа копирования:

Пример:

Существуют следующие массивы:

```
int[] первыйМассив = {1,2,3};
int[] второйМассив = new int[3];
```

Вариант 1: Копировать только ссылку

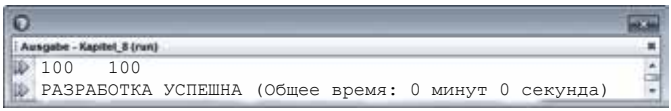
```
второймассив = первыймассив;
второймассив[0] = 100;
```

Присваивание ссылки

Изменение элемента второго массива

```
System.out.println(первыйМассив[0] + " " + второйМассив[0]);
```

После запуска экран выглядит так:



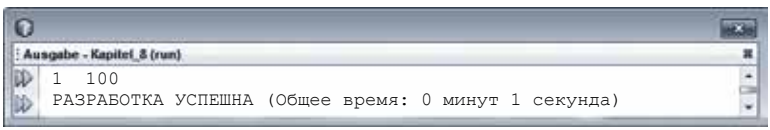
Можно видеть, что и первый, и второй массив подверглись изменениям, поскольку в памяти есть только один массив, на который ссылаются две ссылки.

Вариант 2: Последовательное копирование массива

```
int[] первыйМассив = {1,2,3};
int[] второйМассив = new int[3];

for (int i = 0; i < первыйМассив.length;i++)
     {второйМассив[i] = первыйМассив [i];
}

второйМассив [0] = 100;
System.out.println(первыйМассив [0] + " " + второйМассив [0]);
```

Последовательное копирование элементов

После запуска экран выглядит так:



Теперь можно видеть, что было произведено настоящее копирование. Первый массив не подвергся изменению.

Вариант 3: Использование статического метода java.util.Arrays

```
int[]первыйМассив = {1,2,3};
```

Статический метод класса массива

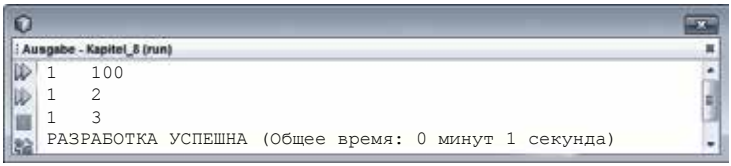
Задать массив, подлежащий копированию

```
int[] второйМассив = java.util.Arrays.copyOf(первыйМассив,3);
```

Задать количество элементов, подлежащих копированию

```
второйМассив[0] = 100;

for (int i = 0; i < первыйМассив.length; i++) {
   System.out.println(первыйМассив[i] + "" + второйМассив[i]);
```

После запуска экран выглядит так:



Теперь также можно видеть, что было произведено настоящее копирование.

Примечание:

Класс **`java.util.Arrays`** предлагает другие полезные статические методы для обработки массивов:

- Метод копирует массив от индекса `Start` (incl.) до индекса `End` (excl.):

  `copyOfRange(одинМассив,start,end)`
- Метод заполняет элементы массива значением:

  `fill(одинМассив,значение)`

8.1.5 Массивы объектов

Уже знакомый класс `object` можно рассматривать как базовый класс всех классов в Java. Все классы, которые создаются самостоятельно, автоматически производятся от этого класса. Поэтому можно создать массив по типу `object`. В таком массиве могут быть сохранены все переменные типов значений (с помощью упаковки) и экземпляры объекта классов (ссылки). Такой массив является своего рода контейнером для любых значений и ссылок. Поэтому в самостоятельно написанных классах необходимо также перезаписать метод `toString()`, чтобы с помощью элементов объекта всегда иметь доступ к корректному методу.

Пример:

```
object[] objectМассив = new object[3];

int x = 10;
String s = "Привет";
Лицо лицо = new лицо("Ханзен");

objectМассив[0] = x;

objectМассив [1] = s;

objectМассив [2] = лицо;

for (object o : objectМассив
     System.out.println(o.toString());

x = 20;
s = "Ага";
лицо.задайтеИмя("Майер");

System.out.println();
for (object o : objectМассив)
     System.out.println(o.toString())
```

Массив объектов

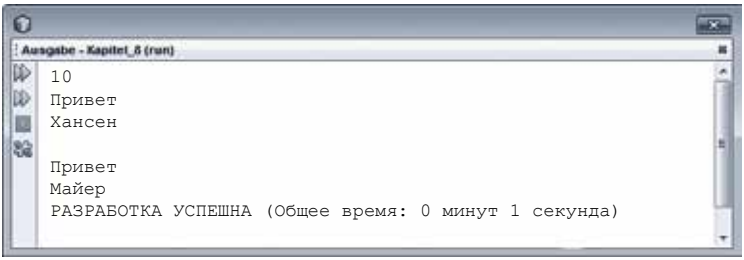
Присваивание первого элемента массива с помощью ***упаковки***

Присваивание цепи символов

Присваивание ссылки

Изменение значений присваивания

После запуска экран выглядит так:



Вывод элементов массива функционирует в нормальном режиме благодаря *полиморфизму*. Присваивание целого значения осуществлено с помощью упаковки и поэтому изменение переменной целого не влияет на элемент массива. Со ссылками это происходит иначе, как и следовало ожидать – соответствующий элемент массива ссылается только на объект класса `Лицо`. И только строка ведет себя не как ссылка, хотя класс `String` это тип ссылки. Это связано с тем, что при присваивании строка всегда резервируется новая область памяти, и там цепь символов сохраняется – старая цепь символов удаляется с помощью программы *garbage collector*, если на нее не существует другой ссылки.

8.1.6 Передача массивов методам

При передаче любых массивов методам следует обратить внимание, что все массивы в Java это типы ссылок – как и массивы простых типов данных. Поэтому изменения элементов массивов в методе всегда влияют на прежний массив, переданный методу. На следующем примере показана эта проблема:

Пример:

```
public class массивы {
   public static void массивПередача(int [][] einArray{
      for (int i=0; i < одинМассив.length; i++)
          for (int j=0; j < одинМассив [0].length; j++)
                          одинМассив [i][j] = (i+1)*(j+1);
}
public static void main(String[] args) {
    int [][] таблица = { {0,0,0,0} , {0,0,0,0} , {0,0,0,0} };

System.out.println("До передачи:");
for (int i=0; i < таблица.length; i++) {
      for (int j=0; j < таблица[0].length; j++) {
              System.out.print(таблица[i][j] + "       ");
      }
      System.out.println();
}
массивпередача(таблица);
System.out.println("После передачи:");
 for (int i=0; i < таблица.length; i++) {
      for (int j=0; j < таблица[0].length; j++) {
         System.out.print(таблица[i][j] + "    ");
      }
      System.out.println();
   }
 }
}
```

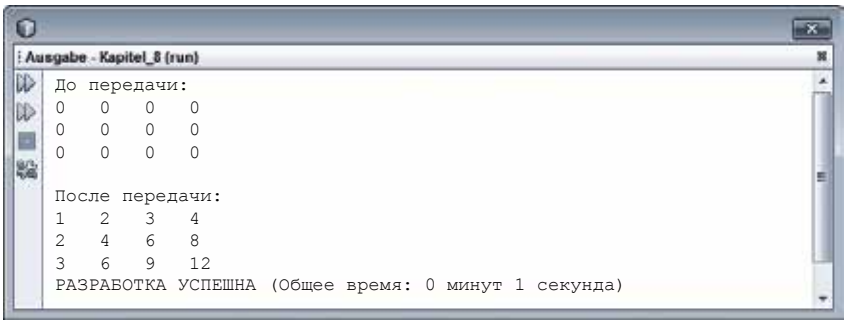
Метод перенимает двумерный массив и изменяет все элементы.

Вывод массива

Передача массива

Вывод массива

После запуска экран выглядит так:



Неопределенное количество передаваемых значений

Один вариант передачи массивов реализуется с помощью троеточия «...». Эта методика известна, прежде всего, программистам языка C применяется, к примеру, в таких функциях языка C, как printf. На следующем примере приведены различия и сходства вышеописанной передачи массивов:

Пример:

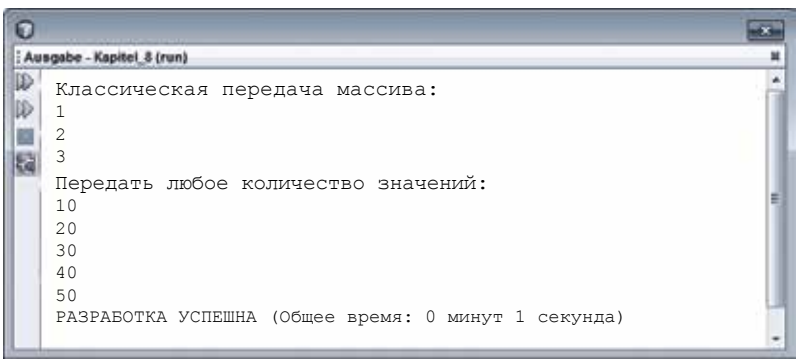
```
public class массивы {
public static void любыеАргументы (int...одинМассив)
        for (int i : одинМассив) System.out.println(i);
}
public static void main(String[] args) {
        int [] одинМассив = {1,2,3};

        System.out.println("Классическая передача массива:");

        любыеАргументы(одинМассив);


        System.out.println("Передать любое количество значений:");
        любыеАргументы (10,20,30,40,50);
 }
}
```

Вариант «Троеточие»

Передача массива

Неограниченное количество передаваемых значений

После запуска экран выглядит так:



Примечания:

• Даже при передаче с помощью троеточия можно получить доступ к переданному массиву:

```
for (int i=0; i < одинМассив.length; i++)
                System.out.println(одинМассив[i]);
```

• Если метод должен перенять несколько параметров, то параметр «Троеточие» всегда должен стоять в конце списка.

```
public static void одинМетод(double x, int ... одинМассив)
```

8.2 Сортировка массивов

На практике массивы часто используются для сохранения измеряемых значений. При оценке измеряемых значений имеет смысл располагать значения в отсортированной последовательности. Определение среднего значения (*медианы*, специальной средней величины) можно осуществить только в отсортированном массиве. Существует множество алгоритмов сортировки, которые работают по-разному и в зависимости от условий быстрее или медленнее. Вид и предварительная сортировка измеряемых значений важны для выбора лучшего алгоритма. Если скорость не имеет значения (что бывает достаточно редко), можно обойтись алгоритмом. В следующих абзацах представлены простой алгоритм и статический метод класса `массивов`. С введением интерфейса `Comparable` можно сортировать любые объекты.

8.2.1 Сортировка методом выбора

Принцип данного алгоритма можно описать в следующих командах:

1. Выполните поиск самого маленького и самого большого элемента в массиве.
2. Поменяйте местами этот элемент с первым элементом массива.
3. Сместите стартовый индекс массива на 1.
4. Повторяйте действия из п. 1–3 до тех пор, пока стартовый индекс не окажется в конце массива.

Пример: Массив с 5 значениями

Индекс	0	1	2	3	4
Значение	10	55	23	18	5

После каждого шага повышается индекс, с которого рассматривается массив. Начальные элементы массива, которые уже не рассматриваются, отмечены серым цветом.

1 Шаг: Поиск минимума → Замена с первым элементом

Индекс	0	1	2	3	4
Значение	5	55	23	18	10

2 Шаг: Поиск минимума от индекса 1 → Замена со вторым элементом

Индекс	0	1	2	3	4
Значение	5	10	23	18	55

3 Шаг: Поиск минимума от индекса 2 → Замена с третьим элементом

индекс	0	1	2	3	4
значение	5	10	18	23	55

4 Шаг: Поиск минимума от индекса 3 → не нужна замена

индекс	0	1	2	3	4
значение	5	10	18	23	55

Так массив отсортирован по возрастающей.

Для применения алгоритма в Java пишутся два статических метода в классе сортировкаВыбор. При этом метод minimumIndex дает индекс минимального значения массива внутри определенных границ массива. Метод Сортировка проводит непосредственно сортировку.

```
class СортировкаВыбор {

      private static int минимумИндекс(int[] сортМассив,
                                       int начало,int конец) {
            int min = сортМассив[начало];
            int minIndex=начало;

            for (int i=начало+1 ; i < конец ; i++){
                 if (min >сортМассив[i])
                     min = сортМассив[i];
                     minIndex=i;
                 }
            }
            return minIndex;
}
public static void сортировка(int [] сортМассив, int количество) {

   int i;
   int minIndex;
   int dummy;
   for (i=0 ; i<количество ; i++)

   minIndex = minimum Index(сортМассив,i,количество);

   if ( i!=minIndex) { dummy = сортМассив[i];

         сортМассив[i]=сортМассив[minIndex];
         сортМассив[minIndex]=dummy;
       }
     }
   }
}

   public class сортировать {
         public static void main(String[] args) {
               int [] одинМассив = {10,55,23,18,5};
     сортировкаВыбор.сортировка(одинМассив, одинМассив.length);

          for (int i : одинМассив) System.out.println(i);
     }
}
```

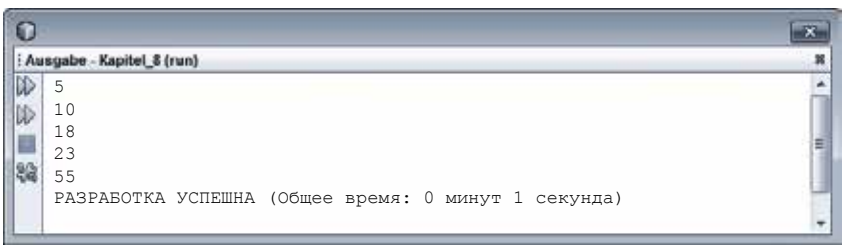
Отметить минимум

Добавить индекс

Последовательно определить минимальный индекс

При разнице – только замена

После запуска экран выглядит так:



Примечание:

Статический метод `сортировка` имеет модификатор `public`, чтобы его можно было выполнить. Метод `minimumIndex` также статический, но имеет модификатор `private`, так как он вызывается только методом `сортировка`, а не из внешнего доступа.

8.2.2 Статический метод сортировки Sort

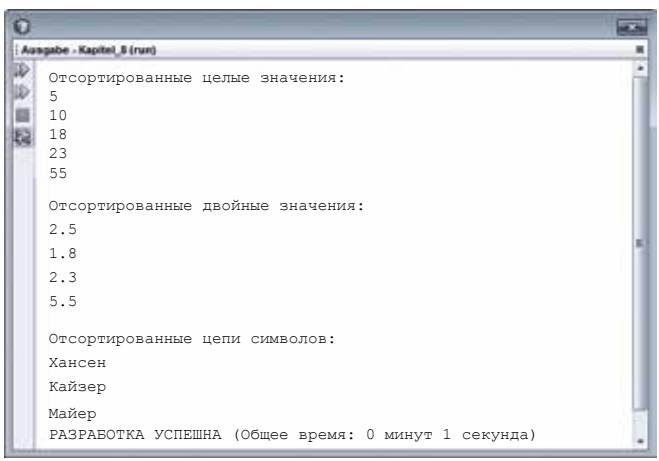
Класс `java.util.Arrays` предлагает статический метод Sort, который может сортировать любые массивы. Это особенно легко для массивов простых типов данных, так как метод `Sort` уже настроен для этого, как показано на примере.

Пример:

```
public class сортировка {
   public static void main(String[] args) {

      int[] intArray = { 10, 55, 23, 18, 5 };
      double[] doubleArray = { 1.5, 5.5, 2.3, 1.8 };
      String[] stringArray = {"Майер","Кайзер","Хансен"};

      java.util.Arrays.sort(intArray);
      java.util.Arrays.sort(doubleArray);
      java.util.Arrays.sort(stringArray);

      System.out.println("Отсортированные целые значения:");
      for (int i : intArray) System.out.println(i);
      System.out.println();
      System.out.println("Отсортированные двойные значения:");
      for (double d : doubleArray) System.out.println(d);
      System.out.println();
      System.out.println("Отсортированные цепи символов:");
      for (String s : stringArray) System.out.println(s);
   }
}
```

Просто передать массивы методу сорт

После запуска экран выглядит так: все массивы отсортированы по возрастающей.

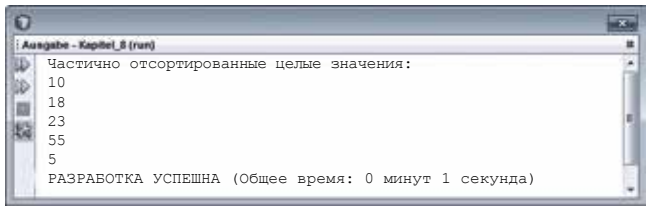


Примечание:

Метод `Sort` может также быть вызван другими параметрами для сортировки только части массива:

```
int[] intArray = { 10, 55, 23, 18, 5 };

    java.util.Arrays.sort(intArray,1,4);

    System.out.println("Частично отсортированные целые значения:");

    for(int i : intArray)
     System.out.println(i);
```

После запуска экран выглядит так: Массив отсортирован от элемента с индексом 1 (вкл.) до элемента с индексом 4 (искл.).



8.2.3 Интерфейс Comparable

Вышеописанный метод `Sort` может сортировать любые массивы простых типов данных, однако не массивы любых объектов, так как у него отсутствует возможность сравнения этих объектов – у простых типов данных эта возможность сравнения присутствует по умолчанию. Однако с помощью определенного метода `compareTo` эта возможность сравнения может быть реализована для объектов. Для этого необходимо исполнить интерфейс `Comparable`, в котором объявляется этот метод. На следующем примере показано исполнение метода в классе Лицо.

Пример:

```
class лицо implements Comparable
:
:

@Override
public int compareTo(object obj) {
        лицо dummy = (лицо) obj;
        return this.name.compareTo(dummy.name);

}
}

public class сортировка {
   public static void main(String[] args) {

        лицо[] лицаМассив = {
                                new лицо("Майер"),
                                new лицо("Кайзер"),
                                new лицо("Хансен") };

java.util.Arrays.sort(лицаМассив);
```

Интерфейс Comparable

Исполнить метод `compareTo`

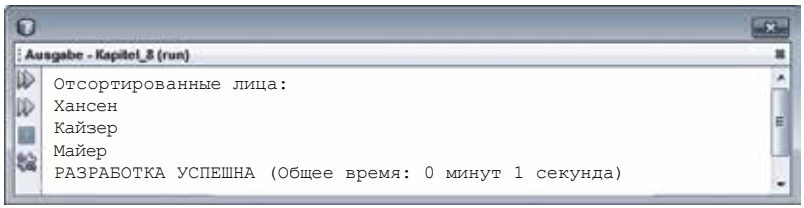
Преобразовать объект передачи в Лицо

Использовать метод `compareTo` для объекта строки и возвратить значение сравнения

Массив лиц теперь можно сортировать с помощью исполненного метода `compareTo`.

```
        System.out.println("Отсортированные лица:");
        for (лицо p : лицаМассив)
            System.out.println(p);
    }
}
```

После запуска на экране появляется отсортированный массив лиц.



Примечания:

Метод `compareTo` должен возвращать целое значение. Это значение показывает результат сравнения объектов:

- Актуальный объект «==» Объект передачи → Возвращаемое значение: 0
- Актуальный объект «<» Объект передачи → Возвращаемое значение: –1
- Актуальный объект «>» Объект передачи → Возвращаемое значение: 1

В вышеприведенном примере использован существующий метод класса строки `compareTo`, который имеет это возвращаемое значение.

8.3 Особые классы массивов

8.3.1 Класс ArrayList

Предыдущие массивы могли быть созданы во время исполнения, то есть с динамичным резервированием памяти, однако изменение размеров во время исполнения было невозможно, то есть невозможно повышение области памяти для сохранения большего количества элементов. Такие мобильные массивы, которые во время исполнения могут принимать и отдавать любое количество элементов, находятся в пакете `java.util`. Один из этих классов – `ArrayList`. Этот класс – простой динамичный класс из списка. Он имеет методы добавления (`add`) и удаления (`remove`) любых элементов. Класс `ArrayList` основан на базовом классе `object`. С помощью него можно сохранять любые значения и объекты (или ссылки).

Пример:

Класс Лицо из предыдущего примера с методом `toString`

```
class лицо { ... }

public class спискиКлассов {
   public static void main(String[] args) {

java.util.ArrayList одинСписок = new java.util.ArrayList();


eineListe.add(10);
eineListe.add(20.5);
eineListe.add("Привет")

for (object obj : одинСписок)
            System.out.println(obj);

System.out.println("Новый элемент Лицо!");
одинСписок.add(2, new лицо("Майер"));
```

Класс **ArrayList**

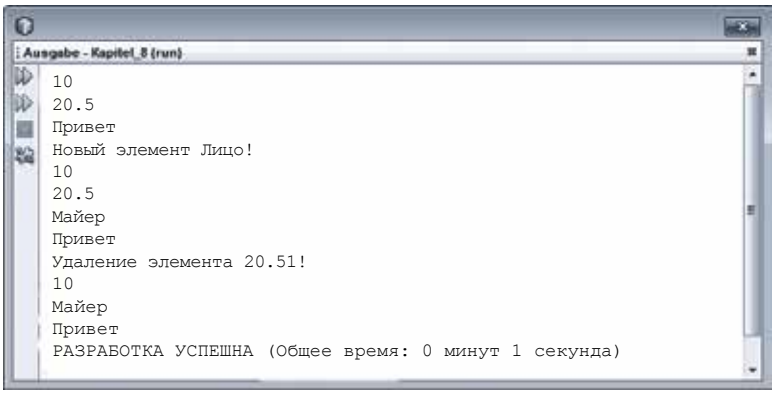
Метод `add` добавляет элемент в конце списка.

Метод `add` с дополнительными параметрами индекса добавляет элемент на позиции индекса.

```
    for (object obj : одинСписок)
                System.out.println(obj);

    System.out.println("Удаление элемента 20.5!");
    одинсписок.remove(20.5);

    for (object obj: одинСписок
            System.out.println(объ);
  }
}
```

Метод remove удаляет заданный элемент.

После запуска экран выглядит так:



Лицо «Майер» добавлено на третьей позиции (индекс 2) и затем удален элемент «20.5».

ВНИМАНИЕ: При удалении всегда удаляется первое значение списка.

8.3.2 Класс HashMap

Класс `java.util.HashMap` реализует так называемый ассоциативный массив, который во многих современных языках программирования уже является стандартом. При этом используется ключ (Key) для сохранения и считывания значений массива. **ВНИМАНИЕ: Этот ключ должен быть однозначным.**

Пример ассоциативного массива:

В массиве нужно сохранить столицы. При этом соответствующая страна должна быть ключом:

Ключ	Значение
Германия	Берлин
Франция	Париж
Швеция	Стокгольм
Австрия	Вена
Нидерланды	Амстердам

С помощью класса `HashMap` эта ассоциация может быть применена на практике. При этом следует учесть, что сохранение значений в `HashMap` осуществляется не в отсортированной последовательности и не в последовательности добавления. Если, например, необходимо вывести отсортированные ключи, то можно дополнительно использовать метод сортировки класса `Массивы`, как показано на примере:

Пример:

Класс HashMap

```
   public class Спискиклассы {
      public static void main(String[] args) {
java.util.HashMap ассоциативныйСписок = new java.util.HashMap();


   ассоциативныйСписок.put("Франция","Париж");
   ассоциативныйСписок.put("Германия","Берлин");
   ассоциативныйСписок.put("Швеция","Стокгольм");
   ассоциативныйСписок.put("Австрия","Вена");
   ассоциативныйСписок.put("Нидерланды","Амстердам");
```

Присвоить ключи и значения с помощью метода put

```
   System.out.println("Без сортировки ключей:");
```

Метод keySet.toArray дает object- массив ключа.

```
   for(object obj : ассоциативныйСписок список.keySet().toArray()
   System.out.println(obj + " " + ассоциативныйСписок.get(obj));
```

Метод get дает значение ключа (здесь obj).

```
   System.out.println();
   System.out.println("С сортировкой ключей:");

   объект [] ключ = ассоциативныйСписок.keySet().toArray();
```

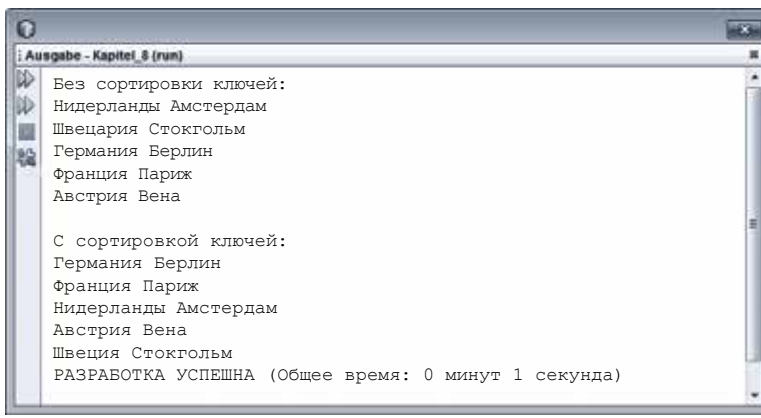
Сохранить ключи в массиве объект

```
   java.util.Arrays.sort(ключ);
```

Сортировать массив ключа

```
   for (object obj : ключ)

   System.out.println(obj + " " + ассоциативныйСписок.get(obj));
   }
}
```

После запуска экран выглядит так:



Примечание:

Конечно, `HashMap` может принимать ключи и значения не только типа данных `String`, но и любые ключи и значения, так как сохраняются только ссылки `object` (как в классе `ArrayList`):

Ключ и значение типа `String`

```
ассоциативныйСписок.put("Нидерланды","Амстердам");
```

Ключ и значение типа `double` и Лицо

```
ассоциативныйСписок.put(1.5 , new лицо("Хансен"));
```

Ключ и значение типа Лицо и `int`

```
ассоциативныйСписок.put(new лицо("Майер") , (int)20 );
```

При сортировке таких разных ключей метод `Sort` класса Массив, конечно, не поможет. Для этого следует вводить собственную логику.

9 Файловые операции в Java

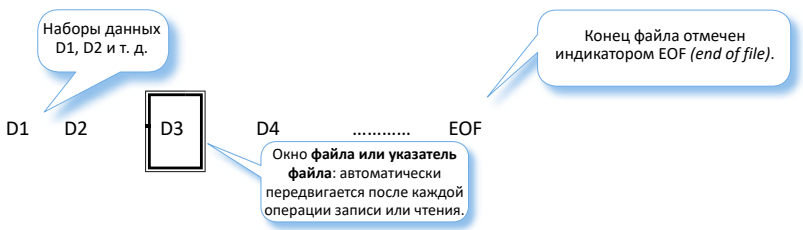
У истоков компьютерной истории хранение данных осуществлялось на перфо картах. Позже использовали магнитную ленту для хранения данных. Также при появлении бытовых компьютеров в 80-е годы стандартным устройством для хранения данных был кассетный магнитофон, на который компьютерные данные передавались в акустической форме. Этот вид хранения данных был не только неудобным, но и чреват ошибками. На сегодняшний день комфортное хранение данных на CD, DVD, внешних жестких дисках или картах памяти уже не имеет много общего с этими устаревшими техниками. Тем не менее, основной принцип хранения данных остался таким же. С помощью таких языков программирования как Java данные можно последовательно писать в файлы, как раньше на магнитную ленту. Это можно представить себе так, что с помощью так называемых Stream-объектов данные могут быть записаны в файлы друг за другом и так же считаны.

Примечание:

Файл (англ. *file*) – это сборник данных или набор данных. Каждый набор данных может состоять из нескольких компонентов. Хранение данных на внешних носителях (жесткие диски и т. д.) производится в форме таких данных. Управление данными при этом осуществляется операционной системой.

В целом можно выделить две организации данных: **последовательную** и **прямую организацию**. При последовательной организации подходит представление о магнитной ленте. Данные последовательно записываются в файл и могут быть считаны только в этой последовательности. Эта форма доступа к данным очень легко программируется. Существенный недостаток – в низкой скорости. Например, в большом файле сначала должно быть считано огромное количество данных для получения искомого набора.

На следующем графике представлен последовательный доступ к файлу. При этом так называемое окно файла (указатель файла) передвигается после каждого доступа на чтение к следующему набору данных.



При прямой организации окно файла имеет четкую позицию, и наборы данных могут считываться из файла в определенном месте. Как последовательный, так и прямой доступ с помощью Stream-объектов возможен в Java.

В пакете **`java.io`** существует множество классов, с помощью которых можно проводить файловые операции. Однако при этом функциональность значительно превосходит возможности чтения и записи файлов. Существуют классы, которые позволяют копировать или перезаписывать файлы, или классы, считывающие и составляющие каталоги. Также возможны обработка архивированных данных и обмен данными через сеть – однако эти сложные темы не могут быть изучены в рамках данной книги.

9.1 Чтение и запись файлов

9.1.1 Последовательное чтение и запись

Как было сказано во введении к данной главе, последовательное чтение и запись можно сравнить с хранением на магнитной ленте. С помощью метода `write` из класса `FileWriter` можно записывать либо отдельные символы, либо цепи символов в файл. После каждой записи указатель файла передвигается вперед на столько ячеек, сколько было записано. Также указатель передвигается на соответствующее количество ячеек при чтении из файла. На следующем примере показано, как можно записывать в файл или читать из файла. Для этого необходимо создать объект класса `FileWriter` и указать в конструкторе имя файла (вкл. путь к файлу). Файл можно открыть для записи (или для изменений).

Запись в файл

Интегрировать пакет `java.io.*`!

```
import java.io.*;
```

ВНИМАНИЕ: Включить обработку особых ситуаций для ошибок IO, подробнее об этом позже!

```
public class файловые операции {
   public static void main(String[] args) throws IOException {
```

Инстанцировать объект класса `FileWriter` и указать конструктору путь и имена файлов, которые необходимо записать.

```
      запись FileWriter = new FileWriter("C:/temp/java.txt");
```

С помощью метода `write` можно писать отдельные символы. При этом целые значения передаются как символы (значение юникода).

```
      for (int i = 65; i < 91; i++) запись.write(i);
```

Имеет смысл писать делать разрыв строки с помощью метода `System.getProperty`. Для этого в каждой системе (например, Linux или Windows) используется правильный символ (или правильные символы).

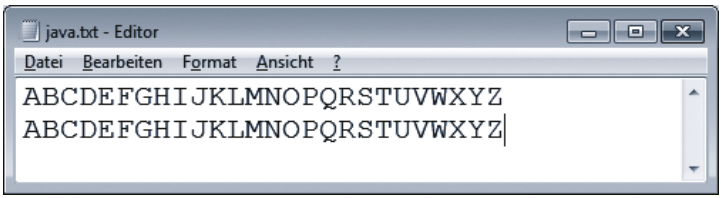
```
      запись.write(System.getProperty("line.separator"));
```

С помощью метода write можно также записать целую строку символов.

```
      запись eiben.write("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
      запись.write(System.getProperty("line.separator"));

      запись.close();
```

ВАЖНО: Закрыть файл, чтобы не использовать ненужные ресурсы!

После запуска файл выглядит так:



Можно видеть, что прописные буквы от A до Z пишутся дважды. Первые 26 букв были записаны с помощью цикла, а второй ряд букв – с помощью строки.

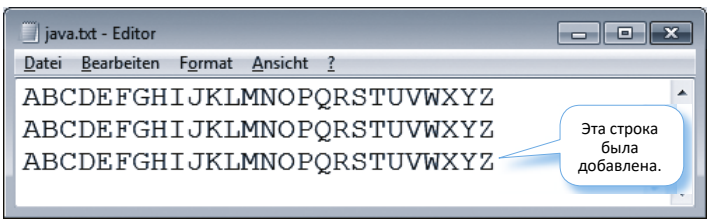
Дополнить данные:

Использовать перегруженный конструктор и установить второй параметр `true`. Так файл откроется для изменений.

```
запись = new FileWriter("C:/temp/java.txt",true);
запись.write("ABCDEFGHIJKLMNOPQRSTUVWXYZ");

запись.close();
```

Добавить следующую строку символов!

После этого файл выглядит так:



Посимвольное чтение файла:

Инстанцировать объект `FileReader` и указать конструктору путь и имена файлов, которые необходимо считать.

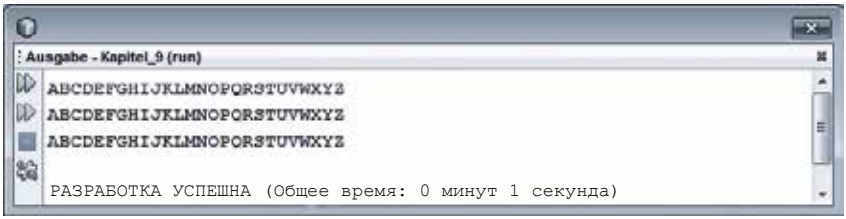
```
чтение FileReader = new FileReader("C:/temp/java.txt");

char [] c = new char[1];
```

Создать массив с элементом, чтобы считать один символ

Считывать с помощью метода `read` такое количество символов, которое соответствует передаваемому массиву – в данном случае один символ. Пока указатель файла стоит на EOF, метод `read` дает значение –1.

```
while (чтение.read(c) != -1) System.out.print(c[0]);
чтение.close();
System.out.println();
```

После запуска файл считывается символ за символом и выводится на экран:



Полностью считать файл:

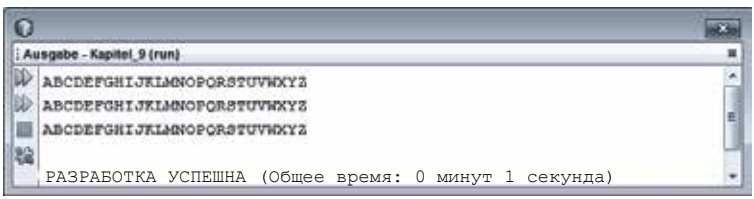
```
Читать FileReader = new FileReader("C:/temp/java.txt");

char [] символ = new char[82];
```

Считать файл полностью в массив!

В файле находятся 3*26 букв (три алфавита) и два разрыва строки, которые в Windows больше символов в два раза. Поэтому в целом требуется 82 элемента.

```
читать.read(символ)
читать.close();

for (int i = 0; i < символ.length; i++)
                    System.out.print(символ[i]);
System.out.println();
```

После запуска файл полностью считывается в массив и выводится на экран:



9.1.2 Прямой доступ к файлам

С помощью класса `RandomAccessFile` можно не только записать и считать, но и позиционировать указатель файла. Так в файле возможен доступ в любом месте, как видно в программе:

```
import java.io.*;

public class файловые операции {
       public static void main(String[] args) throws IOException {
```

Создание объекта типа `RandomAccessFile`

```
RandomAccessFile свободныйДоступ   = new
                  RandomAccessFile("C:/temp/java.txt","rw");
```

Задать путь и имена файлов!

Задать режим:
«r» чтение
«rw» чтение и запись

```
    свободныйДоступ.writeBytes("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
```

С помощью метода writeBytes цепи символов можно записать в файл

```
    свободныйДоступ.close();
```

Открыть файл только для чтения!

```
свободныйДоступ = new RandomAccess FileReader("C:/temp/java.txt","r");

    System.out.println(свободныйДоступ.readLine());
```

С помощью метода readLine цепь символов считывается и возвращается. Метод считывает до следующего разрыва строки или до конца файла.

```
    свободныйДоступ.close();
```

Открыть файл для чтения и записи!

```
    свободныйДоступ = new RandomAccessFile("C:/temp/java.txt","rw");


    long длина = свободныйДоступ.length();
    System.out.println("Указатель файла находится на позиции: " +
                       свободныйДоступ.getFilePointer());
```

С помощью метода length определить длину файла

С помощью метода getFilePointer определить актуальную позицию указателя файла

```
    System.out.println("Длина файла: " + длина);

    свободныйДоступ.seek(длина / 2);
```

С помощью метода seek установить позицию указателя файла

```
    System.out.println("Указатель файла теперь находится на позиции: " +
                            свободныйДоступ.getFilePointer());


    System.out.println("Символ на данной позиции: " +
                           (char) свободныйДоступ.read());
```

Необходима конвертация в char, так как метод read дает целое значение символа.

С помощью метода read считывается отдельный символ в актуальной позиции указателя файла.

```
    System.out.println("Указатель файла теперь находится на позиции: " +
                            свободныйДоступ.getFilePointer());
```

```
свободныйДоступ.write('X');
```

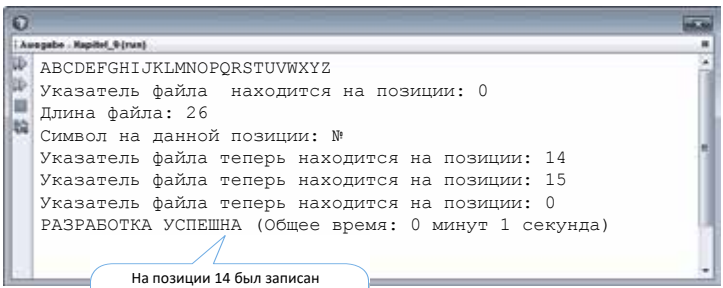
С помощью метода `write` записывается отдельный символ в актуальной позиции указателя файла.

```
System.out.println("Указатель файла теперь находится на позиции: " +
                           свободныйДоступ.getFilePointer());

свободныйДоступ.seek(0);

System.out.println("Указатель файла теперь находится на позиции: " +
                           свободныйДоступ.getFilePointer());

System.out.println(свободныйДоступ.readLine());

свободныйДоступ.close();
}
}
```

Вернуть файл в начало!

Повторный вывод всей строки!

После запуска экран выглядит так:



На позиции 14 был записан «X»!

9.2 Чтение и запись текстовых файлов

Для комфортного написания и чтения чистых текстовых файлов предложено использование классов PrintWriter и Scanner. С помощью них чтение и написание файлов так же просты, как вывод на экран или считывание с клавиатуры.

9.2.1 Запись текстовых файлов с помощью PrintWriter

На следующем примере показано применение PrintWriter, чтобы писать в текстовый файл.

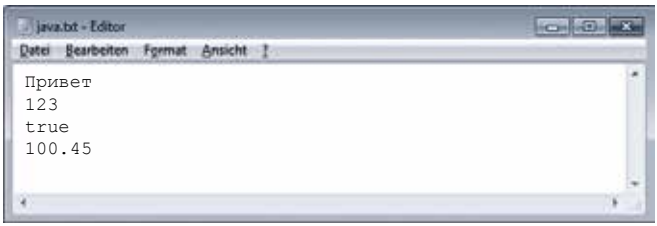
```
import java.io.*;

public class файловые операции {
       public static void main(String[] args) throws IOException {

запись PrintWriter =
          new PrintWriter("C:/temp/java.txt");
```

Создание объекта типа `PrintWriter`

```
        запись.println("Привет");
        запись.println(123);
        запись.println(true);
        запись.println(100.45);
        запись.close();
    }
}
```

Метод println перегружен для всех типов данных!

После запуска текстовый файл выглядит так:



9.2.2 Чтение текстовых файлов с помощью сканера

На следующем примере показано использование сканера и FileReader для форматированного считывания из файла. Для удобства считывается файл «java.txt», который был записан в предыдущем примере.

```
import java.io.*;
import java.util.локальные;
import java.util.сканер;
```

Импортирование классов сканер и локальные, чтобы учесть локальные настройки десятичной системы записи.

```
public class файловые операции {
       public static void main(String[] args) throws IOException {

            сканер чтение = new сканер (
                  new FileReader("C:/temp/java.txt"));
```

Создать объект класса сканер и связать с файлом через FileReader!

Метод nextLine считывает строку символов и возвращает ее!

```
        System.out.println(чтение.nextLine());
```

Метод nextInt считывает целое значение и возвращает его!

```
        System.out.println(чтение.nextInt());
```

Метод nextBoolean считывает логический тип и возвращает его!

```
        System.out.println(чтение.nextBoolean());
```

Метод useLocale меняет формат на заданный. Это точно интерпретирует десятичную дробь.

```
чтение useLocale(Locale.ENGLISH);
```

Метод nextDouble считывает число с плавающей запятой и возвращает значение. **ВНИМАНИЕ:** Учитывать локальные данные!

```
    System.out.println (чтение.nextDouble()
    чтение.close();
   }
}
```

После запуска программа считывает существующий текстовый файл и выводит строки на экран:

9.3 Сериализация объектов

Чтение и запись текстовых файлов имеет смысл, когда необходимо записать и считать много простых данных (по возможности, одного типа). Если же требуется сохранить объекты в памяти, то это в принципе возможно и с помощью вышеописанных методов, однако это может стать сложным, когда, например, дополнительно к объектам необходимо сохранить связи между объектами. Для этих случаев Java предлагает механизм – ***сериализацию***. С помощью определенных потоков ввода и вывода объекты можно сохранять полностью и снова считывать. Чтобы объект можно было сериализовать, нужно только внедрить интерфейс Serializable. Не требуется никаких других методов. На следующем примере показано, как два класса становятся сериализуемыми, и также сохраняются связи между классами:

```
class клиент implements Serializable {

   private String имя;

   public клиент(){
          имя = "ПУСТОЙ";
   }

   public клиент(String n){
          имя = n;
   }

   @Override
   public String toString() {
          return "Имя Клиента: " + имя;
```

Класс клиент вводит интерфейс **Serializable**, не требуется исполнение других методов. Метод toString обычно возвращает строку символов с соответствующими данными.

```
class заказ implements Serializable {
      private клиент клиент;
      private String обозначение;

      public заказ(клиент k, String b) {
       клиент = k;
       обозначение = b;
      }


     @Override
     public String toString() {
          return    "Обозначение: " + обозначение + "     "
                  + клиент.toString();
     }
}
```

Класс Заказ также вводит интерфейс `Serializable`. Дополнительно класс имеет так называемый атрибут связи **клиент**. С помощью него сохраняется ссылка на клиента, которая при инстанцировании объекта также передается конструктору.

Доступ к данным клиента!

Примечание:

Созданная выше связь между клиентом и заказом на профессиональном языке UML называется ассоциацией. Так заказ *знает* своего клиента.

```
import java.io.*;

public class файловые операции {
       public static void main(String[] args)
                  throws IOException,ClassNotFoundException {


    клиент первыйКлиент = new клиент ("Хансен"
    клиент второйКлиент = new клиент ("Майер")

    заказ первыйЗаказ = new заказ (первыйклиент,"Ноутбук");
    заказ второйЗаказ = new заказ (второйклиент,"Настольный ПК");



    System.out.println(первыйЗаказ);
    System.out.println(второйЗаказ);
    }
}
```

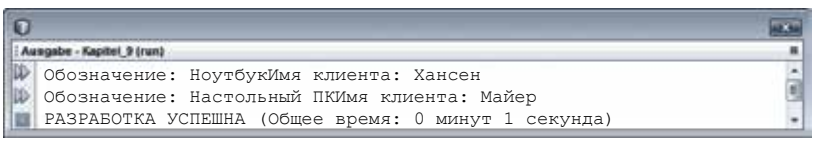
Требуется дополнительная обработка ошибок!

Инстанцирование двух клиентов

Инстанцирование двух заказов и передача ссылок клиента. Так заказы *узнают* своих клиентов.

Вывод заказов (вкл. данные клиента)

После запуска экран выглядит так:



Сериализация:

Создание `ObjectOutputStream` и интеграция с файлом с помощью объекта `FileOutputStream`

```
ObjectOutputStream сериализовать =
        new ObjectOutputStream(
            new FileOutputStream("C:/temp/заказ.xxx"));
```

Указание любого имени файла с любым окончанием (здесь „xxx“)

```
сериализовать.writeObject(первыйКлиент);
сериализовать.writeObject(второйКлиент);
сериализовать.writeObject(первыйЗаказ);
сериализовать.writeObject(второйЗаказ);
сериализовать.close();
```

Защита всех объектов методом `writeObject`

После сериализации файл «заказ.xxx» выглядит так:

```
¬í |sr %kapitel_9.Kunde"6Á6*Š
[¯┐  L ┘namet ǁLjava/lang/String;xpt -Hansensq ~  t |Maiersr ◀ka
pitel_9.Auftrag 6IR1Å^┤ ┐L
bezeichnungq ~  L ■derKundet ◀Lkapitel_
9/Kunde;xpt -Laptopq ~ ┐sq ~ -t
Desktop-PCq ~ ┘
```

Объекты (вкл. связи) сохранены в специальном формате и не могут быть считаны через стандартные файловые операции – для этого используется ***десериализация***.

Десериализация:

Четыре ссылки для принятия объектов из файла!

```
клиент dummyKunde1;
клиент dummyKunde2;
заказ dummyAuftrag1;
заказ dummyAuftrag2;
```

Создание `ObjectIntputStream` и интеграция с файлом с помощью объекта `FileInputStream`

```
ObjectInputStream ввод =
        new ObjectInputStream(
            new FileInputStream("C:/temp/заказ.xxx"));
```

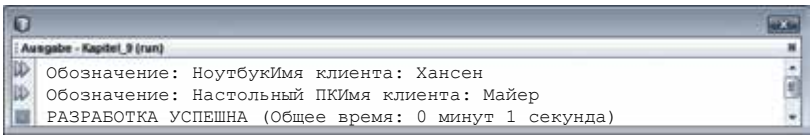
Необходима конвертация в соответствующий тип

Считывание объекта методом `readObject` и присваивание соответствующей ссылке

```
dummyKunde2 = (клиент) ввод.readObject();

dummyAuftrag1 = (заказ) ввод.readObject();
dummyAuftrag2 = (заказ) ввод.readObject();

ввод.close();

System.out.println(dummyAuftrag1);
System.out.println(dummyAuftrag2);
```

Вывод заказов (вкл. данные клиента)

После запуска экран выглядит так:



На экране можно видеть, что с помощью сериализации не только «данные» объекта, но и связи между объектами сохранены корректно.

9.4 Методы класса File

Кроме чтения и записи файлов с помощью потоковых классов класс File предлагает множество методов для работы с файлами и каталогами.

9.4.1 Методы класса File

Объект класса `File` представляет файл или путь. С помощью соответствующих методов можно считывать всю важную информацию. На следующем примере показано применение некоторых важных методов для обработки файла:

Пример:

Создать объект типа `File` и передать конструктору путь и имя файла

```
File Фаил = new File («c:/temp», «java.txt»)

System.out.println("Имя файла: " + файл.getName();
```

Считать имя файла!

```
System.out.println("Путь к файлу: " + файл.getPath();
```

Считать путь файла!

```
System.out.println("Длина файла: " + файл.length();

System.out.println("Файл (да == true / нет == false): "
                          + файл.isFile());
```

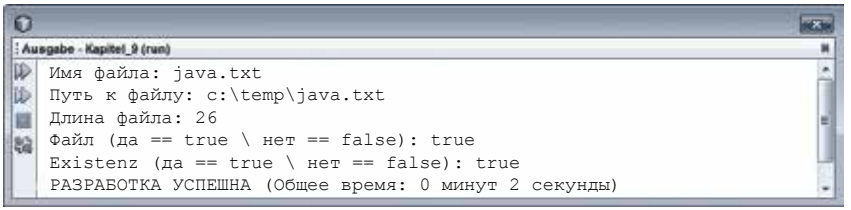
Считать длину файла!

```
System.out.println("Существование (да == true / нет == false): "
                          + файл.exists());
```

Проверить, если это файл!

Проверить, существует ли файл!

После запуска экран выглядит так:



9.4.2 Ведение папок

Класс `File` имеет статический метод `listRoots`, который возвращает все имеющиеся накопители в массиве типа `File`. Каждый из этих накопителей (или папок) может быть далее считан с помощью метода `listFiles`. На следующем примере показано считывание всех накопителей и образец считывания каталога.

Пример:

Создать массив типа `File` и собрать все накопители с помощью `listRoots`

```
File [] накопители = File. listRoots ();

for (File lw : накопители) {
            System.out.println(lw.getPath());

}

File путь = new File("c:/temp/java");

File [] каталоги = путь.listFiles();

for (File vz : каталоги) {
          if (vz.isFile() == true)
              System.out.println("Файл: " + vz.getName());
          else
             System.out.println("Каталог: " + vz.getName());
}
```

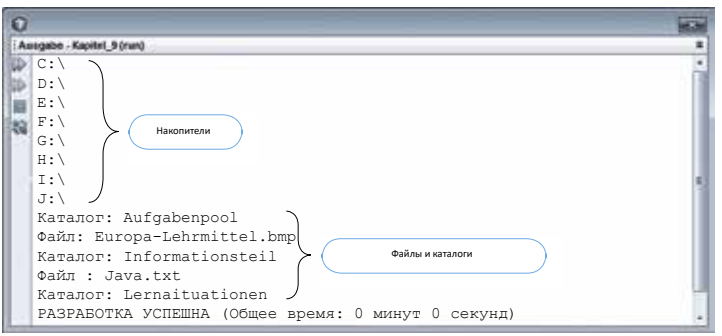
Показать все накопители на экране

Инстанцировать объект типа `File` и задать путь

Создать массив типа `File` и собрать все каталоги и файлы с помощью метода `listFiles`

Идентификация и вывод файлов и каталогов

После запуска экран выглядит так:



Для сравнения копия обзора в Explorer:

Примечание:

Все операции с файлами и каталогами могут быть неудачными, если, например, отсутствует заданный файл, или каталог не может быть создан из-за отсутствия прав. Поэтому файловые операции всегда следует защищать с помощью Exception Handling ситуаций, так как это позволяет управлять и перехватывать ошибки. Обработка Exception Handling будет подробно изучена в следующей главе.

10 Темы Java для продвинутого уровня

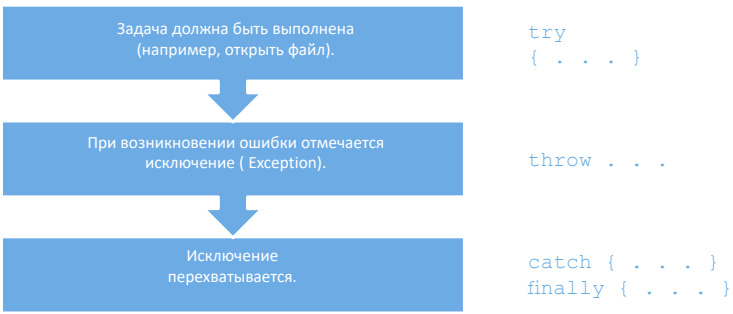
10.1 Исключения – Exceptions

Перехват ошибок – это важная задача в программировании. Зачастую ошибки можно идентифицировать с помощью метода возвращаемых значений. Недостаток этого метода в том, что программист сам решает, производит ли он оценку возвращаемых значений или ошибок и реагирует ли на них или нет.

Возможные источники ошибок:

- Превышение зарезервированной области массива
- Деление на ноль
- Ввод неожиданных символов с клавиатуры
- Ошибки файловых операций
- Ошибки доступа к базе данных

Обработка исключений в Java помогает решить эти проблемы. При этом обработка ошибок отделяется от самого программного кода. На рисунке представлен процесс обработки исключений:



10.1.1 Исключения и перехват ошибок (try and catch)

Обработка исключений начинается с так называемого блока `try`. Внутри этого блока находится программный код, который может вызвать ошибки, – поэтому ключевое слово `try` обозначает попытку. В следующем примере необходимо считать число с клавиатуры. Однако если пользователь вводит буквы вместо чисел, будет «вызвано» исключение.

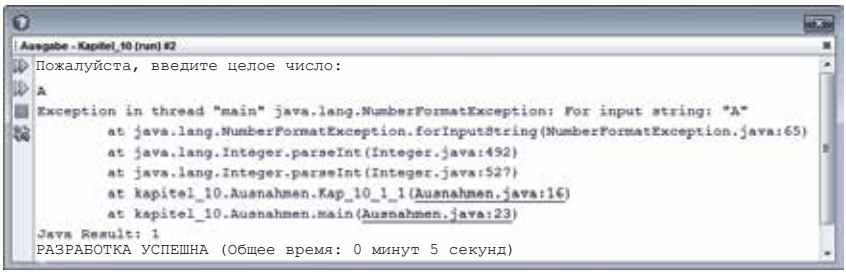
Пример:

Уже знакомая инициализация системной обработки исключений для ошибок IO

```
public class Исключения {
   public static void main String [] args) throws IOException {

  int x;
  BufferedReader ввод      =      new BufferedReader(new
                                  InputStreamReader(System.in));
```

```
    System.out.println("Пожалуйста, введите целое число: ");


    x = Integer.parseInt(Ввод.readLine());

    System.out.println("Ввод: " + x);
    }
}
```

После ввода букв программа дает сбой с сообщением об ошибке:



Данная ситуация для пользователя программы очень неприятна. Программист либо следит за безошибочным вводом (когда, он, к примеру, считывает только в переменных `String`), либо он самостоятельно программирует обработку исключений в Java, что показано на следующем примере:

Пример:

ВНИМАНИЕ: Здесь больше не требуется обработка исключений, так как теперь исключения перехватываются сами!

```
public class исключения {
   public static void main(Str ing[] args) {

int x;
BufferedReader ввод     =     new BufferedReader(new
                              InputStreamReader(System.in));
System.out.println("Пожалуйста, введите целое число: ");
```

В блоке `try` проводятся критические операции.

```
try {
     x = целое.parseInt(ввод.readLine());
     System.out.println("Ввввод: " + x);

}
```

Ошибка типа `Exception` Передается автоматически.

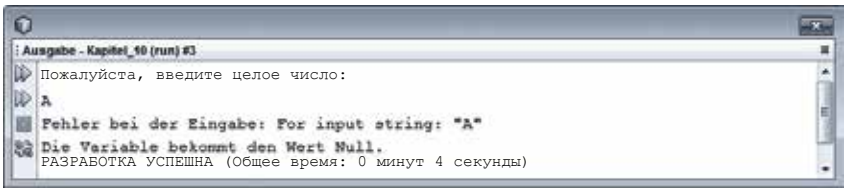
В случае возникновения ошибки (исключения) она под контролем обрабатывается в блоке `catch`.

```
catch (Exception e) {
System.out.println("Ошибка при вводе: " +
                   e.getMessage());
```

Вид ошибки может быть считан с помощью метода `getMessage`.

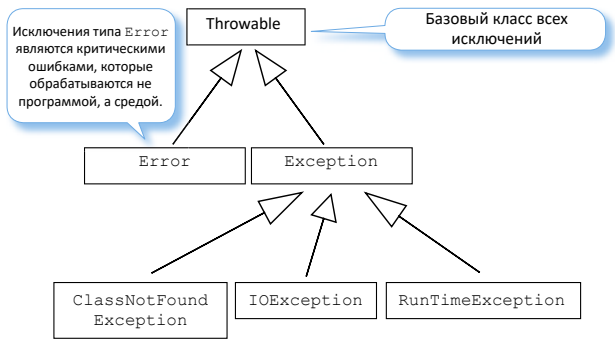
```
        System.out.println("Переменная получает значение ноль.");
    x = 0;
    }

    }
}
```

После запуска и ввода буквы вместо цифры исключение перехватывается в блоке `catch` и программа не дает сбой:



10.1.2 Системные исключения

Базовым классом всех исключений является класс `Throwable`, от которого наследуются классы `Error` и `Exception`. Класс `Exception` уже был использован в предыдущем примере. На следующей схеме представлен обзор важных классов:



Блок `catch` с параметром класса `Exception` перехватывает каждую ошибку (кроме исключений `Error`). Тем не менее, перехват очень неспецифичен. Если нужно сделать обработку ошибок более дифференцированной, то можно предварительно включить другие блоки `catch`, в которых есть специальные классы. `Exception` в качестве параметров. Некоторые из этих классов представлены в таблице:

Классы Exception, производные от типа IOException:

Класс Exception	Описание
`FileNotFoundException`	Исключение вызывается, если отсутствует файл, который нужно открыть для чтения, записи или других операций.
`CharConversionException`	Исключение вызывается, если при конвертации символов возникает проблема.
`EOFException`	Исключение вызывается, если файл необходимо считать через последний отличительный символ.
`NotSerializableException`	Исключение вызывается при попытке сериализации объекта, который не внедрил интерфейс `Serializable`.

Классы Exception производные от типа RunTimeException:

Exception класс	Описание
`ArithmeticException`	Исключение вызывается, если возникает арифметическая ошибка (например, целое значение делится на ноль).
`IndexOutOfBoundsException`	Исключение вызывается, если индекс массива или строки используется ошибочно (например, выходит за пределы).
`NullPointerException`	Исключение вызывается, если вместо объекта используется нулевая ссылка.
`SecurityException`	Когда доступ осуществляется в обход правил безопасности, вызывается это исключение (например, доступ к файлу в апплете).

На следующем примере показано использование некоторых из этих классов для дифференцированного реагирования на ошибку:

Пример:

```
int индекс;
int [] значения = new int [5];
BufferedReader ввод   =   new BufferedReader(new
                          InputStreamReader(System.in));

 try {

 System.out.println("Пожалуйста, введите индекс:");
 индекс = целое.parseInt(ввод.readLine());
 значения[индекс] = 100;
 FileReader читать = new FileReader("C:/temp/nichtvorhanden.txt");
 читать.close();
 }

 catch(IndexOutOfBoundsException индексИсключение) {
       System.out.println("Ошибочный индекс");
       System.out.println ("Сообщение об ошибке: " +
                   индексИсключение.getMessage());
        System.out.println();
 }

 catch (FileNotFoundException файлИсключен
        System.out.println("Файл не найден");
        System.out.println("Сообщение об ошибке: " +
                   файлИсключение.getMessage());
        System.out.println();
 }

 catch(Exception другиеисключения) {
       System.out.println("Общие ошибки: " +
                   другиеИсключения.getMessage());
       System.out.println();
 }

 System.out.println("Программа функционирует ...");
 System.out.println();
```

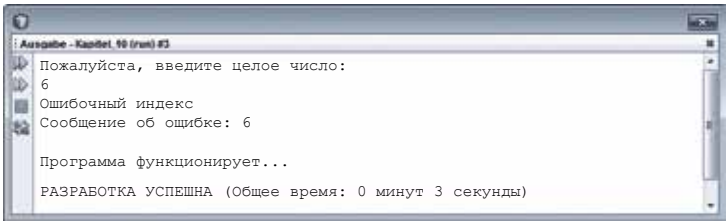
Возможные причины ошибок

Реакция на ошибочный индекс

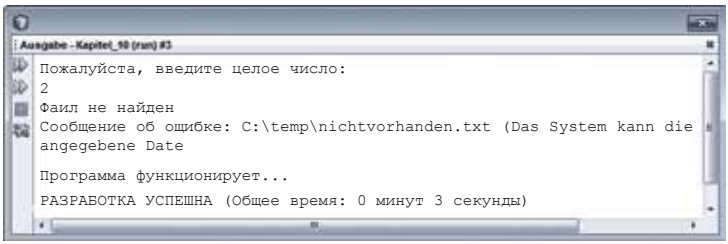
Реакция на ошибку файла

Реакция на другие ошибки

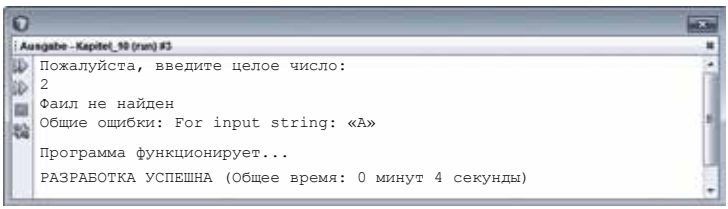
После запуска сначала выводится ошибочный индекс:



Выполняется соответствующий блок catch с Exception для ошибочного индекса. В последующем вводится корректный индекс, однако файл отсутствует:



Другие ошибки перехватываются в общем блоке catch, например, ввод буквы:



Примечание:

На экранах видно, что при ошибке в блоке `try` вызывается исключение и перехватывается подходящим блоком `catch`. После вызова исключения операции в блоке `try` больше не производятся.

10.1.3 Финальный блок

Если после блока `try` обязательно необходимо произвести определенные операции, можно выполнить так называемый блок `finally` после блоков `catch`. Этот блок выполняется всегда, вне зависимости от того, было ли вызвано исключение или нет.

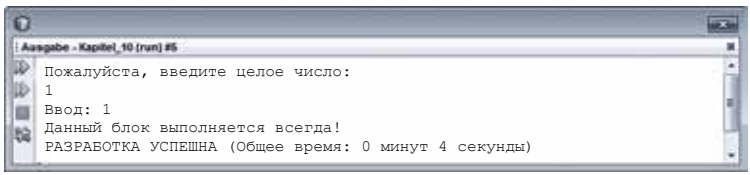
Пример:

```
int x;
BufferedReader ввод  =  new BufferedReader(new
                        InputStreamReader(System.in));

System.out.println("Пожалуйста, введите целое число: ");
```

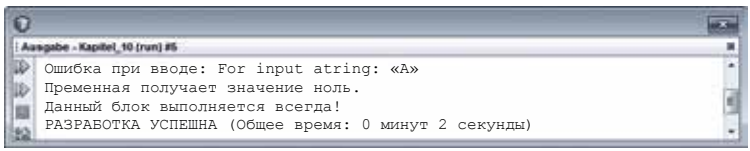
```
try {
        x = целое.parseInt(ввод.readLine());
        System.out.println("Ввод: " + x);
}
catch(Exception e) {
      System.out.println("Ошибка ввода: " +
                         e.getMessage());
finally {
System.out.println("Данный блок выполняется всегда!");
}
```

Блок finally выполняется всегда.

После ввода цифры не вызывается исключение, а выполняется блок finally:



После ввода буквы вызывается исключение, выполняется блок catch и блок finally.



10.1.4 Вызов исключений

Все предыдущие исключения вызывались средой автоматически. Однако также возможно вызвать исключение в явном виде, чтобы, например, структурировать обработку ошибок. Для этого можно вызвать исключение с помощью команды throw. Необходимо только задать экземпляр объекта класса Exception.

На следующем примере показан явный вызов и перехват исключения с помощью throw, когда пользователь вводит число ноль, потому что на ноль делить нельзя.

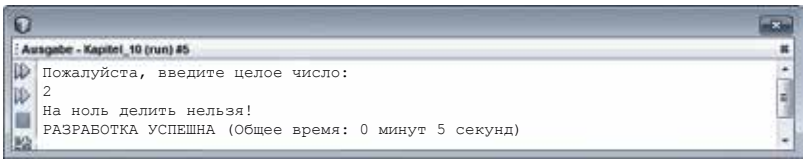
Пример:

```
int x = 10;
int число;
int y;
BufferedReader ввод = new BufferedReader(new
                      InputStreamReader(System.in));

System.out.println("Пожалуйста, введите целое число: ");
try {
        число = целое.parseInt(ввод.readLine());
        if (число == 0) throw (new ArithmeticException());
        y = x / число;

catch(ArithmeticException неделимыйИсключение) {
         System.out.println("Деление на ноль запрещено!");
```

Явный вызов исключения

```
catch(Exception e) {
            System.out.println("Недействительное число!");
            число = 1;
}
```

После запуска вследствие ввода числа ноль вызывается исключение:



10.1.5 Создание собственных классов исключений

Дифференцированное реагирование на ошибки можно использовать еще лучше, если определить характерные классы исключений. С помощью них можно вызывать и перехватывать определенные исключения. Обработка ошибок в программе становится еще более структурированной, и существенно повышается надежность программы. Собственные классы исключений должны быть производными класса `Exception` и иметь стандартный конструктор и конструктор с параметрами для соответствующей передачи ошибки базовому классу. На следующем примере создается собственный класс исключений, вызываемый при ошибочном вводе.

Пример:

```
class EigeneException extends Exception {

public собственноеИсключение
       super();
}


public собственноеИсключение (сообщение String) {
       super(сообщение);
}
```

Класс, производный от `Exception`.

Вызывается конструктор базового класса.

Вызывается конструктор с параметрами базового класса и сообщение передается.

Теперь этот новый класс исключения используется для перехвата ошибок ввода:

```
int x = 10;
int число;
int y;
BufferedReader ввод = new BufferedReader(new
                      InputStreamReader(System.in));
System.out.println("Пожалуйста, введите целое число: ");

try {
       число = целое.parseInt(ввод.readLine());
       if (число == 0)
              throw (new EigeneException("Ошибочный ввод"));
       y = x / число;
}
```

Вызвать исключение с передачей ошибки конструктору с параметрами класса.

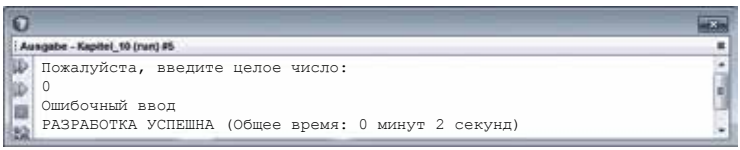
```
        catch(собственноеException собственноеИсключение) {
        System.out.println(собственноеИсключение.getMessage());
}
```

Для IO-Ошибок должен быть также общий блок.

Новый класс исключений наследовал метод `getMessage`.

```
        catch(Exception e){
              System.out.println("Другая ошибка");
}
```

После запуска вызов исключения выглядит так:



Сообщение «Ошибочный ввод» передано конструктору с параметрами базового класса Exception и поэтому может быть показано с помощью метода `getMessage`.

Примечание:

При исключении ведется поиск и выполнение подходящего блока `catch`. Если подходящий блок не найден, то исключение передается вызываемой среде. Это происходит до тех пор, пока не будет найден блок или исключение не попадет на более высокий уровень и программу, возможно, придется закрыть.

10.2 Обобщенное программирование

Под обобщенным программированием понимают разработку методов или классов, содержащих один или более типов данных (типов параметров), которые необходимо точно определить только при инстанцировании объекта или при использовании метода. Преимуществом обобщенного программирования является мобильность, так как обобщенный класс или метод в идеале принимает все типы данных как параметр. Также возможность повторного использования обобщенных классов и методов намного выше повышается и эффективность разработки ПО.

10.2.1 Обобщенные методы

Обобщенный метод использует один или несколько типов параметров, которые заменяются только при вызове с соответствующим типом данных. Следующий метод определяет, являются ли два переданных значения одинаковыми.

Пример:

В угловых скобках указывается тип параметра (здесь `T`).

```
public class Generics{

  public static <T> boolean равный (T значение1. T значение2){
        if (значение1.equals(значение2))return true;
        return false;
}

public static void main(String[] args) {
        int a = 5;
        int b = 6;

        double x = 1.5;
        double y = 1.5;
```

Здесь перенимаются два значения типа `T`.

Создать четыре переменные, которые необходимо протестировать на равнозначность.

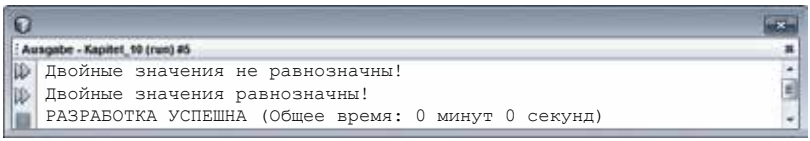
Вызов статического метода путем передачи целых значений. Компилятор автоматически заменяет символ-заполнитель T типом данных int.

```
if равнозначный(a,b)== true)
        System.out.println("Целые значения равнозначны!");
else
        System.out.println("Целые значения не равнозначны!");
```

Вызов статического метода путем передачи двойных значений. Компилятор автоматически заменяет символ-заполнитель T типом данных double.

```
if (равнозначный(x,y)==true)
     System.out.println("Двойные значения равнозначны!");
else
System.out.println("Двойные значения не равнозначны!");
     }
}
```

После запуска экран выглядит так:



На экране можно видеть, что обобщенный метод функционирует в нормальном режиме как с переменными целых, так и с переменными двойных значений.

10.2.2 Обобщенные классы

При обобщенном классе сразу после имени класса указывается тип параметра (или несколько параметров, отделенных запятой) и далее он присутствует в классе. На следующем примере класс имеет параметр T, созданный как частный атрибут класса. Конструктор класса принимает значение (или ссылку) и выводит содержание с помощью метода toString. Дополнительно выводится тип принятого значения.

Пример:

Указать параметр типа сразу после имени класса

Частный атрибут типа T

Конструктор с параметром T

Определить значение и тип

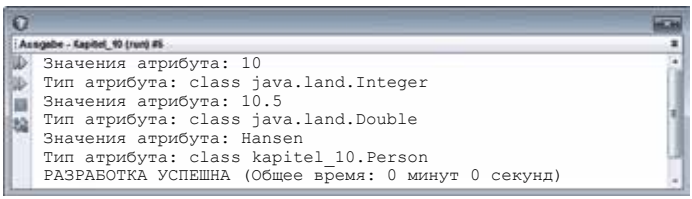
```
class Пример <T> {

        private T атрибут;

        public Пример (T param){

            атрибут = Param;
            System.out.println("Значение атрибута: " +
                                атрибут.toString());
            System.out.println("Тип атрибута: " +
                                атрибут.getClass().toString());
}
public class Generics {
    public static void main(String[] args) {
```

ВНИМАНИЕ:
Параметры обобщенного класса всегда должны быть типами ссылок, то есть нельзя использовать `int` или `double`.

```
        Пример<целое> целОбъект =
                          new Пример <целое>(10);
        Пример<двойное> двойнойОбъект =
                          new Пример <двойное>(10.5);

        Пример<лицо> лицоОбъект =
                          new Пример <лицо>(
    }                                     new лицо("Хансен"));
}
```

Создать ссылку с типом лица. Класс Лицо знаком из предыдущих глав и также имеет метод `toString`.

После запуска экран выглядит так:



Обобщенный класс управляет одним объектом классов ***Wrapper*** Integer или Double или одним объектом класса Лицо.

10.2.3 Использование обобщенных списковых классов

Динамичное хранение любого количества значений или ссылок объектов в объекте спискового класса уже было изучено в главе о массивах. При этом использовались списковые классы, которые работают с базовым классом объект. Преимущества этих списковых классов проявляются, прежде всего, в мобильном использовании (добавление любых значений и ссылок). Однако, если ясно, что речь идет о хранении значений (или ссылок) только одного типа, лучше использовать обобщенные списковые классы. Они функционируют по аналогии с описанными списковыми классами, однако служат для хранения значений одного типа. На следующем примере показано применение обобщенных классов **`ArrayList`** <> :

Пример:

```
ArrayList <Integen> целСписок = new ArrayList <Integen>();
ArrayList <Person> лицаСписок = new ArrayList <Person>>();
```

Использовать обобщенный класс **`ArrayList`** и задать соответствующий тип

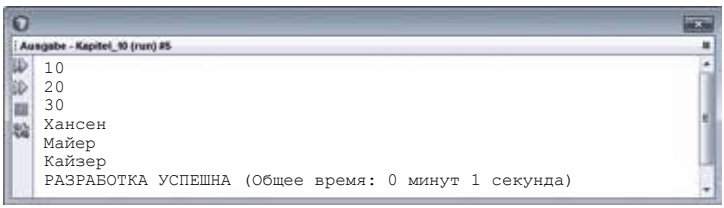
```
целСписок.add(10);
целСписок.add(20);
целСписок.add(30);

лицаСписок.add(new лицо("Хансен"));
лицаСписок.add(new лицо ("Майер"));
лицаСписок.add(new лицо ("Кайзер"));
```

Упаковка!

Добавлять любое количество значений соответствующего типа

```
for (int i : целСписок) {
        System.out.println(i);
}

for (лицо p : лицаСписок) {
        System.out.println(p);
}
```

После запуска экран выглядит так:



Примечание:

Вследствие определения типа при инстанцировании списка допускаются элементы только данного типа. Попытка добавления любого элемента будет неудачной:

```
целСписок.add(new лицо("Кайзер"));
```

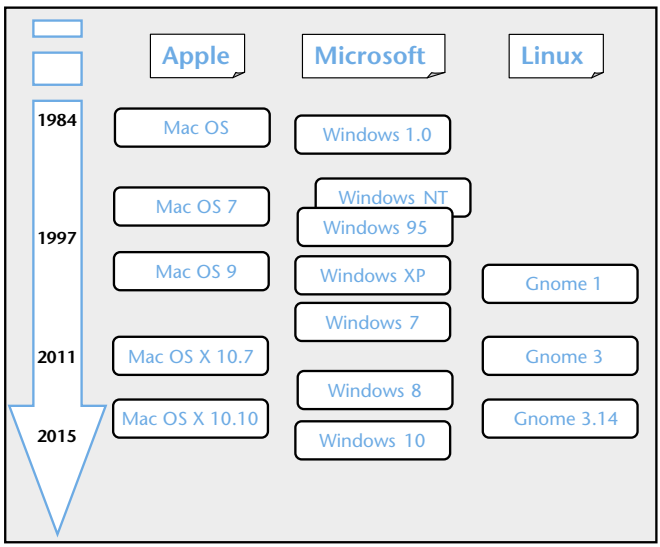
ОШИБКА:
К целСписок можно присваивать только целые значения!

11 GUI-программирование с помощью AWT

11.1 GUI-программирование

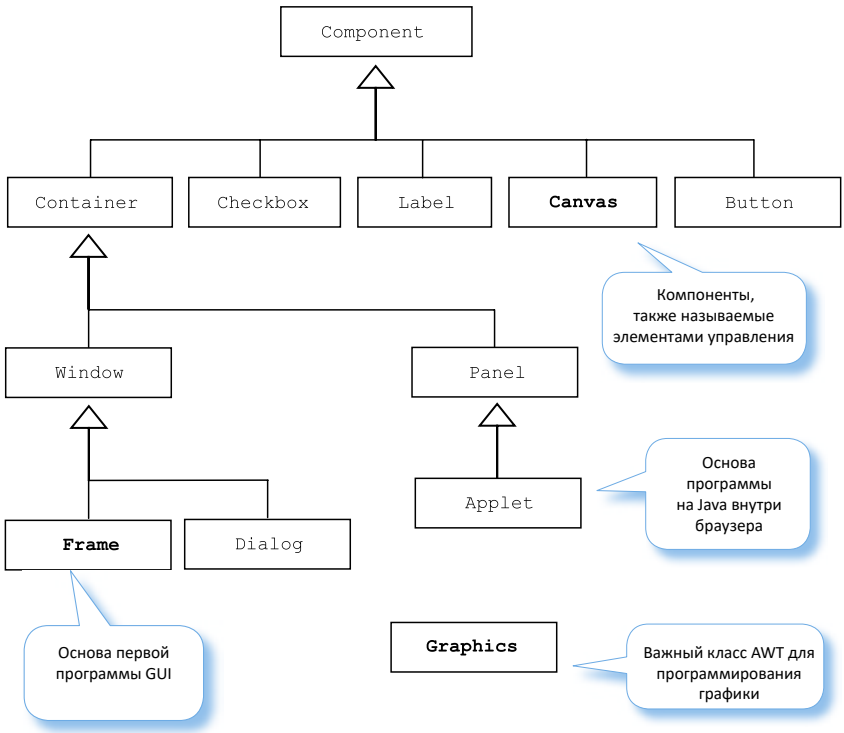
11.1.1 История GUI-программирования

У истоков возникновения компьютеров коммуникация с компьютером осуществлялась с помощью перфокарт. В 60-х годах на рынок вышли первые компьютеры, похожие на ПК, имевшие клавиатуру и вывод на экран, с помощью которого иногда можно было вывести только три строки. Позже компьютеры были оснащены ЭЛТ-мониторами, позволявшими вывести на экран несколько строк и столбцов. Этот вывод похож на консольный вывод, который использовался в данной книге. В 70-х годах были запущены первые испытания графического интерфейса пользователя **GUI** (**G**raphical **U**ser **I**nterface). В большинстве случаев эти испытания не удавались в связи с недостатком производительности компьютеров. Только в 80-х годах *Apple Macintosh* был представлен компьютер, который был доступным и обладал достаточной производительностью для поддержки графического интерфейса пользователя *Mac OS*. GUI имел много аспектов, которые хорошо знакомы современному пользователю – прежде всего, без использования мыши сегодня не обойтись. Через какое-то время компания Microsoft представила первую версию Windows, которая, однако, имела большой успех только через несколько лет с версиями *Windows 95* и параллельно разработанной *Windows NT-Linie*. На схеме ниже представлено развитие графического интерфейса пользователя во времени (вкл. Linux-GUI *Gnome*):



11.1.2 Структура AWT

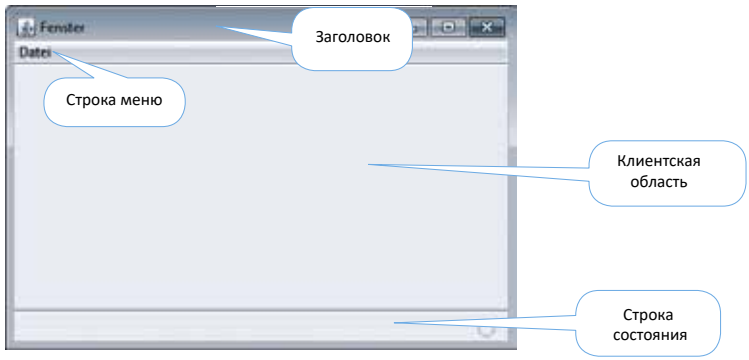
Программы GUI основаны на так называемых окнах, которые служат как для вывода, так и для взаимосвязи с пользователем. Для этой взаимосвязи существует множество компонентов и предварительно изготовленных элементов, которые хорошо знакомы компьютерному пользователю (неважно, пользователю Windows, Linux или Apple). Эти компоненты (или классы) объединены в пакете **AWT** (**A**bstract **W**indow **T**oolkit). *Toolkit* в GUI-программировании получает доступ к встроенным GUI-компонентам соответствующей операционной системы. Поэтому внешний вид AWT-программы адаптируется под соответствующую операционную систему. У этого есть и преимущества (известность), и недостатки – если, например, ПО должно иметь собственный стиль. Для таких случаев могут быть использованы ***Swing***-классы (подробнее об этом позже). На схеме ниже представлена копия из иерархии классов AWT.



В первых примерах преимущественно были использованы классы `Frame`, `Graphics` и `Canvas`. Позже вводятся классы элементов управления для интеграции таких элементов, как `Button` или `Checkbox`.

11.1.3 Основные понятия GUI-программирования

Базой GUI-приложения является окно. Кроме заголовка окно также может иметь строку состояния и строку меню. Окно окружено рамкой. Внутри окна находится область, где выводится содержание. Эта область называется **клиентской областью**.



11.2 Первая программа GUI

11.2.1 Использование класса Frame

Первая GUI программа, в принципе, очень короткая. Она состоит только из инстанцирования объекта класса **java.awt.Frame** и метода вывода (setVisible) для вывода окна. Исходный код GUI-программ

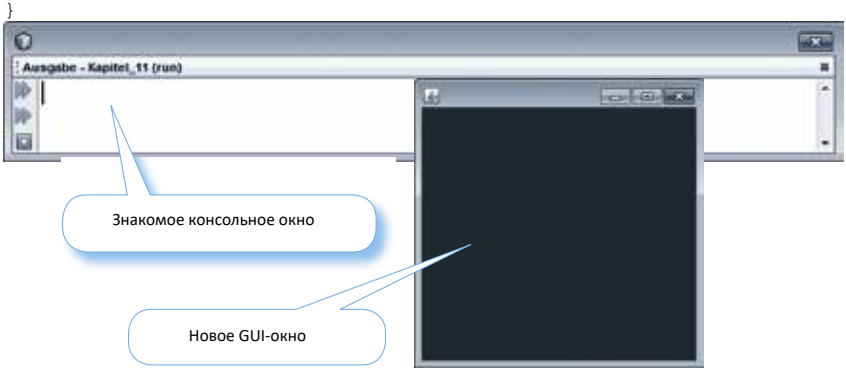
```
package глава_11;
import java.awt.*;

public class GUIStart{
        public static void main(String[] a
                Frame окно = new Frame();


                окно.setVisible(true);

                окно.setBackground(Color.BLUE);
                окно.setSize(300, 300);
                окно.setLocation(400, 300);
        }
}
```

Интеграция пакета AWT

Инстанцирование объекта Frame

Вывод окна

Установить фон, размер и стартовое положение



Консольное окно, конечно, не является центральным объектом изучения. Тем не менее, оно полезно именно в начале тем, что существует, поскольку может быть использовано для ввода и вывода параллельно с GUI-окном. Позже проектная форма ***NetBeans*** будет изменена (на *десктопные приложения*), и тогда консольное окно больше не будет иметь значения.

Внимание:

После запуска очевидно, что окно не закрывается. Это связано с тем, что не был создан объект-событие, который реагирует на событие «закрыть окно». Поэтому для удобства используется консоль для ожидания ввода. Далее окно можно закрыть с помощью метода `dispose`:

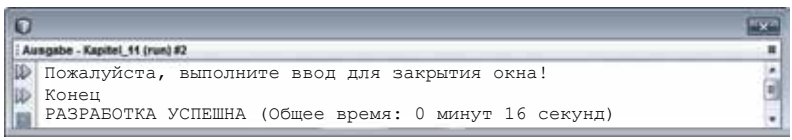
```
package глава_11;
import java.awt.*;
import java.io.*;
public class GUIStart
   public static void main(String[] args) {

     Frame окно = new Frame();
     окно.setVisible(true);
     окно.setBackground(Color.BLUE);
     окно.setSize(300, 300);
     окно.setLocation(400, 300);

     String ввод;
     BufferedReader считать  = new BufferedReader(new
                               InputStreamReader(System.in));

     System.out.println("Пожалуйста, выполните ввод для
                         закрытия окна!");

     try {
         ввод = считать.readLine();
     }
     catch(Exception e) {
           System.out.println("Ошибка: " + e.getMessage());
     }

     окно.dispose();
  }
}
```

Интеграция пакета IO для ввода с клавиатуры

Ожидание ввода

Закрыть окно с помощью dispose

После запуска снова появляются два окна. Далее после любого ввода окно закроется:



11.2.2 Написание собственного класса Frame

В первой GUI-программе был инстанцирован объект класса `Frame`. С помощью него можно создать и вывести окно. Однако возможности индивидуальной адаптации окна очень ограничены. По этой причине целесообразно создать собственный класс `Frame`, производный от класса `Frame`. Так будут доступны все основные функции и возможны дополнения. На следующем примере показан собственный класс `Frame`, описывающий конструктор, который уже работает над небольшой адаптацией:

```
package глава_11;
import java.awt.*;
import java.io.*;

class окно extends Frame {
   public окно() {
      this.setBackground(Color.BLUE);
      this.setSize(300, 300);
      this.setLocation(400, 300);
    }
}

public class GUIStart {
      public static void main(String[] args) {

      окно окно = new окно();
      окно.setVisible(true);

      String ввод;
      BufferedReader считать  = new BufferedReader(new
                                InputStreamReader(System.in));

      System.out.println("Пожалуйста, выполните ввод
                         для закрытия окна!");

      try {
         ввод = считать.readLine();
      }
      catch(Exception e) {
            System.out.println("Ошибка: " + e.getMessage());
      }

       окно.dispose();
    }
}
```

Создать собственный класс-окно, производный от класса `Frame`

Доступ к существующим характеристикам окна по ссылке `this`, и определение размера, стартового положения и фонового цвета

Инстанцирование объекта Окно

11.3 Вывод текстовой и графической информации

11.3.1 Paint и первый вывод текста

Особенность GUI программирования в том, что окно не закреплено на экране в одном месте, а может менять свой размер. Его также можно закрыть или частично перекрыть другими окнами. Когда окно меняет свой размер или повторно вызывается пользователем на передний план, оно должно быть в состоянии повторно вывести свое содержание. Операционная система хоть и берет на себя управление окном, однако **не несет ответственности за содержание**. Как только операционная система узнает, что окно изменило свой размер, или было повторно возвращено на передний план, она отправляет определенное сообщение приложению. Или на языке событийно-ориентированного программирования: вызывается определенное событие. Это событие является **Paint-событием**. Вызов Paint-события в приложении отвечает за то, что содержание должно быть выведено повторно. Для этого случая существует виртуальный метод, который должен быть перезаписан в собственном классе `Frame` – а именно метод `paint`.

```
package глава_11;
import java.awt.*;
class окно extends Frame {
       public окно() {
          this.setSize(300, 300);
       }
```

Перезапись метода paint()

```
@Override
public void paint(Graphics g)}
        g.drawString("Привет, GUI", 100, 100);

    }
}
```

Метод paint получает так называемый контекст устройства типа Graphics. С помощью него возможны такие графические операции, как запись цепи символов с помощью drawString на определенную позицию.

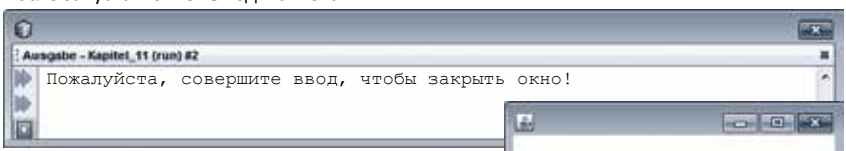
Метод paint вызывается именно тогда, когда вызывается Paint-событие. Метод имеет передаваемый параметр типа Graphics. Этот параметр получает ссылку на контекст-устройство. Контекст-устройство можно представить себе как карандаш, который предоставляет операционная система, чтобы делать чертеж в клиентской области. Далее с помощью метода drawString выводится цепь символов:

Координаты для вывода

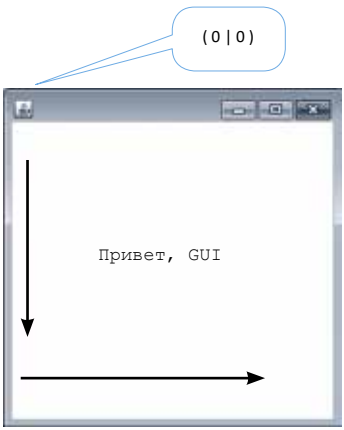
```
g.drawString("Привет, GUI", 100, 100);
```

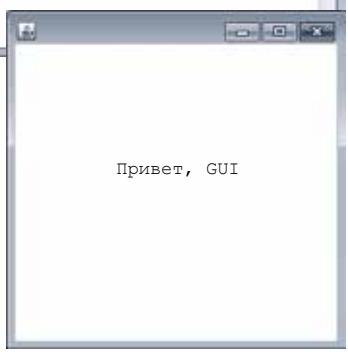
Выводимая цепь символов

После запуска в окне выводится текст:



Примечание





Система координат окна отличается от привычной математической системы координат. В левом верхнем углу находится точка (0 | 0). В правом нижнем углу находится максимальное значение(MAX_X | MAX_Y).

Если окну (или другим компонентам) дается команда вызвать метод paint для нового графика клиентской области, то это можно легко осуществить путем вызова метода repaint.

11.3.2 Добавление клиентской части

Вышеприведенные примеры показывают, как можно делать запись в клиентской области окна. Проблема в этом методе в основных координатах. Они начинаются не в левом верхнем углу клиентской области, а относятся ко всему окну. Это приводит к сложностям при точном расчете вывода, который, к примеру, должен начинаться в левом верхнем углу клиентской области. По этой причине целесообразно определить вид новой клиентской области и прошить в окно. Эта клиентская область основывается на классе `Canvas`. Этот класс предлагает вид пустой поверхности, которую можно использовать для графиков или обработки событий. Этот новый экран необходимо только добавить к окну, и далее он заполнит всю клиентскую область. Далее метод `paint` будет перезаписан, как обычно.
На следующем примере показано определение класса `Клиентская область` внутри класса окна. Он имеет преимущество: вшитый класс можно использовать только в окружающем классе. Однако также существует возможность внешнего описания класса `Клиентская область`. Так он будет доступен многим классам окон.

```
class окно extends Frame {

      public окно() {
             this.setSize(200, 100);
             this.setLocation(400, 300);

             this.add (new клиентская область());
      }


class клиентская область extends Canvas {

             @Override
             public void paint(Graphics g) {

            int высотаШрифта = this.getFont().getSize();
            g.drawString("Привет, Canvas", 0, высотаШрифта);
           }
      }
}
```

С помощью метода `add` добавляется клиентская область и вшивается в окно.

Описание класса `клиентская область` как `внешнего класса` и наследование от класса `Canvas`

Перезапись метода `paint` как обычно

Получение точной высоты шрифта с помощью метода `getSize`

Расположение вывода цепи символов в левом верхнем углу клиентской области

После запуска текст в клиентской области выводится следующим образом:

Новая клиентская область!

11.3.3 Простой графический вывод

С помощью методов класса `Graphics` в клиентской области любые элементы можно представить в графической форме. Кроме таких простых графических элементов, как линии и прямоугольники, можно также графически изображать такие сложные элементы, как полигон (многоугольник) или сектор круга. На следующем примере представлено использование некоторых простых методов класса `Graphics`:

```
package глава_11;
import java.awt.*;
class графикОкно extends Frame {

public графикОкно() {

            this.setLocation(400, 300);

            this.add (new клиентскаяОбласть());
}

class клиентскаяОбласть extends Canvas {

@Override
public void paint(Graphics g) {

g.setColor(Color.BLUE);

g.drawLine(50, 90, 130, 90);
g.drawLine(160, 90, 240, 90);

g.drawOval(50, 100, 80, 80);

g.fillOval(170, 110, 60, 60);

g.drawOval(160, 100, 80, 80
g.fillOval(170, 110, 60, 60);
g.setColor(Color.BLACK);


g.drawRect(140, 130, 10, 70);

g.drawARC (50, 150, 200, 100, 0, -180);
```

Определить с помощью метода `setColor` новый цвет символа

Начертить линию с помощью метода `drawLine`

Определить начальные и конечные координаты для линии
ВНИМАНИЕ: Ширину линии нельзя настроить – альтернатива: начертить заполненные прямоугольники или использовать классы `Java2D`.

Начертить эллипс с помощью метода `drawOval`

Определить начальные координаты, высоту и ширину окружающего прямоугольника

Начертить заполненный эллипс с помощью метода `fillOval`

Начертить прямоугольник с помощью метода `drawRect`

Начертить дугу с помощью метода `drawArc` (см. примечания)

Прямоугольник

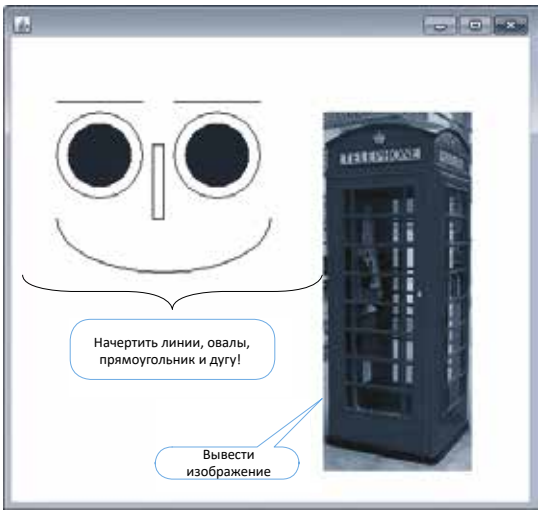
Начальный угол и угол расстояния

С помощью метода `getImage` объекта `Toolkit` загружается изображение.

```
        Imge bild = getToolkit().getImage("c:/temp/telefon.jpg");
        g.drawImage(bild, 300, 100, this);

    }
}
}
```

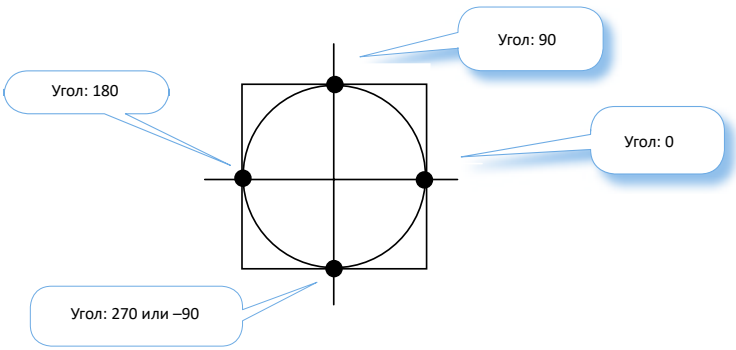
Вывод изображения с помощью метода `drawImage`

После запуска программа выглядит так:

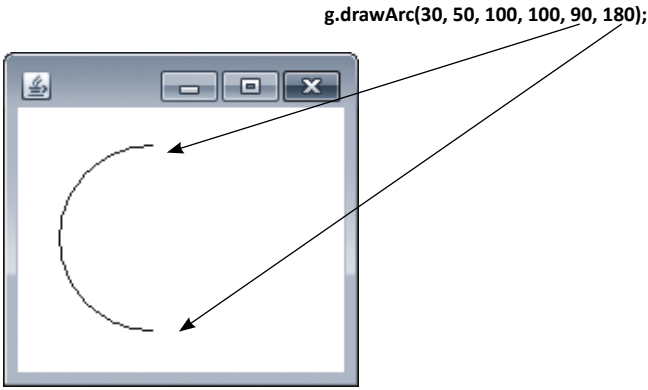


Примечание:

В методе для дуги указываются начальный угол и угол расстояния, описывающие дугу внутри прямоугольника. При этом начальный угол начинается в середине правой стороны окружающего прямоугольника с нуля. Снизу углы задаются с отрицательными значениями:

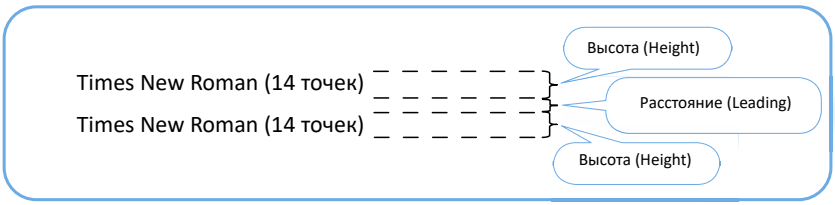


На следующем примере изображена дуга в виде левого полукруга. Начальный угол равен 90, а конечный угол равен 90 + 180 = 270:

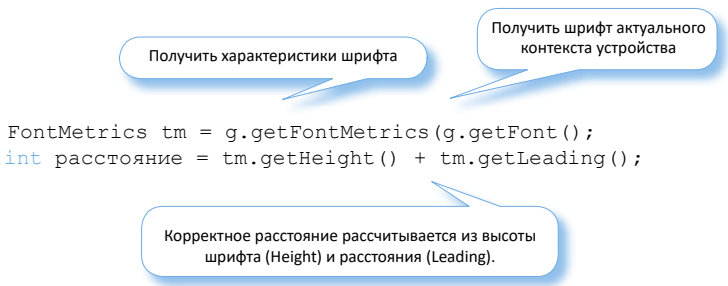


11.3.4 Многострочный вывод текста

С помощью метода `drawString` текстовая строка может быть расположена прямо в окне. Если необходимо вывести несколько текстовых строк друг под другом, то важна высота шрифта для определения соответствующего интервала между строками. Эту высоту легко можно считать определенными методами. Для этого важно знать, что каждый вид шрифта имеет свои особенности, которые определяют не только высоту, но и подходящее расстояние между двумя строками этого вида шрифта.



Высоту и расстояние можно считать по ссылке `FontMetrics`:

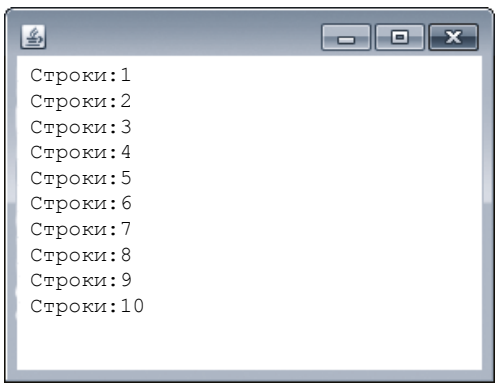


С помощью расстояния теперь можно вывести любое количество текстовых строк, как показано на примере:

```
@Override
public void paint(Graphics g) {

            FontMetrics tm = g.getFontMetrics(g.getFont());
            int расстояние = tm.getHeight() + tm.getLeading();

            for (int i = 1; i <= 10 ; i++) {

                g.drawString("Строки: " + i, 0 , i * расстояние);
            }
}
```

Координата `y` рассчитывается из расстояния и переменной `i`.

После запуска десять текстовых строк корректно выводятся друг за другом:



Установка новых видов шрифта

Предыдущий вывод использует стандартный шрифт системы. Установка новых видов шрифта производится очень просто. Перед выводом в контекст-устройстве необходимо установить новый шрифт. При этом выводятся имя шрифта, маркировка (жирный, курсив, обычный и т. д.) и размер.

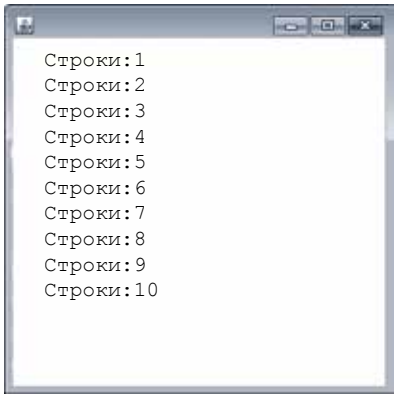
Маркировка (здесь, например, жирный и курсив)

```
g.setFont(new Font("Lucida Handwriting",Font.BOLD + Font.ITALIC, 20));
```

Имя шрифта

Размер в точках (pt)

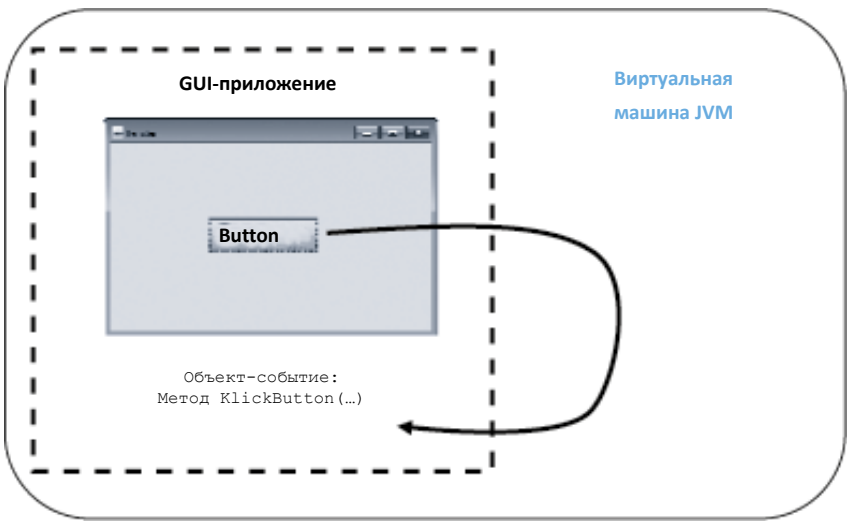
После устновки нового шрифта вывод на экран предыдущего примера будет выглядеть так:



11.4 Событийно-ориентированное программирование

11.4.1 Основа событийно-ориентированного программирования

Предыдущие консольные программы взаимодействовали с пользователем через ввод с клавиатуры. При этом консольная программа ожидает до тех пор, пока пользователь не совершит ввод. Только потом совершаются следующие операции. В GUI-программировании используется другой концепт для взаимодействия с пользователем – а именно, оно происходит с помощью событийно-ориентированного программирования. Пользователь может при этом производить различные операции (например, нажимать клавишу) и с таким событием будет связан объект, предлагающий метод для реагирования на это событие. GUI-программа, как вид цикла, ожидает наступления события, которое нужно будет обработать. При этом события могут быть не только те, которые вызывает пользователь, но и те, которые могут быть вызваны операционной системой (как Paint-событие). На следующей схеме эта связь представлена наглядно:



GUI-приложение функционирует внутри виртуальной машины JVM, которая, конечно, работает в операционной системе и взаимодействует с ней. Нажатие клавиши регистрируется операционной системой и приложение или клавиша получает соответствующее сообщение. Это сообщение приводит к тому, что вызывается определенный метод объекта-события, который, создан именно для этого случая.

11.4.2 Типы событий и приемники событий

Для окна (или клиентской области типа `Canvas` или других элементов управления) существует множество различных событий, на которые можно реагировать. Некоторыми важными событиями являются:

- Действия мыши
- Действия клавиатуры
- События окна-состояния

В целом операционная система передает событие соответствующему окну или элементу управления (подробнее об этом позже). Поэтому в классе окна должен быть так называемый приемник события (англ. *listener*), ответственный за обработку события. Такой приемник затем исполняет соответствующий метод для обработки события.

На следующей таблице представлены важнейшие приемники:

Приемник	Описание
`ActionListener`	Этот класс ожидает событие типа `ActionEvent`. Оно происходит при нажатии клавиши.
`FocusListener`	Этот класс ожидает событие типа `FocusEvent`. Например, получение окном фокуса.
`MouseListener`	Этот класс ожидает событие типа `MouseEvent`. Например, действия пользователя с мышкой.
`KeyListener`	Этот класс ожидает событие типа `KeyEvent`. Например, действия пользователя с клавиатурой.
`WindowListener`	Этот класс ожидает событие типа `WindowEvent`. Например, изменение состояния окна.

На следующем примере исполняется приемник события типа `MouseListener`. Для этого описывается внутренний класс приемникМышь, который внедряет интерфейс `MouseListener`. Объект класса приемникМышь затем регистрируется в классе окно с помощью метода `addMouse-Listener`. Таким образом, класс подготовлен к событию типа `MouseEvent`:

```
class событиеОкно extends Frame {
      public событиеОкно () {
            this.setSize(400, 600);
            this.setLocation(400, 300);

            this.addMouseListener(new приемникМышь());
}
```

Регистрация экземпляра класса приемникМышь как `Listener`.

Внутренний класс приемникМышь исполняет интерфейс `MouseListener`.

```
      class MasAbhoerer implements MouseListener {
```

Перезапись метода `MausClicked`, чтобы реагировать на щелчок мыши.

```
@Override
public void mouseClicked(MouseEvent e) {
```

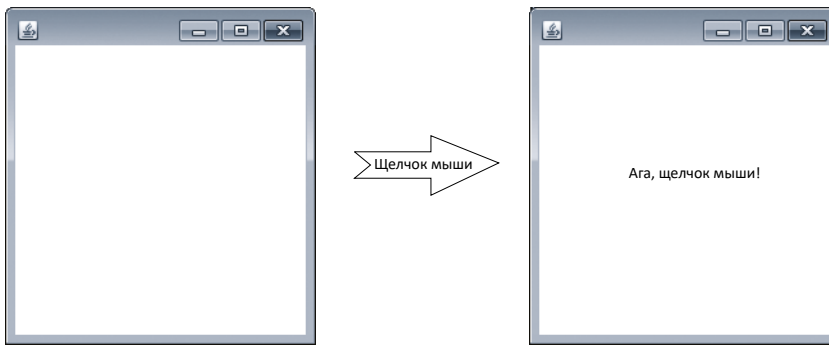
```
        getGraphics().drawString("Ага, щелчок мыши!", 100, 100);
    }
```

Метод `getGraphics` дает контекст устройства, с которым можно записать строку символов.

Все методы интерфейса необходимо перезаписать!

```
    @Override
    public void mousePressed(MouseEvent e) {}
    @Override
    public void mouseReleased(MouseEvent e) {}
    @Override
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}
}
}
```

После запуска приложение реагирует на щелчок мыши:



Использование классов адаптера

Введение класса ПриемникМышь показало, что должны быть введены все методы соответствующего интерфейса (даже с пустым телом). Это иногда может быть очень затратно. По этой причине существуют так называемые классы адаптера. Эти классы исполнили все методы пустыми. Поэтому такой класс, как ПриемникМышь, необходимо только произвести от такого класса-адаптера и перезаписать методы, которые действительно используются. На следующем примере показано применение класса-адаптера:

Класс ПриемникМышь наследует класс-адаптер и перезаписывает только необходимый метод.

```
class ПриемникМышь extends MouseAdapter
      @Override
      public void mouseClicked(MouseEvent e) {
        getGraphics().drawString("Ага, щелчок мыши!", 100, 100);
      }
}
```

На следующей таблице представлены классы-адаптеры для приемников и методов событий:

Приемник	Класс адаптера	Методы
`ActionListener`	**Отсутствует, так как существует только один метод**	`actionPerformed(ActionEvent)`
`FocusListener`	**FocusAdapter**	`focusGained(FocusEvent)` `focusLost(FocusEvent)`

`MouseListener`	**MouseAdapter**	`mouseClicked(MouseEvent)`
		`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`

`KeyListener`	**KeyAdapter**	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

`WindowListener`	**WindowAdapter**	`windowActivated(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowClosing(WindowEvent)`
		`windowDeactivated(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowGainedFocus(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowLostFocus(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowStateChanged(WindowEvent)`

Теперь, наконец, можно выполнить и закрытие окна. Для этого необходимо только один класс-приемник и перезапись метода `windowClosed`:

```
    class событиеОкно extends Frame {
          public событиеОкно() {
                this.setSize(400, 600);
                this.setLocation(400, 300);

                this.addWindowListener(new окноПриемник());
    }

    class ОкноПриемник extends WindowAdapter {
          @Override
          public void windowClosing(WindowEvent e) {
                System.exit(0);
          }
    }
}
```

Регистрация экземпляра объекта класса ОкноПриемник как `Listener`

Теперь класс ОкноПриемник наследует класс-адаптер и перезаписывает необходимый метод.

`System.exit` закрывает приложение.

Событие «закрыть окно» обрабатывается.

Альтернативное создание приемников событий

Кроме создания с помощью внутреннего класса существуют также другие возможности создания, как показано в следующих примерах:

Вариант 1: инстанцирование анонимного класса прямо при регистрации

```
class вариантАнонимныйКласс extends Frame {
      public вариантАнонимныйКласс () {
             this.setSize(400, 600);
             this.setLocation(400, 300);

             this.addWindowListener( new WindowAdapter () {
                                          @Override
                                 public void windowClosing(WindowEvent e){
                                        System.exit(0);
                                 }
                          }
                    );
      }
}
```

Инстанцирование анонимного класса

Вариант 2: Определить сам класс окна как приемник события

```
class вариантОкноПриемник extends Frame implements WindowListener {

      public вариантОкноПриемник () {

          this.setSize(400, 600);
          this.setLocation(400, 300);


          this.addWindowListener(this);
}

public void windowClosing(WindowEvent e) {
       System.exit(0);
}

public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowGainedFocus(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowLostFocus(WindowEvent e){}
public void windowOpened(WindowEvent e){}
public void windowStateChanged(WindowEvent e){}
}
```

Внедрение интерфейса

Регистрация самого экземпляра объекта окна как *приемника*

Перезапись метода закрытия

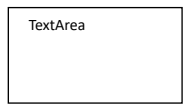
Все методы интерфейса должны быть перезаписаны, так как не произведены от адаптера

12 Элементы управления с помощью AWT и классов Swing

12.1 Элементы управления с помощью AWT

12.1.1 Простые элементы управления

Простые элементы управления (или компоненты) служат для отображения данных или взаимодействия с пользователем. Некоторые важные элементы управления приведены в данной таблице:

Элемент управления	Вид	Описание
`Button`		Пользователь может щелкнуть по этому элементу управления, и, как правило, вызвать действие.
`Checkbox`		Этот элемент управления служит для проставления галочки, например, для выбора опции.
`TextField`		Этот элемент управления служит для ввода и вывода текстовых строк.
`TextArea`		Этот элемент управления служит для ввода и вывода нескольких текстовых строк. В принципе, `TextArea` сравним с простым редактором.
`Label`		Этот элемент управления служит для надписей. Зачастую надписи также создаются на полях ввода (`TextFields`).
`Canvas`		Этот элемент управления уже известен. Это вид пустого экрана.

12.1.2 Использование элементов управления

В предыдущей главе уже применялся элемент управления типа `Canvas` для симуляции *клиентской области*. Создание других элементов управления функционирует аналогично. Сначала необходимо инстанцировать объект желаемого типа, затем добавить к окну (или другому *контейнеру*) с помощью метода **`add`**.
К примеру, так создается элемент управления типа `Button`:

```
class элементУправленияButton extends Frame {

        private Button кнопка = new Button();
         public элементУправленияButton() {

           this.setSize(200, 200);
           this.setLocation(400,300);

           this.setLayout(null);
```

Создать частный атрибут типа `Button`

Выключить Layoutmanager. Так элементы управления могут позиционироваться самостоятельно. Разные возможности расположения будут представлены позже.

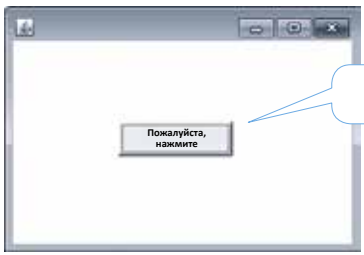
```
        кнопка.setLabel("Пожалуйста, щелкните");
        кнопка.setBounds(100, 100, 100, 3 )
        add (кнопка);
            this.addWindowListener(new окноПриемник());
    }
    class окноПриемник extends WindowAdapter {
          @Override
          public void windowClosing(WindowEvent e) {
             System.exit(0);
          }
       }
}
```

Создать надпись на Button

Добавить окну элемент управления с помощью add

Расположить Button (100|100) и указать ширину (100) и высоту (30)

После запуска окно выглядит так:



Button размещен и надпись создана.

12.1.3 Реагирование на события

Добавление элемента управления хоть и показывает в целом принцип функционирования, но не помогает при взаимодействии с пользователем. По этой причине снова необходима запись приемников, которые могут реагировать на события. На следующем примере показано, как Button из предыдущего примера может реагировать на щелчок:

```
кнопка.setLabel("Пожалуйста, нажмите");
кнопка.setBounds(100, 100,30)
```

Добавить приемник события для Button

```
кнопка.addActionListener(ken") 100, 30)
                          new ActionListener()
                          {
                         public void actionPerformed(ActionEvent e){
                              кнопка.setLabel("Новый текст!");
                         }
                      }
                   );
this.add(кнопка);
```

Реакция на щелчок. В этом случае Button получает новую надпись.

После запуска окно выглядит так:

12.1.4 Пример приложения с простыми элементами управления

Следующий пример программы продемонстрирует, как можно создать и использовать в приложении простые элементы. При этом производится адаптация вида шрифта и цвета элементов управления. Также показано, как может происходить взаимодействие с пользователем. Для этого служит класс `Анкета`, который представляет простую анкету с оцениванием.

```
class анкета extends Frame {
      private Button отправитьBtn = new Button();
      private Label заголовокLab = new Label();

      private Label nameLab = new Label();
      private TextField имяТекстF = new TextField();


      private Checkbox javaChkbox = new Checkbox();
      private Checkbox csharpChkbox = new Checkbox();
      private Checkbox cppChkbox = new Checkbox();

      private TextArea summaryTextA = new TextArea();
      public анкета() {

            this.setSize(500, 650);
            this.setLocation(300.200);
            this.setLayout(null);

          заголовокLab.set Font(new Font("Verdana",Font.BOLD,20) );
          заголовокLab.setForeground(Color.BLUE);


          заголовокLab.setBounds(20, 40, 120, 40);
          заголовокLab.setText("Анкета");

          nameLab.setFont(new Font("Verdana",Font.PLAIN,16) );
          nameLab.setBounds(20, 100, 200, 30);
          nameLab.setText("Пожалуйста, введите имя:");

          имятекстF.setFont(new Font("Verdana",Font.PLAIN,16) );
          имятекстF.setBounds(230, 100, 150, 30);
```

Создать `Button`

Создать `Label`

Создать `TextField`

Создать несколько `Checkbox`

Создать `TextArea`

Установить новый вид шрифта для элемента управления с помощью метода `setFont`

Установить цвет шрифта с помощью метода `setForeGround`

```
    javaChkbox.setFont(new Font("Verdana",Font.PLAIN,16));
    javaChkbox.setBounds(20, 150, 150, 30);
    javaChkbox.setLabel("Знание Java");

    csharpChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
    csharpChkbox.setBounds(20, 200, 150, 30);
    csharpChkbox.setLabel("Знание C#");

    cppChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
    cppChkbox.setBounds(20, 250, 150, 30);
    cppChkbox.setLabel("Знание C++");

    отправитьBtn.setFont(new Font("Verdana",Font.PLAIN,16) );
    отправитьBtn.setBounds(280, 250, 100, 40);
    отправитьBtn.setLabel("Отправить");

    отправитьBtn.addActionListener(
                 new ActionListener()
                 {
                public void actionPerformed(ActionEvent e) {
                    итог();
                }
            }
            );

    summaryTextA.setFont(new Font
    "Verdana",Font.BOLD + Font.ITALIC,20) );
    summaryTextA.setForeground(Color.BLUE);
    summaryTextA.setBounds(20, 350, 360, 250);
    summaryTextA.setEditable(false);

    this.add(заголовокLab);
    this.add(nameLab);
    this.add(nameTextF);
    this.add(javaChkbox);
    this.add(csharpChkbox);
    this.add(cppChkbox);
    this.add(отправитьBtn);
    this.add(summaryTextA);

    this.addWindowListener(new окноПриемник());

}

class окноПриемник extends WindowAdapter {
    @Override
    public void windowClosing(WindowEvent e) {
    System.exit(0);
    }
}

private void итог());

    String name = имяТекстF.getText();
    String знания =  "Вы знаете
                     следующие языки:";
```

Метод события вызывает частный метод. Это более структурно.

Элемент управления устанавливается как «не редактируемый».

Добавить все элементы управления

Частный метод, вызываемый методом события.

Вызов содержания `TextField` помощью метода `getText`

Вызвать состояние Checkbox с помощью метода getState

```
        if (javaChkbox.getState()== true)
             Знания = знания + "-Java\

        if (csharpChkbox.getState()== true)
             знания = знания + "-C#\n";

        if (cppChkbox.getState()== true)
             знания = знания + "-C++\n";

        summaryTextA.setText("Имя: " + имя + "\n\n"+ знания);

        }
}
```

Вставить разрывы строк!

Внести итог в TextArea

После запуска появляется анкета, и щелчок по кнопке дает итог в TextArea.

Примечание:

Все элементы управления получают «говорящие» имена и вид описания типа. Это немного более затратно при вводе, но, очень помогает при дальнейшем распределении имен и элементов. Иначе в большом проекте с 50–100 элементами управления программист легко может потерять контроль. Использованные элементы управления в предыдущем примере были названы так:

- Заголовок
- nameLab
- nameTextF
- javaChkbox
- csharpChkbox
- cppChkbox
- Отправить Btn
- summaryTextA

Этот пример является делом вкуса, но очень способствует удобочитаемости исходного текста.

12.1.5 Упорядочивание элементов управления с помощью менеджера Layout

Использованное до настоящего момента расположение позволяло программисту располагать элементы управления с точными координатами окна. Это практично тогда, когда окно всегда имеет одинаковый размер, и поэтому должно иметь тот же вид. Однако зачастую более целесообразно, чтобы окно могло адаптировать свои элементы управления под актуальный размер так, чтобы и маленькое окно отображало все элементы управления, как запланировано. Дополнительное преимущество такого метода в высокой мобильности приложения. Для таких требований AWT предоставляет так называемые менеджеры Layout. С помощью таких менеджеров элементы управления располагаются в окне независимо от конкретных позиций. На следующей таблице представлены некоторые из этих менеджеров:

Менеджер расположения	Описание
BorderLayout	Элементы управления располагаются по сторонам света (NORTH, EAST, SOUTH, WEST и CENTER). Размер элементов управления при этом адаптируется автоматически.
FlowLayout	Элементы управления располагаются по строкам. Если в строке больше нет места, они переносятся на следующую строку.
GridLayout	Элементы управления располагаются в виде таблицы.
GridBagLayout	Элементы управления располагаются в виде таблицы. Дополнительно элемент управления может быть описан через несколько ячеек таблицы.
CardLayout	Элементы управления располагаются друг за другом, как в многосторонней карточной игре.

На следующем примере показано влияние выбранного менеджера расположения.

Менеджер BorderLayout (предварительная настройка фрейма)

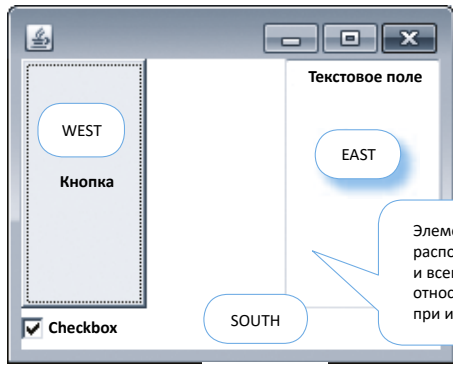
```
class BorderLayoutПример extends Frame {
       private Button кнопка = new Button();
       private TextField текстПоле = new TextField();
       private Checkbox переключатель = new Checkbox();

       public BorderLayoutПример() {
              this.setSize(250, 200);
              this.setLocation(300, 200);

              this.setLayout(new BorderLayout());
              кнопка.setLabel("Кнопка");
              текстполе.setText("Текстовое поле");
              Checkbox.setLabel("Chickbox");
              Checkbox.setState(true);
              this.add(кнопка,BorderLayout WEST);
              this.add(текстПоле,BorderLayout.EAST);
              this.add(переключатель,BorderLayout.SOUTH);

              this.addWindowListener(new окноприемник());
}
```

Включить BorderLayout

Соответствующее расположение элементов управления

```
class окноПриемник extends WindowAdapter {
      @Override
      public void windowClosing(WindowEvent e) {
            System.exit(0);
      }
   }
}
```

После запуска появляется окно с `BorderLayout`.



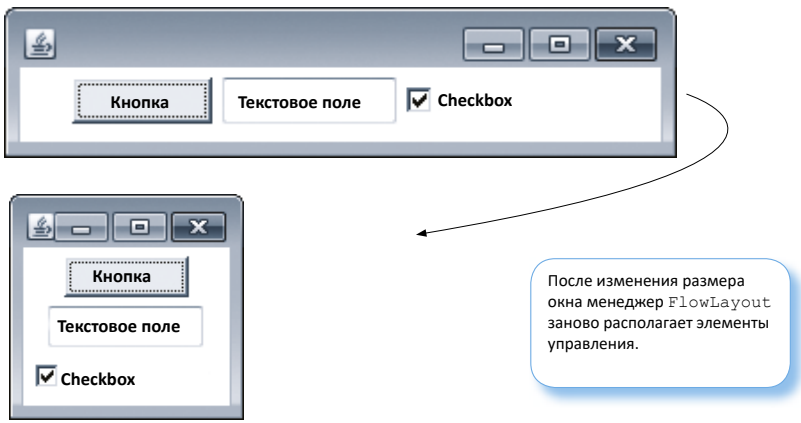
Менеджер FlowLayout

```
:
this.setLayout(new FlowLayout());

кнопка.setLabel("Кнопка");
текстполе.setText("Текстовое поле");
Checkbox.setLabel("Checkbox");
Checkbox.setState(true);

this.add(кнопка);
this.add(текстПоле);
this.add(Checkbox);
:
```

Включить `FlowLayout`

Просто добавить элементы управления

После запуска появляется окно с `FlowLayout`.



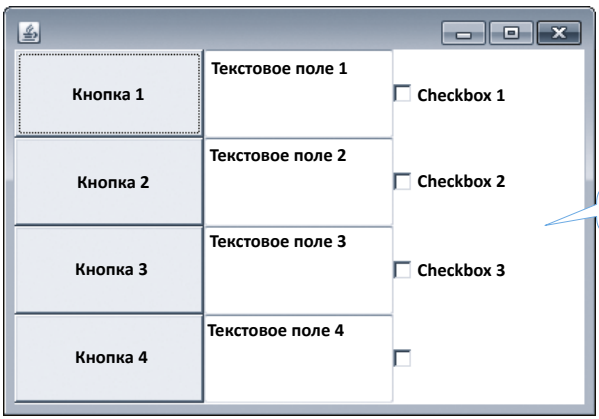
Менеджер GridLayout

```
:
this.setLayout(new GridLayout(4,3));


for (int i=0; i < 4;i++) {
   this.add(new кнопка("Кнопка " + (i+1)));
   this.add(new Текстовое поле("Текстовое поле " + (i+1)));
   this.add(new Checkbox("Checkbox " + (i+1)));
}
:
```

Включить GridLayout и определить 4 строки и 3 столбца

Для удобства добавить 12 элементов управления в цикл

После запуска появляется окно с `GridLayout`.



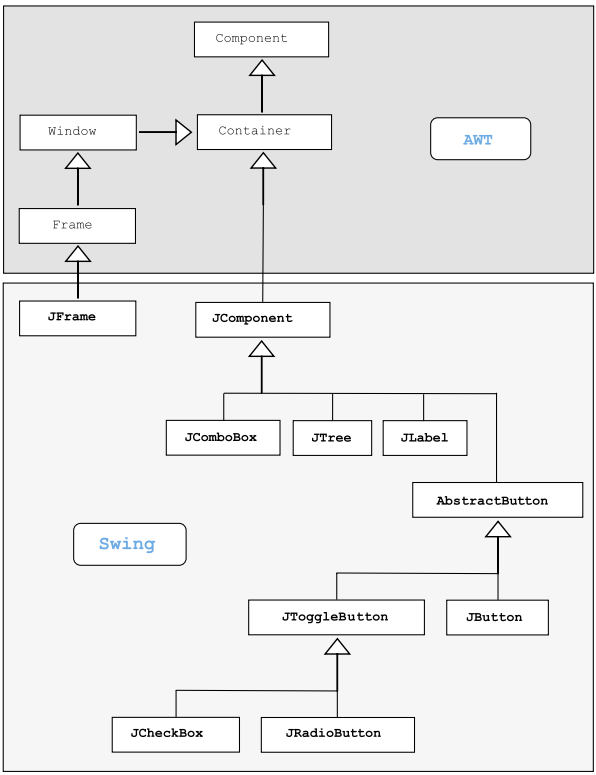
12.2 Элементы управления с помощью классов Swing

12.2.1 Основы классов Swing

В ***Java Development Kit 1.2*** (1998) были введены классы **Swing**, которые предлагают альтернативу AWT. AWT использует встроенные GUI-элементы соответствующей платформы, и поэтому AWT-приложение всегда выглядит как стандартная программа для этой платформы. Классы Swing идут по другому пути. Они предлагают кроссплатформенное представление компонентов. Поэтому компоненты классов Swing называются *легкими компонентами (англ. lightweight components)*. Все компоненты представляются самостоятельно с помощью простых графических команд. Поэтому компоненты полностью поддаются оформлению. AWT компоненты также называются *тяжелыми компонентами* (англ. *heavyweight components*), так как они всегда согласовываются с соответствующим компонентом или функцией платформы.

У истоков классов Swing легкие компоненты приводили к сильной потере производительности. Однако техники постоянно улучшали классы Swing, и теперь уже нет заметной разницы в производительности по сравнению со встроенными компонентами.

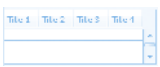
На схеме ниже представлен обзор некоторых важных классов Swing. Структура частично основана на классах AWT, а перед классом Swing, как правило, стоит буква «J».



12.2.2 Элементы управления Swing

Пакет Swing имеет намного больше элементов управления, чем AWT. Такие элементы, как дерево (`JTree`) или таблица (`JTable`) обязательны в современном формировании оболочки. Некоторые важные элементы управления представлены в данной таблице:

Элемент управления	Вид	Описание
`JButton`	Button	Пользователь может щелкнуть по этому элементу управления, и вызвать действие.
`JCheckBox`	CheckBox	Служит для проставления галочки, чтобы, например, выбрать опцию.
`JRadioButton`	RadioButton	Служит для установки опции и обычно используется в группе. В группе всегда может быть установлен только один элемент
`JComboBox`	ComboBox	Поле объединения из выпадающего списка и поля ввода.

JLabel		Служит для надписей. Часто с помощью него создаются надписи полей ввода (TextFields).
JTextField		Служит для ввода и вывода текстовых строк.
JTextArea		Служит для ввода и вывода нескольких текстовых строк. В принципе, TextArea сравним с простым редактором.
JPanel		Вид контейнера или пустого экрана.
JProgressBar		Отображает ход выполнения процесса.
JScrollBar		Исполняет полосу прокрутки.
JSlider		Исполняет ползунковый регулятор.
JTable		Моделирует таблицу для представления данных.
JTree		Служит для представления данных в просмотре дерева.

12.2.3 Использование простых элементов управления Swing

Элементы Swing используются по аналогии с элементами AWT. Если программист выбирает классы Swing, то он должен использовать исключительно элементы Swing, чтобы не провоцировать побочные эффекты или осложнения. Класс окна теперь основан на классе JFrame, и все элементы управления также образуются от классов Swing. Приемник событий и менеджер Layout идентичны таковым в AWT. В отличие от класса Frame AWT, класс JFrame работает с несколькими уровнями. Нижний уровень это так называемый коренной уровень (RootPane) – в принципе, он находится непосредственно в окне и принимает другие уровни. Элементы управления принимаются уровнем содержания (ContentPane), который соответствует клиентской области. Тогда элементы будут не просто добавляться в окно, а должны быть добавлены к уровню содержания. Доступ к этим уровням может осуществляться с помощью соответствующих методов, как get-ContentPane класса JFrame. Также закрытие окна в Swing может производиться несколько проще. Класс JFrame предлагает метод setDefaultCloseOperation, с помощью которого окно реагирует на *событие закрытия.* На следующем примере показана знакомая анкета из главы об элементах управления AWT, только теперь с элементами Swing.

```
class Swingанкета extends JFrame {

        private JButton отправитьКнп = new JButton();
        private JLabel заголовокLab = new JLabel();
        private JLabel имяLab = new JLabel();
        private JTextField имяТекстП = new JTextField();
        private JCheckBox javaChkbox = new JCheckBox();
        private JCheckBox csharpChkbox = new JCheckBox();
        private JCheckBox cppChkbox = new JCheckBox();
        private JTextArea summaryTextA = new JTextArea();
```

Наследовать класс JFrame

Использовать элементы управления Swing

```
public Swingанкета() {

    this.setSize(500, 650);
    this.setLocation(300, 200);
    this.setLayout(null);

    заголовокLab.setFont(new Font("Verdana",Font.BOLD,20) );
    заголовокLab.setForeground(Color.BLUE);
    заголовокLab.setBounds(20, 40, 150, 40);
    заголовокLab.setText("Анкета");

    nameLab.setFont(new Font("Verdana",Font.PLAIN,16) );
    nameLab.setBounds(20, 100, 230, 30);
    nameLab.setText("Пожалуйста, введите имя:");

    nameTextF.setFont(new Font("Verdana",Font.PLAIN,16) );
    nameTextF.setBounds(270, 100, 140, 30);

    javaChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
    javaChkbox.setBounds(20, 150, 180, 30);
    javaChkbox.setText("Знание Java");
```

Метод `setText` заменяет метод `setLabel`.

```
    csharpChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
    csharpChkbox.setBounds(20, 200, 180, 30);
    csharpChkbox.setText("Знание C#");

    cppChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
    cppChkbox.setBounds(20, 250, 180, 30);
    cppChkbox.setText("Знание C++");

    отправитькнп.setFont(new Font("Verdana",Font.PLAIN,16) );
    отправитькнп.setBounds(280, 250, 130, 40);
    отправитькнп.setText("Отправить");

    отправитькнп.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e) {
                        zusammenfassung();
                }
                } );
    summaryTextA.setFont(
              new Font("Verdana",Font.BOLD + Font.ITALIC,20) );
    summaryTextA.setForeground(Color.BLUE);
    summaryTextA.setBounds(20, 350, 360, 250);
    summaryTextA.setEditable(false);
```

Знакомый приемник событий

Доступ к уровню содержания с помощью метода `getContentPane` и добавление элементов управления

```
    this.getContentPane().add(nameLab);
    this.getContentPane().add(nameTextF);
    this.getContentPane().add(javaChkbox);
    this.getContentPane().add(csharpChkbox);
```

```
        this.getContentPane().add(cppChkbox);
        this.getContentPane().add(ОтправитьBtn);
        this.getContentPane().add(summaryTextA);
```

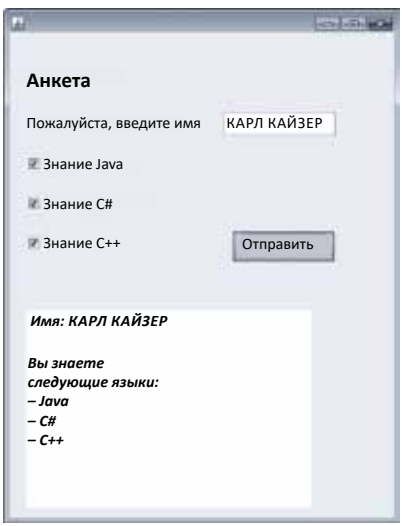
Метод Swing для реагирования на закрыть-событие

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private void итог() {

        String имя = имяТекстП.getText();
        String знания =  "Вы знаете следующие
                                        языки:";
```

Метод isSelected заменяет метод getState.

```
        if (javaChkbox.isSelected()== true)
            знания = знания + "-Java\n";

        if (csharpChkbox.isSelected()== true)
             знания = знания + "-C#\n";

        if (cppChkbox.isSelected()== true)
             знания = знания + "-C++\n";

        summaryTextA.setText("Имя: " + имя + "\n\n"+ знания);

        }
    }
```

После запуска анкета Swing выглядит так:



12.2.4 Look and Feel

Преимущество классов Swing – это легкие компоненты, вид которых можно адаптировать как угодно. Это можно осуществлять даже во время исполнения. Ответственным за ***Look and Feel*** Swing-приложения является `UIManager`. С помощью метода `setLookAndFeel` можно включить множество вариантов. На следующем примере показано использование в Swing-приложении. При этом важно включать обработку исключений, так как могут возникать различные ошибки.

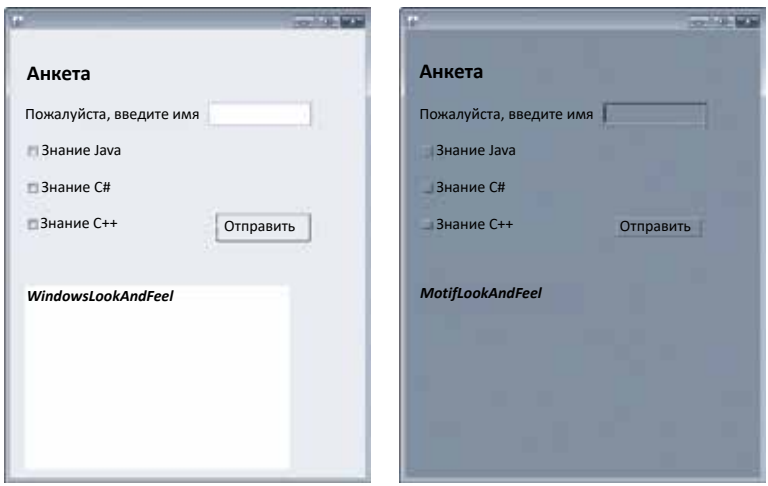
```
try {
     UIManager.setLookAndFeel("com.sun.java.swing.
                          plaf.windows.WindowsLookAndFeel");
SwingUtilities.updateComponentTreeUI (this);
}
catch(Exception e) {

}
```

Предустановленный Windows-*Look and Feel*

Запустить обновления для использования *Look and Feel!*

```
Альтернатива:
UIManager.setLookAndFeel("com.sun.java.swing.
                         plaf.motif.MotifLookAndFeel");
```

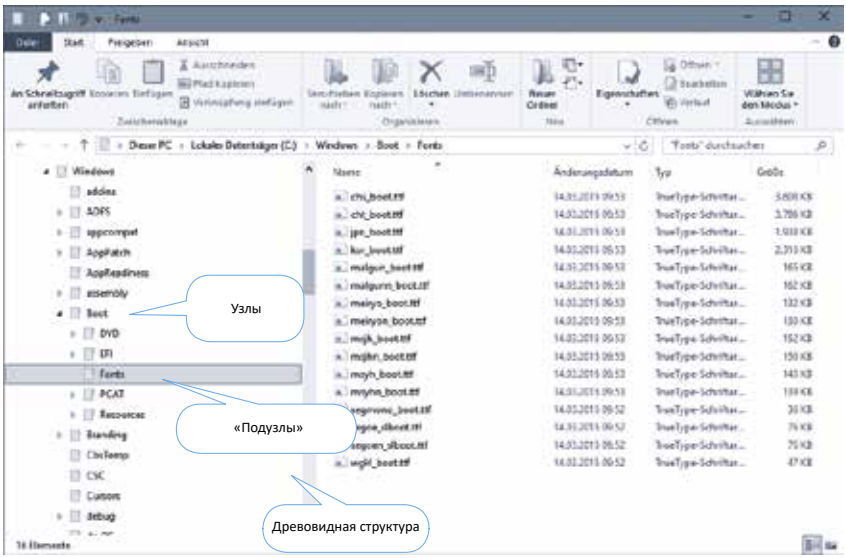
После запуска анкета-приложение может появиться в разном виде *Look and Feel*:



12.3 Более сложные элементы управления классов Swing

12.3.1 Дерево JTree

Дерево (`JTree`) это элемент управления, хорошо знакомый пользователю – например, из такого файлового менеджера *Windows-Explorer* или Linux, как *Nautilus*. На левой половине каталоги представлены в таком виде. Дерево может быть заполнено любым содержанием, не только каталогами и именами файлов. Элементы дерева называются узлами (`TreeNodes`). На следующей схеме представлено типичное дерево в Windows-Explorer:



Древовидная структура инстанцируется как любой другой элемент управления и добавляется уровень содержания.

```
private JTree дерево = new JTree();
дерево.setBounds(20, 20, 100, 150);
this.getContentPane().add(Древовидная структура);
```

После запуска появляется дерево, которое уже имеет некоторые примеры содержания:

12.3.2 Создание узлов в JTree

Для создания узла в `JTree` необходимо просто создать экземпляр объекта класса `DefaultMutableTreeNode`. Тогда конструктору сразу может быть передан текст узла:

```
DefaultMutableTreeNode узел =
                        new DefaultMutableTreeNode("Узел");
```

С помощью метода `add` к узлу можно добавить другой узел (подузел).

```
DefaultMutableTreeNode подузел =
                        new DefaultMutableTreeNode("Подузел");

Узел.add(подузел);
```

На следующем примере показана структура простого дерева `JTree`:

```
class Swing дерево extends JFrame {

        private JTree Древовидная структура;

        public Swingдерево() {
           this.setSize(300, 350);
           this.setLocation(300, 200);
           this.setLayout(null);

           DefaultMutableTreeNode корень =
                 new DefaultMutableTreeNode("Корень");

           Древовидная структура = new JTree(Корень);

           Древовидная структура.setBounds(20, 20, 200, 150);
           DefaultMutableTreeNode подузел =
                   new DefaultMutableTreeNode("Подузел");

           корень.add(подузел);
           DefaultMutableTreeNode подподузел =
                   new DefaultMutableTreeNode("Подподузел");

           подузел.add(подподузел);
           this.getContentPane().add(Древовидная структура);

        }
}
```

Создать ссылку типа `JTree`

Создать корневой узел

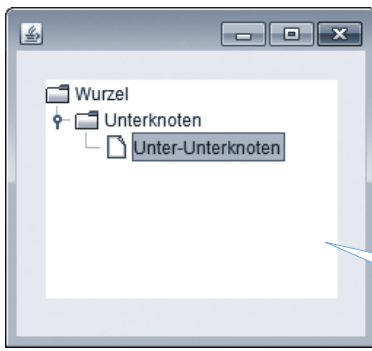
Инстанцировать объект `JTree` и передать корневые узлы

Добавить подузел

Добавить подузел

Добавить дерево уровня содержания

После запуска появляется дерево с узлами:



Узел, не имеющий других подузлов, также называется листом (англ. leaf). Помимо узла он также получает другой символ.

12.3.3 Обзор важных методов JTree

На следующих таблицах показаны методы классов `JTree` и `DefaultMutableTreeNode`, которые могут быть полезны для программирования дерева:

`JTree`:

Метод	Описание
`void collapsePath(TreePath)`	Узел, на который ссылается путь типа `TreePath`, закрывается.
`boolean isCollapsed(TreePath)`	Проверяет, закрыт ли узел, на который ссылается путь типа `TreePath`.
`void expandPath(TreePath)`	Узел, на который ссылается путь типа `TreePath`, открывается (расширяется).
`boolean isExpanded(TreePath)`	Проверяет, открыт ли узел, на который ссылается путь типа `TreePath`.
`TreePath getSelectionPath()`	Дает путь к выбранному узлу.

`DefaultMutableTreeNode`:

Метод	Описание
`void setUserObject(Object)`	Передача строки, к примеру, меняет текст узла.
`void add(DefaultMutableTreeNode)`	Метод `add` добавляет другие подузлы.
`void insert DefaultMutableTreeNode,int)`	Метод `insert` добавляет другие подузлы в определенное место (индекс).
`void remove DefaultMutableTreeNode)`	Удаляет указанный подузел.
`boolean isLeaf()`	Проверяет, является ли узел листом.
`int getChildCount()`	Дает количество подузлов.
`DefaultMutableTreeNode getChildAt(int)`	Доставляет узел к переданному индекс-месту.
`DefaultMutableTreeNode getParent()`	Доставляет узел-родитель.
`DefaultMutableTreeNode getPreviousNode()`	Доставляет предыдущий узел.
`DefaultMutableTreeNode getNextNode()`	Доставляет следующий узел.

12.3.4 Реагирование на события `JTree`

Для события JTree может быть создан приемник типа `TreeSelectionListener`. С помощью него можно реагировать на событие `valueChanged`. На следующем примере показан такой приемник и дополнительное использование некоторых вышеописанных методов.

```
class SwingEreignisДерево extends JFrame {
   private JTree дерево;
   private JTextField выбор = new JTextField();

   public SwingEreignisДерево() {
          this.setSize(500, 650);
          this.setLocation(300, 200);
          this.setLayout(null);
          DefaultMutableTreeNode корень =
                new DefaultMutableTreeNode("Корень");
          дерево = new JTree(корень);
          дерево.setBounds(20, 20, 200, 150);
          выбор.setBounds(20, 180, 400, 30);
```

```
DefaultMutableTreeNode подузел =
      new DefaultMutableTreeNode("подузел ");
wurzel.add(подузел);

DefaultMutableTreeNode подподузел =
      new DefaultMutableTreeNode("подподузел ");
подузел.add(подподузел);
```

Добавление приемника событий

```
дерево.addTreeSe lectionListene

      new TreeSelectionListener() {
```

Создание TreeSelection-Listener

```
Создание TreeSelection-Listener
```

Исполнение метода событий valueChanged

```
@Override
public void valueChanged (TreeSelectionEvnt e)}

   String путь =
       дерево.getSelectionPath().toString();

   выбор.setText(путь);
```

С помощью метода getSelectionPath выбрать путь выбранного узла

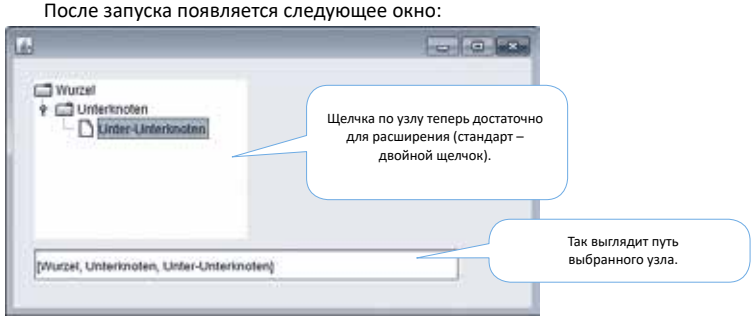
Расширить узел с помощью пути

```
   baumansicht. expandPath(e.getNewLeadSelectionPath());
   }
} );
```

Альтернативно выбранный путь можно выбрать и с помощью метода передаваемого параметра e типа TreeSelectionEvent.

```
this.getContentPane().add(дерево);
this.getContentPane().add(selektion);
    }
}
```

После запуска появляется следующее окно:



12.3.5 Создание таблицы с помощью JTable

С помощью класса `JTable` очень просто создавать в окне таблицы любых размеров. Класс предлагает для этого конструктор, который может перенять массив с данными и заголовки столбца. Самый простой путь при этом – передача данных в строках символов. На следующем примере показана простая таблица, заполненная некоторыми именами и фамилиями.

```
class SwingТаблица extends JFrame {

private JTable таблица;

public SwingТаблица () {

        this.setSize(200, 200);
        this.setLocation(300, 200);
        this.setLayout(null);

        String[][] данные ={
             {"Карл","Кайер"},
             {"Франц","Краутер"},
             {"Лиза","Баум"},
             {"Фред","Юпитер"},
        };

  String[] столбцы = {"Имя", "Фамилия"};

        таблица = new JTable(данные,столбцы);
        таблица.setBounds(20,20,150,65);

        this.getContentPane().add(таблица);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Создать частный атрибут типа `JTable`

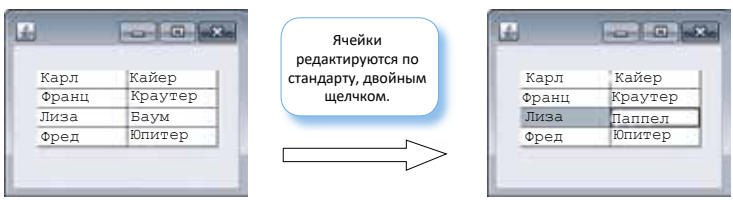
Создать данные как двумерный массив типа `String`

Сохранить имена столбцов в `String`-массиве. **ВНИМАНИЕ:** Имена столбцов не отображаются. Это будет возможно только с применением `JScrollPane` (см. позже).

Инстанцировать объект `JTable` и передать данные и имена столбцов.

Добавить таблицу к уровню содержания

После запуска появляется следующее окно с таблицей:



12.3.6 Обзор важных методов JTable

На следующей таблице представлены некоторые методы `JTable`, которые могут быть очень полезны для программирования:

`JTable`:

Метод	Описание
`void clearSelection()`	Удаляет актуальный выбор ячеек таблицы.
`int [] getSelectedRow()`	Дает индексы выбранных строк.
`int [] getSelectedColumn()`	Дает индексы выбранных столбцов.
`Object getValueAt (int, int)`	Считывает содержание ячейки в заданной строке и столбце (ВНИМАНИЕ: индекс основан на нуле).
`void setValueAt (Object, int, int)`	Пишет содержание ячейки в указанной строке и столбце (ВНИМАНИЕ: индекс основан на нуле).
`int getColumnCount()`	Дает количество столбцов.
`String getColumnName(int)`	Дает имя столбца для указанного индекса.
`int getRowCount()`	Дает количество строк.
`void setBackground(Color)`	Устанавливает цвет фона.
`void setForeground(Color)`	Устанавливает цвет в ячейки.
`void setBorder(Border)`	Устанавливает рамку. Рамка может быть создана с помощью метода класса `BorderFactory`.
`void setGridColor(Color)`	Устанавливает цвет линий между ячейками.

Следующая программа демонстрирует использование некоторых из этих методов:

```
class SwingТаблицыМетоды extends JFrame {
      private JTable таблица;
      public SwingТаблицыМетоды() {

            this.setSize(200, 200);
            this.setLocation(300, 200);
            this.setLayout(null);
            String[][] данные =     {
                  {"Карл","Кайзер"},
                  {"Франц","Краутер"},
                  {"Лиза","Баум"},
                  {"Фред","Юпитер"},
           };

           String[] столбцы = {"Имя", "Фамилия"};

           таблица = new JTable(данные, столбцы);
           таблица.setBounds(20,20,150,65);
```

Установить новое значение ячейки для первой строки и второго столбца (индекс, основанный на нуле)

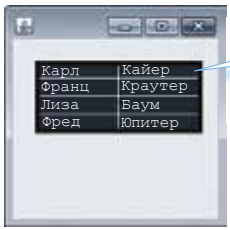
```
           таблица.setValueAt("Кениг", 0, 1);
```

Создать рамку с помощью метода класса BorderFactory типа Border

```
        таблица.setBorder(
             BorderFactory.createLineBorder(Color.BLACK. 4));

        таблица.setBackground(Color.BLUE);

        таблица.setForeground(Color.WHITE);

        таблица.setGridColor(Color.WHITE);

        this.getContentPane().add(таблица);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Установить цвет фона, символов и линий

После запуска появляется следующее окно с таблицей:



Карл Кайзер получил новую фамилию (Кениг) с помощью метода setValueAt.

12.3.7 Реагирование на события JTable

Для события JTable может быть создан приемник типа ListSelectionListener. С помощью него можно реагировать на события valueChanged. На следующем примере показан такой приемник и дополнительное использование некоторых вышеописанных методов. Приемник реагирует при этом на выбор ячейки таблицы. Затем содержание ячейки перезаписывается с новым значением. Старое значение возобновляется после другого выбора.

```
class SwingсобытиеТаблицы extends JFrame {

        private JTable таблица;

        private int строка, столбец;
        private String содержание;
        boolean первыйВыбор = true;

        public SwingСобытиеТаблицы () {
               this.setSize(200, 200);
               this.setLocation(300, 200);
               this.setLayout(null);

               String[][] данные = {
                   {"Карл","Кайзер"},
                   {"Франц","Краутер"},
                   {"Лиза","Баум"},
                   {"Фред","Юпитер"},
              };
```

Дополнительные переменные для защиты содержания ячеек и возобновления после выбора

```
String[] столбцы = {"Имя", "Фамилия" };
таблица = new JTable(данные,столбцы);
таблица.setBounds(20,20,150,65);
```

С помощью метода `getSelectionModel` берется актуальная модель выбора таблицы. Затем этой модели с помощью метода `addListSelectionListener` можно присвоить приемник.

```
таблица.getSelectionModel().addListSelectionListener(
    new ListSelectionListener() {

    @Override

    public void valueChanged( ListSelectionEvent e ) {
       if (первыйВыбор == false)
       таблица.setValueAt(содержание,строка,столбец);
```

Остановить дополнительную переменную на `false` после первого выбора

Возобновление старого содержания уже выбранной ранее ячейки. При первом выборе этого не требуется.

```
       ПервыйВыбор = false;
```

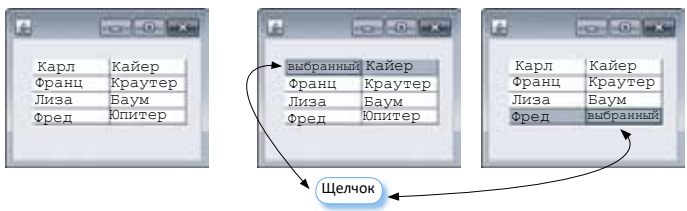
Защита выбранной строки и столбца

```
       строка = таблица.getSelectedRows()[0];

       столбец = таблица.getSelectedColumns()[0];
```

Защита содержания

```
       содержание = таблица.getValueAt(
                    строка ,столбец).toString();

       таблица.setValueAt(
                  "выбранный",строка ,столбец );
    }
  }
  );

  this.getContentPane().add(таблица);
  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```

Содержание ячейки меняется после выбора:



12.3.8 Снабжение элементов управления полосой прокрутки

С помощью класса `JScrollPane` такие элементы, как таблица или дерево, легко могут быть снабжены полосой прокрутки. Для этого элемент необходимо только встроить в объект типа `JScrollPane`. После встраивания таблицы будут отображаться и имена

столбцов. На следующем примере представлена таблица с несколькими вложениями и использованием полосы прокрутки.

```
class SwingScrollтаблица extends JFrame {

      public SwingScrollтаблица () {
            this.setLayout(new BorderLayout());

            String[][] данные =   {
               {"Вена","Австрия"},
               :
               {"Берлин","Германия"},
               {"Амстердам","Нидерланды"}
      };
      String[] столбцы = {"Столица", "Страна"};
      JTable таблица = new JTable(данные,столбцы);
      this.getContentPane().add(new JScrollPane(eineTabelle));

      this.pack();

   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

Включить менеджер Layout, чтобы JScroll- Pane без проблем отображал таблицу.

Создать динамичный объект типа JScrollPane и вшить таблицу (или передать конструктору)

Метод pack отвечает за то, чтобы элементы управления в вышестоящем контейнере (здесь – в окне) корректно отображались в соответствии с расположением.

После запуска окно выглядит так:

Столица	Страна
Вена	Австрия
Варшава	Польша
Стокгольм	Швеция
София	Болгария
Рим	Италия
Прага	Чехия
Париж	Франция
Будапешт	Венгрия
Брюссель	Бельгия
Берн	Швейцария
Берлин	Германия
Амстердам	Нидерланды

Уменьшить окно

Столица	Страна
Стокгольм	Швеция
София	Болгария
Рим	Италия
Прага	Чехия
Париж	Франция
Будапешт	Венгрия
Брюссель	Бельгия

Полоса прокрутки легко прокручивает таблицу.

Примечание: Сортировка столбцов

С помощью объекта типа `TableRowSorter` таблица сразу может производить сортировку содержания по желаемому столбцу с помощью щелчка. Для этого нужно только интегрировать два следующих оператора:

```
TableRowSorter сорт = new TableRowSorter(таблица.getModel());

таблица.setRowSorter(сорт);
```

13 Меню, диалоги и апплеты

13.1 Создание меню с помощью AWT

13.1.1 Создание меню

Меню служат пользователю для выбора разных опций или команд. Они обычно интегрированы в верхней части окна, а также могут быть вызваны как контекстное меню щелчком правой кнопки мыши. С помощью AWT меню создаются в элементах управления. Сначала инстанцируется строка меню – объект типа `MenuBar`. Далее можно создавать отдельное меню и пункты меню (с помощью классов `Menu` и `MenuItem`). Затем строка меню присваивается окну с помощью метода `setMenuBar`. На следующем примере показана структура простого меню.

```
class ОкносМеню extends Frame {

   private MenuBar строкаМеню;
   private MenuItem открытьВложение;
   private MenuItem закрытьВложениеg;
   private MenuItem закончитьВложение;

   public ОкносМеню() {

      this.setSize(200, 300);
      this.setLocation(300, 200);
      this.setLayout(null);

      меню = new Menu("Главное меню");

      открытьВложение = new MenuItem("Открыть файл");
      закрытьВложение = new MenuItem("Сохранить файл");
      закончитьВложение = new MenuItem("Завершить");

      меню.add(открытьВложение);
      меню.add(закрытьВложение);
      меню.add(закончитьВложение);

      СтрокаМеню = new MenuBar();


      СтрокаМеню.add(Menue);


      this.setMenuBar(строкаМеню);


      this.addWindowListener(new окноприемник());
```

Создать атрибуты для создания строки меню и пунктов меню

Инстанцировать объект меню

Инстанцировать три пункта меню

Добавить три пункта в меню

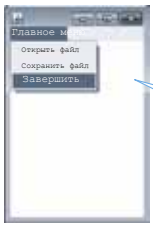
Инстанцировать строку объекта меню

Добавить меню в строку

Добавить строку меню в окно

```
    class окноПриемник extends WindowAdapter {
          @Override
          public void windowClosing(WindowEvent e) {
                System.exit(0);
       }
    }
}
```

После запуска меню готово к использованию:



Меню работает в нормальном режиме, однако еще не реагирует на события.

13.1.2 Реагирование на события меню

Щелчок по пункту меню вызывает событие типа `ActionEvent`. Поэтому приемник события выглядит так же, как у элемента типа `Button`. Этого достаточно, когда приемник описывается. В приемнике затем осуществляется проверка, какой пункт меню был выбран. В следующей программе показано создание приемника и добавление к отдельным пунктам меню – объектам.

```
class менюПриемник implements ActionListener {
```

Создание приемника событий

```
   @Override
   public void actionPerformed(ActionEvent e){
```

С помощью метода `getSource` параметра типа `ActionEvent` определяется источник события (пункт меню).

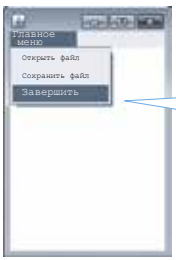
```
   MenuItem какойПунктМеню = (MenuItem) e.getSource();

   if (какойПунктМеню.getLabel().equals("Завершить")){
         System.exit(0);
         }
    }
}
class окноСобытиеМеню extends Frame {

private MenuBar строкаМеню;
      private Menu меню;
      private MenuItem открытьВложение;
      private MenuItem закрытьВложение;
      private MenuItem завершитьВложение;

      public окноСобытиеМеню() {
         this.setSize(200, 300);
         this.setLocation(300, 200);
         this.setLayout(null);
         меню = new Menu("Главное меню");
```

Метод `getLabel` дает описание пункта меню. Так можно соответственно реагировать.

```
        открытьВложение = new MenuItem("Открыть файл");
        закрытьВложение = new MenuItem("Сохранить файл");
        закончитьВложение = new MenuItem("Завершить");

        открытьВложение.addActionListener(new менюПриемник());
        закрытьВложение.addActionListener(new менюПриемник ());
        закончитьВложение.addActionListener(new менюПриемник ());

        меню.add(открытьВложение);
        меню.add(закрытьВложение);
        меню.add(завершитьВложение);
        строкаМеню = new MenuBar();
        строкаМеню.add(меню);
        this.setMenuBar(строкаМеню);
        this.addWindowListener(new окноПриемник());
    }
    class окноПриемник extends WindowAdapter {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        }
    }
```

Добавить приемник события

После запуска меню реагирует на события:



Кнопка «Завершить» закрывает приложение.

13.1.3 Создание контекстного меню

Создание контекстного меню так же просто, как создание меню. Вместо класса `MenuBar` используется класс `PopupMenu`. Объект класса добавляется в окно, как простой элемент управления и вызывается с помощью метода `show`. Как правило, контекстное меню запускается щелчком правой кнопки мыши. Поэтому имеет смысл создать для этого соответствующий приемник. В следующих программах создается простое контекстное меню и отображается с помощью щелчка правой кнопки мыши.

```
class окноКонтекстноеМеню extends Frame {

        private PopupMenu контекстноеМеню;
        private MenuItem открытьВложение;
        private MenuItem закрытьВложение;
        private MenuItem завершитьВложение;

        public окноКонтекстноеМеню () {
             this.setSize(200, 300);
             this.setLocation(300, 200);
             this.setLayout(null);
```

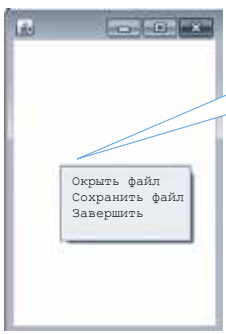
```
        контекстноеМеню = new PopupMenu("Главное меню");

        открытьВложение = new MenuItem("Открыть файл");
        закрытьВложение = new MenuItem("Сохранить файл");
        закончитьВложение = new MenuItem("Завершить");

        контекстноеМеню.add(открытьВложение);
        контекстноеМеню.add(закрытьВложение);
        контекстноеМеню.add(завершитьВложение);

        this.add(контекстноеМеню);

        this.addMouseListener(new мышьПриемник());
        this.addWindowListener(new окноПриемник());
    }

    class мышьПриемник extends MouseAdapter {
        @Override
        public void mouseClicked(MouseEvent e){


        if (e.getButton()==MouseEvent.BUTTO3)
         контекстноеМеню.show(e.getComponent(), e.getX(), e.getY());
    }
}

    class окноПриемник extends WindowAdapter {. . . } //как было ранее

    }
```

Добавить приемник для действия мыши

Метод `e.getButton` дает значение кнопке мыши, которое может идентифицироваться постоянными из класса `MouseEvent` (правая кнопка мыши соответствует `BUTTON3`).

Метод `show` отображает контекстное меню. С помощью `getComponent` определяется вышестоящий компонент (здесь – окно). Координаты мыши вызываются с помощью `getX` и `getY`.

После запуска контекстное меню отображается по щелчку правой кнопки мыши:



Контекстное меню отображается на месте щелчка мыши.

13.1.4 Создание Меню с помощью классов Swing

Объединение меню с классами Swing проходит почти идентично с вариантом AWT. Только вместо классов `Menu`, `MenuItem` и других классов меню теперь необходимо использовать классы Swing `JMenu`, `JMenuItem` и другие классы меню Swing. Следующая программа показывает интеграцию меню, контекстного меню и приемника с классами Swing.

```
class окноСМенюSwing extends JFrame {
   private JMenuBar строкаМеню;
   private JMenu меню;
   private JMenuItem открытьВложение;
   private JMenuItem закрытьВложение;
   private JMenuItem завершитьВложение;
   private JPopupMenu контекстноеМеню;
   public окноСМенюSwing () {

      this.setSize(200, 300);
      this.setLocation(300, 200);
      this.setLayout(null);

      меню = new JMenu("главное меню");

      открытьВложение = new JMenuItem("Открыть файл");
      закрытьВложение = new JMenuItem("Сохранить файл");
      закончитьВложение = new JMenuItem("Закончить");


      меню.add(открытьВложение);
      меню.add(закрытьВложение);
      меню.add(завершитьВложение);

      строкаМеню = new JMenuBar()
      строкаМеню.add(меню);


      this.setJMenuBar(строкаМеню)


      контекстноеМеню = new JPopupMenu("Главное меню")

      контекстноеМеню.add(new JMenuItem("Открыть файл"));
      контекстноеМеню.add(new JMenuItem("Сохранить файл"));
      контекстноеМеню.add(new JMenuItem("Завершить"));



      this.addMouseListener(new мышьПриемник());
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  class мышьПриемник extends MouseAdapter { //как было ранее
  }
}
```

Наследовать класс JFrame

Использовать классы Swing!

Инстанцировать меню Swing

Инстанцировать элементы меню Swing

Инстанцировать меню Swing

ВНИМАНИЕ: В меню Swing использовать метод setJMenuBar.

Инстанцировать строку меню и добавить в меню

Инстанцировать всплывающее меню

Добавить примеры элементов во всплывающее меню

После запуска отображается и меню, и контекстное меню (при щелчке правой кнопкой мыши):

13.2 Диалоги

13.2.1 Использование стандартных диалогов

Большинство приложений имеет диалоги для сохранения или загрузки данных. Эти диалоги обычно всегда выглядят одинаково. Это целесообразно, так как пользователям не придется изучать использование диалогов каждой новой программы заново. Поэтому Java предлагает предустановленные стандартные диалоги. Эти диалоги могут быть интегрированы и при необходимости открыты в любом приложении. Для AWT существует только один диалог (`FileDialog`), соответствующий классическому диалогу «*Открыть файл*». Классы Swing предлагают другие возможности, например, диалог выбора цвета или диалог опций (похожий на панель сообщений). В следующих программах показано, как эти диалоги могут быть использованы:

Вариант 1: AWT-диалог

```
class стандартныеДиалоги extends Frame{

   private FileDialog файлдиалог;

   public стандартныеДиалоги () {

      файлДиалог = new FileDialog(this,
                        "Открыть файл",FileDialog.LOAD);

      файлДиалог.setDirectory("c:\\Java""));
      файлДиалог.setVisible(true);

      this.addWindowListener(new окноПриемник());
   }
   class окноПриемник extends WindowAdapter { //как было ранее }
   }
}
```

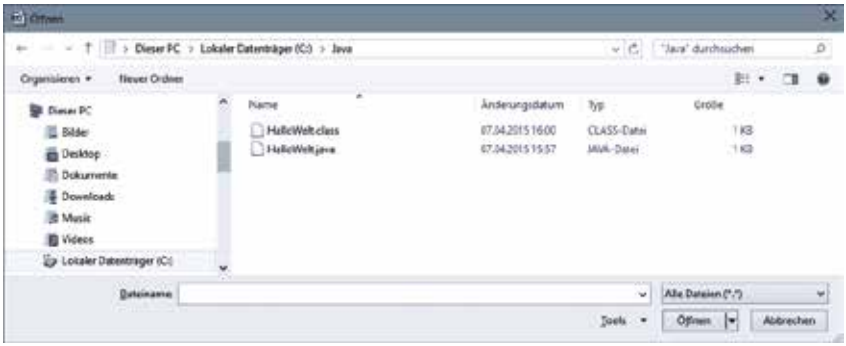
Создать ссылку типа FileDialog

Инстанцировать объект типа FileDialog и указать вышестоящий контейнер (здесь – окно), название и вид (FileDialog.LOAD → открыть файл)

Указать стартовый каталог

Отобразить диалог

После запуска сразу отображается диалог (здесь стандартный Windows-диалог):



Примечание:

С помощью методов `getDirectory` и `getFile` можно вызвать выбранный каталог и файл пользователя.

Вариант 2: Диалоги классов Swing

```
class СтандартныеДиалогиSwing extends JFrame{

    private JFileChooser файлДиалог;

    public СтандартныеДиалогиSwing () {

    if      (       JOptionPane.showConfirmDialog(
                    this,
                    "Сделайте выбор:", "Выбор",
                    JOptionPane.YES_NO_CANCEL_OPTION)

                     == JOptionPane.YES_OPTION
    )       {
                    JOptionPane.showMessageDialog(
                    this,
                    "Вы выбрали ДА");

    }
```

Создать ссылку типа `JFileChooser`. Другие диалоги реализуются с помощью статических методов.

Статический метод `showConfirmDialog` открывает диалог, который может быть снабжен различными опциями (здесь отображаются три опции). Для этого используются постоянные из класса `JOptionPane` (здесь `YES_NO_CANCEL_OPTION`). Метод возвращает значение, которое также можно вызвать с помощью постоянных.

Статический метод `showMessageDialog` открывает простой диалог с сообщением и кнопкой OK. Это можно сравнить с панелью сообщений в других языках программирования.

Статический метод `showInputDialog` открывает диалог ввода. Ввод пользователя возвращается как цепь символов.

```
String каталог =
        JOptionPane.showInputDialog(
        this,
        "Пожалуйста, введите каталог!");
```

Инстанцируется объект класса `JFileChooser`. Выбранный каталог передается конструктору.

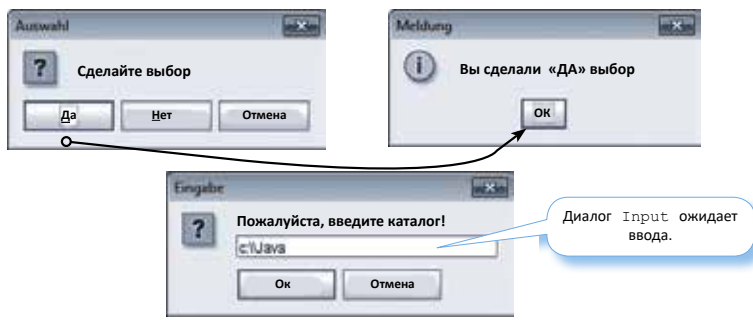
```
dateDialog = new JFileChooser(verzeichnis);
файлдиалог.showOpenDialog(this)
```

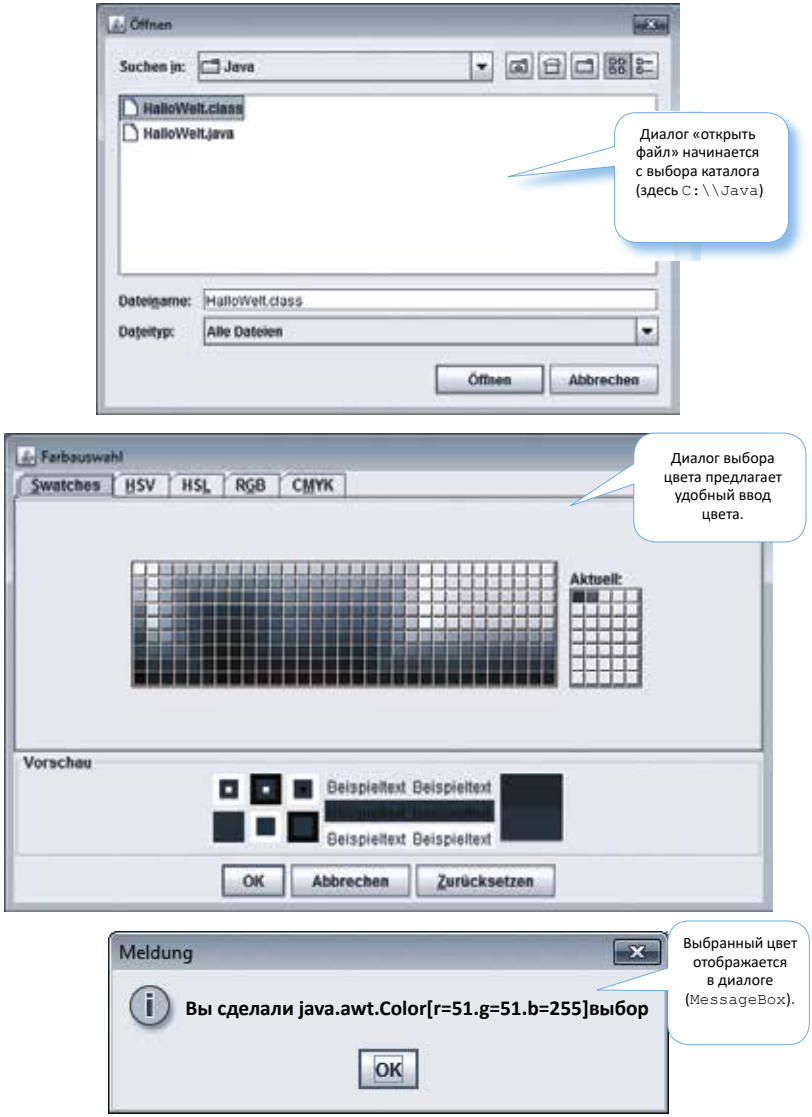
С помощью метода `showOpenDialog` диалог «Открыть файл» открывается. Аналогично можно открыть диалог «Сохранить файл» `showSaveDialog`.

Статический метод showDialog открывает диалог выбора цвета. Выбранный цвет возвращается как объект `Color`.

```
"выбор цвета", Color.BLUE);
                this,
                "Выбор цвета", Color. BLUE);

    JOptionPane.showMessageDialog(this,
                "Вы сделали " + цвет.toString() + выбор");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

После запуска все диалоги отображаются друг за другом:





13.2.2 Создание собственных диалогов

Создание собственных диалогов можно сравнить с созданием класса окна. Однако класс Диалог производится не от класса `Frame` или `JFrame`, а от класса `Dialog` или `JDialog`. Потом в диалог, как обычно, можно добавить элементы управления. Диалоги могут быть *модальными* и *не модальными*. *Модальный* диалог ожидает до тех пор, пока пользователь не закончит диалог. Не модальный диалог может продолжать быть активным на фоне (типичный пример – диалог поиска). В следующих программах показано, как можно использовать собственные диалоги:

Вариант 1: Собственный AWT-диалог

```
class МойAWTдиалог extends Dialog {

private Button кнопка;
private TextField вводтекста;
private String сохранение;

public МойAWTдиалог (     Frame родительокно,
                          String название, boolean модальный) {
        super (родительОкно, название, модальный);

        this.setLayout(null);
        this.setSize(200, 200);

        кнопка = new Button("Пожалуйста,нажмите");
        кнопка.setBounds(50, 50, 100, 30);
        вводтекста = new TextField();
        вводтекста.setBounds(50, 100, 100, 30);

        кнопка.addActionListener( new ActionListener()
              {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                                сохранение = вводтекста.getText();
                                setVisible(false);
                    }
              }
              );

          add(кнопка);
          add(вводтекста);
    }

    public String getSicher
          return сохранение;
    }
}

    class СобственныеAWTдиалоги extends Frame

    МойAWTдиалог диалог;

    public СобственныеAWTдиалоги (){

    диалог = new МойAWTдиалог (this,"Собственный диалог",true);
    диалог.setVisible(true);
```

Класс диалога наследует класс диалог.

Создать элементы управления!

ВАЖНО: вызвать конструктор базового класса и вышестоящий контейнер (здесь – окно), указать название и логический тип (`true` или `false`). Если передается значение `true`, то открывается модальный диалог (в противном случае – не модальный).

При щелчке по `Кнопке` текст ввода сохраняется в атрибуте. Далее включается *невидимый* диалог.

Этот метод служит для возвращения защищенного текста из поля ввода

Создать ссылку на новый тип диалога

Инстанцировать объект класса Диалог. Должен открыться модальный диалог.

Отобразить диалог

После окончания диалога (щелчок по Button) введенный текст вызывается с помощью метода getSicherung.

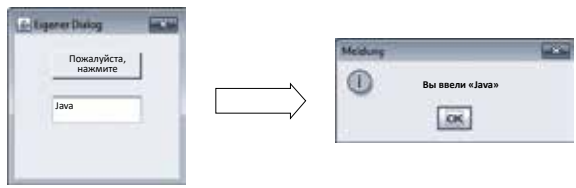
```
        if (einDialog.getSicherung () .equals("Java"))
                JOptionPane.showMessageDialog(this,
                                "Вы ввели 'Java'");
```

Объект Диалог удаляется.

Если пользователь ввел «Java», то запускается Message Box.

```
        диалог.dispose();
        this.addWindowListener(new окноприемник());
    }
    class окноПриемник extends WindowAdapter { //как было ранее }
    }
}
```

После запуска появляется следующий диалог:



Вариант 2: Собственный диалог Swing

Вариант Swing почти идентичен с вариантом AWT. Вместо классов AWT используются классы Swing. Поэтому приведены только соответствующие строки с исходным кодом:

```
class МойSwingДиалог extends JDialog {

   private JButton кнопка;
   private JTextField вводтекста;
   private String сохранение;

   public МойSwingДиалог ( JFrame родительОкно,
                           String название, boolean модальный) {

         super (родительОкно, название, модальный);
         :
         :
   }
   :
   :
}
class СобственныйSwingДиалог extends JFrame {
      МойSwingДиалог диалог;

      public СобственныйSwingДиалог() {

            диалог = new МойSwingДиалог (this,
                                    "Собственный диалог",true);
     :
```

```
        :
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              }
}
```

После запуска появляется вариант Swing:

13.3 Создание апплетов

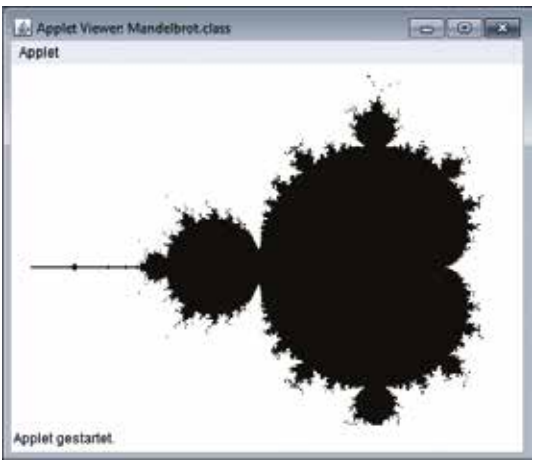
13.3.1 Основы апплетов

Большинство пользователей Интернета знают апплеты, хотя, возможно, и не под этим названием. Апплеты – это программы Java, которые исполняются в браузере. При этом с сервера скачиваются HTML-страница и `Bytecode` апплета и отображаются в браузере. Тем не менее, в браузере должен быть установлен Java-*Plug-in*. Апплеты представляют собой мощный инструмент для веб-программирования. Однако в целях безопасности существуют следующие важные ограничения:

► Апплеты проверяются браузером перед исполнением на угрозы безопасности. Поэтому загрузка апплета длится немного дольше, чем обычной программы Java.

► У апплета не может быть доступа к локальным носителям информации (например, к жесткому диску).

► Доступ к другим ресурсам в сети блокируется. Апплет может иметь доступ только к серверу, с которого он был загружен (чтобы, например, получить другие данные).



Апплет представляет так называемое множество **Мандельброта.** Это интересное изображение возникает из-за понятия *рекурсия*, и поэтому может находить большое применение в графических методах Java.

Примечание:

Вышеприведенный апплет *Мандельброта* был запущен ***NetBeans*** с помощью *Applet Viewer.* Это упрощает процесс работы апплета. Запуск через HTML-страницу будет представлен позже.

13.3.2 Класс апплетов

Программирование апплетов основано на классах `Applet` из пакета `java.applet`. В этом классе создано несколько методов, которые следует перезаписать. Эти методы определяют работу апплетов. Так, внутри методов могут быть использованы все аспекты программирования Java, если только они не противоречат правилам безопасности. Например, попытка доступа к файлу вызовет следующее сообщение об ошибке:

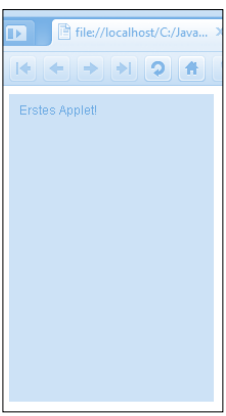
```
File файл = new File("c:/temp/text.txt");
файл.delete();
```

Важные методы класса Applet, которые следует перезаписать:

Метод	Описание
`public void init()`	Этот метод вызывается, когда апплет загружается. Он служит для инициализации атрибутов или элементов апплетов.
`public void start()`	Этот метод вызывается после метода `init`, когда апплет загружен. Также он вызывается всегда, когда пользователь снова переходит на веб-страницу.
`public void stop()`	Этот метод вызывается, когда пользователь переходит на другую веб-страницу.
`public void destroy()`	Этот метод вызывается, когда браузер удаляет апплет из памяти.
`public void paint(Graphics)`	Этот метод уже известен из программирования графики. Он ведет себя аналогично и вызывается, когда апплет необходимо заново представить в графике.

На следующем примере представлен очень простой апплет, который выводит только текст в области апплета:

```
import java.awt.*;
import java.applet.*;

public class первыйапплет
             extends апплет{

public void init() {
    this.setSize(100,100);
    this.setBackground(Color.LIGHT_GRAY);
}

public void paint(Graphics g) {

    g.drawString("первый апплет!",10,20);
  }
}
```



13.3.3 Запуск апплетов

Апплеты можно запустить либо напрямую из среды разработки, либо через HTML-страницу. Первый вариант очень удобен, прежде всего, на этапе разработки апплета. При этом вызывается *Applet Viewer* ***JDK***, который отображает апплет в собственном окне:

Также переведенный апплет можно запустить через HTML-страницу:

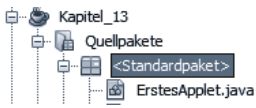
```
<html>
<body>

<applet code="первыйапплет.class" width="200" height="300">

</applet>


</body>
</html>
```

С помощью операции code апплет задается в форме байткода (.class).

Необходимо задать высоту и ширину апплета.

ВНИМАНИЕ:

Сначала апплеты должны разрабатываться не в собственном, а в стандартном пакете. Иначе необходимо проведение некоторых работ по синхронизации и расстановке тегов в апплете. Поэтому для первых испытаний достаточно разработки в стандартном пакете.



13.3.4 Элементы управления в апплетах

Использование элементов управления в апплетах происходит так же, как в программировании с AWT или классами Swing. С помощью метода `add` в апплет добавляются элементы. Приемник событий просто перенимается. На следующем примере представлена знакомая анкета, только в виде апплета:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class АнкетаАпплет extends Апплет{
   private Button отправитькнп = new Button();
   private Label заголовокLab = new Label();
   private Label nameLab = new Label();
   private TextField nameTextF = new TextField();
   private Checkbox javaChkbox = new Checkbox();
   private Checkbox csharpChkbox = new Checkbox();
   private Checkbox cppChkbox = new Checkbox();
   private TextArea summaryTextA = new TextArea();
```

Создать элементы управления

```
public void init() {
      this.setSize(500,650);
      this.setLayout(null);

      ueberschriftLab.setFont(new Font("Verdana",Font.BOLD,20) );
      ueberschriftLab.setForeground(Color.BLUE);
      ueberschriftLab.setBounds(20, 40, 120, 40);
      ueberschriftLab.setText("Анкета");

      nameLab.setFontnew Font("Verdana",Font.PLAIN,16) );
      nameLab.setBounds(20, 100, 200, 30);
      nameLab.setText("Пожалуйста, введите имя:");

      nameTextF.setFont(new Font("Verdana",Font.PLAIN,16) );
      nameTextF.setBounds(230, 100, 150, 30);

      javaChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
      javaChkbox.setBounds(20, 150, 150, 30);
      javaChkbox.setLabel("Знание Java");

      csharpChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
      csharpChkbox.setBounds(20, 200, 150, 30);
      csharpChkbox.setLabel("Знание C#");

      cppChkbox.setFont(new Font("Verdana",Font.PLAIN,16) );
      cppChkbox.setBounds(20, 250, 150, 30);
      cppChkbox.setLabel("Знание C++");

      absendenBtn.setFont(new Font("Verdana",Font.PLAIN,16) );
      absendenBtn.setBounds(280, 250, 100, 40);
      absendenBtn.setLabel("Отправить");

      absendenBtn.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent e) {
         }                          итог();
      }
      );

      summaryTextA.setFont
          (new Font("Verdana",Font.BOLD + Font.ITALIC,20) );
      summaryTextA.setForeground(Color.BLUE);
      summaryTextA.setBounds(20, 350, 360, 250);
      summaryTextA.setEditable(false);

      add(ueberschriftLab);
      add(nameLab);
      add(nameTextF);
      add(javaChkbox);
      add(csharpChkbox);
      add(cppChkbox);
      add(absendenBtn);
      add(summaryTextA);
   }
   private void итог() {

      String name = nameTextF.getText();
         String kenntnisse = "Вы знаете
                             следующие языки:";
```

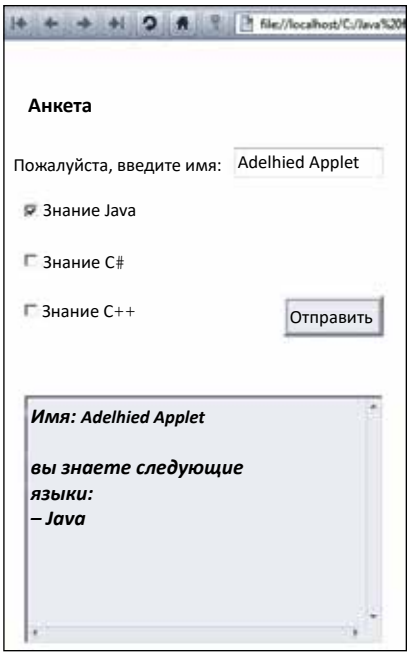
В методе `init` все элементы инициализируются.

Применяется знакомый приемник.

Добавляются элементы.

```
        if (javaChkbox.getState()==true)
        знания = знания + "-Java\n";

        if (csharpChkbox.getState()==true) знания = знания + "-C#\n";

        if (cppChkbox.getState()==true)
        знания = знания + "-C++\n";

        summaryTextA.setText("имя: " + имя + "\n\n"+ знания);
    }
}
```

После запуска через браузер появляется апплет-анкета:



Знакомая анкета теперь представлена в апплете и может быть интегрирована в веб-страницу. Для этого веб-страница и байт-код апплета должны находиться на одном сервере.

13.1.1 Создание апплетов с помощью классов Swing

Так же, как альтернативные классы AWT-`Frame`, классы Swing тоже предоставляют альтернативу классам Applet. Апплет Swing основан на классе **`JApplet`**. Объединение элементов управления и приемников событий производится в обычном режиме. Однако при применении классов Swing следует учитывать следующие аспекты:

- Апплет наследует класс `JApplet`:

```
public class ПервыйАпплет extends JApplet { . . . }
```

- Добавление элементов должно происходить через уровень содержания:

```
this.getContentPane().add(элемент);
```

- Графическое изображение апплета в клиентской области должно производиться не напрямую, а через экран (`JPanel`). При этом следует учитывать, что метод `paint` элементов Swing называется `paintComponent`, и сначала всегда должен вызываться метод базового класса. На следующем примере показан простой апплет Swing, изображенный в клиентской области:

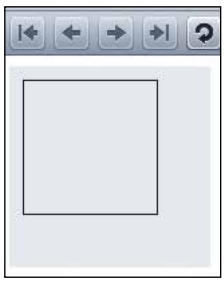
```
import javax.swing.JApplet;
import javax.swing.*;
import java.awt.*;

   class экран extends JPanel {
           public void paintComponent(Graphics g) {
                  super.paintComponent(g);
                  g.drawRect(10, 10, 100, 100);
           }
   }
   public class первыйаплет extends JApplet {

              private экран КлиентскаяОбласть

              public void init() {
                 КлиентскаяОбласть = new экран();
                 this.getContentPane().add(клиентскаяОбласть);
              }

   }
```

Создать класс экрана и перезаписать метод `paintComponent`

Апплет наследует `JApplet`!

Добавление через уровень содержания

После запуска через браузер появляется апплет Swing:

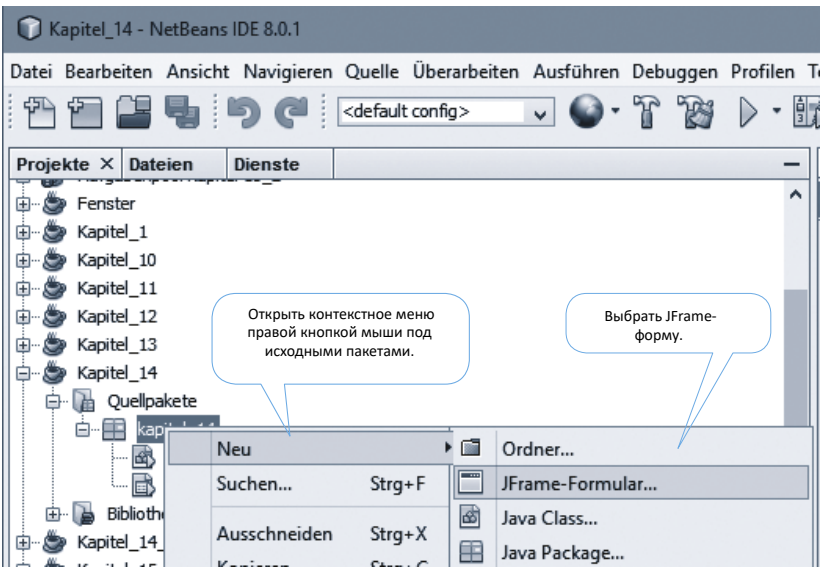


14 GUI-конструктор NetBeans

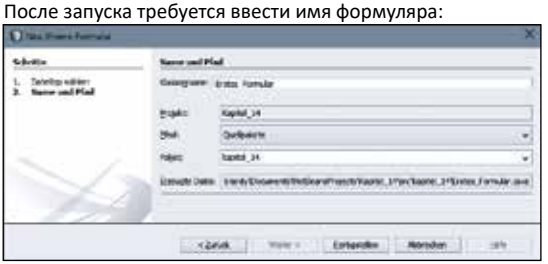
14.1 GUI-конструктор NetBeans

14.1.1 Создание формы JFrame

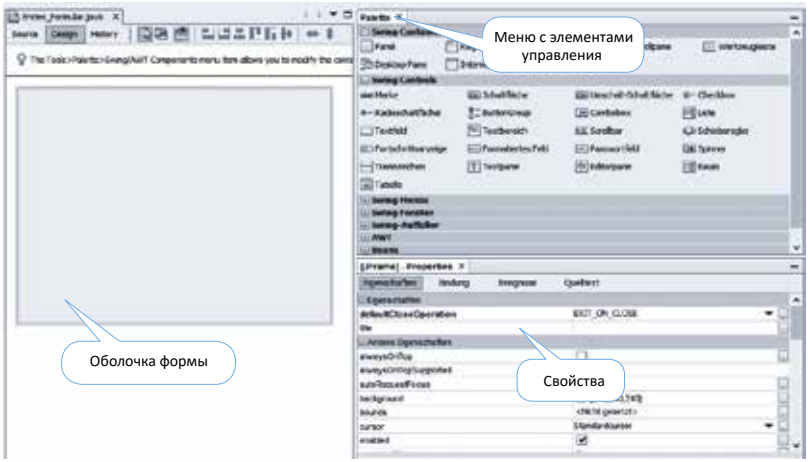
Предыдущие приложения представляли собой микс из консольной программы и GUI-приложения. Программирование приложения было полностью текстового типа. Теперь когда будет применяться GUI-конструктор среды разработки ***NetBeans***. ***GUI-конструктор*** – это инструмент, позволяющий делать графическую разработку приложения. Это, конечно, не означает, что больше не требуется написание программного кода. Тем не менее, ***GUI- конструктор*** существенно облегчает разработку. Для использования GUI-конструктора в проект Java интегрируется специальная форма **JFrame**:



После запуска требуется ввести имя формуляра:



По завершении Netbeans выдает следующую оболочку:



С помощью кнопки можно переключать визуальную конфигурацию и обзор исходного кода:

Щелчок по «Source»

```
package глава_14;

public class Первая_форма extends javax.swing.JFrame {
       public Первая_форма () {
           initComponents();
}
```

Данная аннотация отменяет предупреждения компилятора.

Метод для инициализации компонентов.

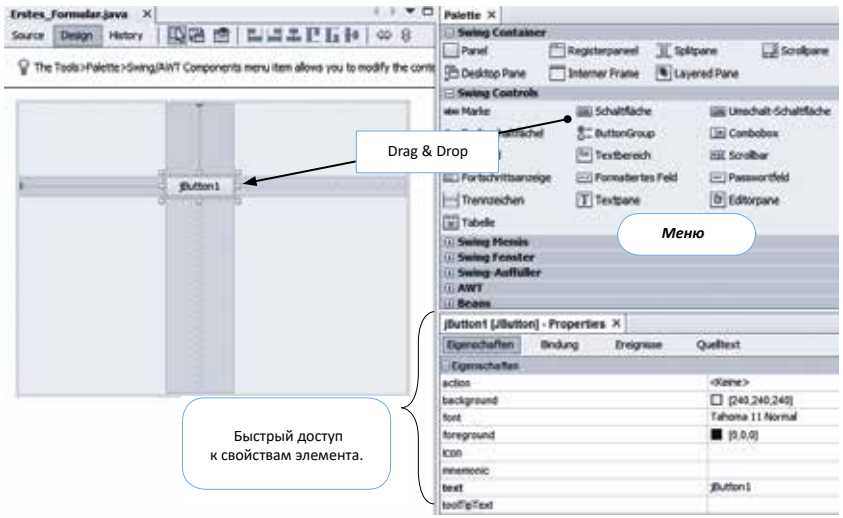
```
@ SuppressWarnings("unchecked")
private void initComponents() {
        setDefaultCloseOperation(javax.swing.WindowConstants.
        EXIT_ON_CLOSE);
        javax.swing.GroupLayout layout =
                new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        :
        :
        pack();
}
public static void main(String args[]) {
   :
   :
   java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
```

```
                new Первая_форма ().setVisible(true);
                }
        });
        :
        :
    }
}
```

Запуск формы.

14.1.2 Добавление элементов управления

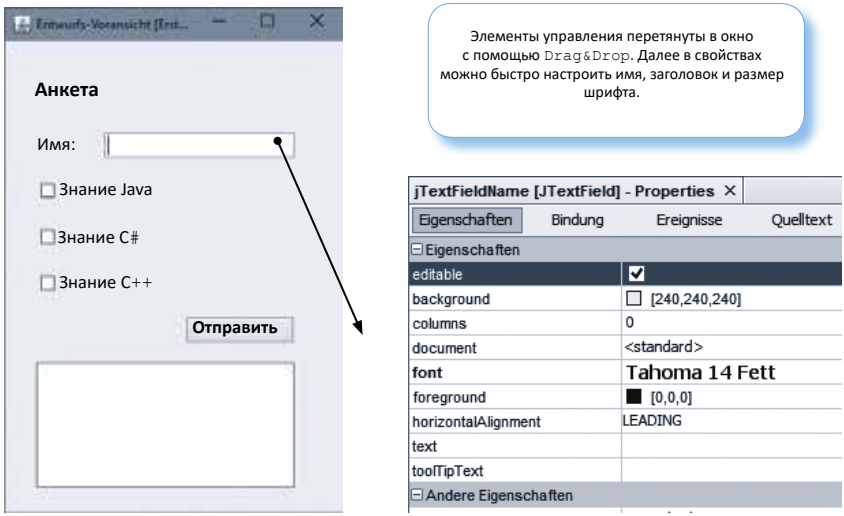
Конфигурация формы может производиться с помощью ***меню***. Меню – это составляющая *GUI-конструктора* для графической конфигурации окна. Для этого желаемые элементы управления просто перетягиваются с помощью `Drag&Drop` из меню в формы. Такую конфигурацию также называют Rapid Application Development **RAD**, так как программирование или визуальная конфигурация может происходить очень быстро и интуитивно. На следующей схеме представлено использование меню.



GUI-конструктор автоматически генерирует соответствующий исходный код для элемента управления в классе и в методе `initComponents`:

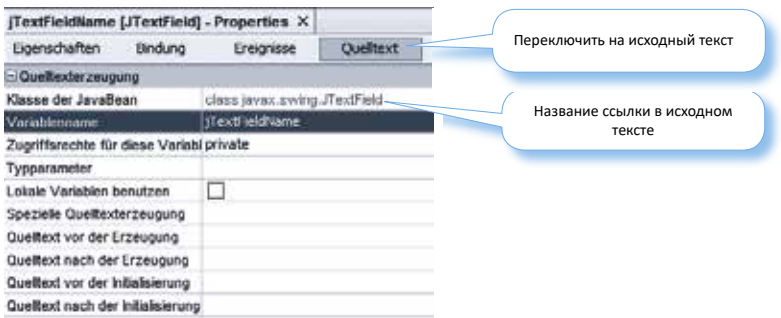
```
private javax.swing.JButton jButton1;
:
private void initComponents() {
        :
        jButton1 = new javax.swing.JButton();
        :
}
```

С помощью ***меню*** и свойств формы, как известно из предыдущих примеров, очень быстро сконфигурировать:



ВНИМАНИЕ:

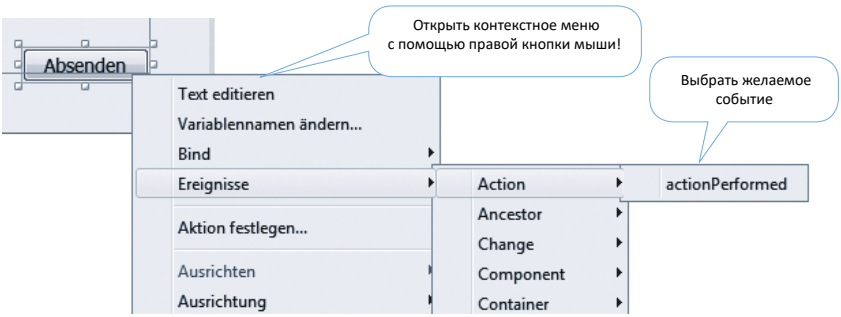
Имя элемента управления в свойствах не следует путать с названием соответствующей ссылки в исходном тексте. Если необходимо изменить имя ссылки в *исходном тексте*, то следует переключить в свойствах на вид *Исходный текст*:



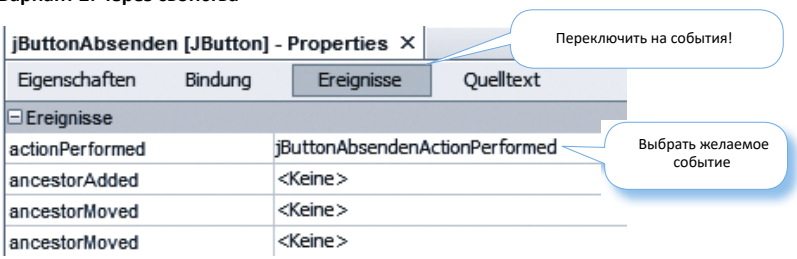
14.1.3 Реагирование на события

Создание приемников событий можно очень легко осуществлять с помощью *GUI конструктора*. Желаемое событие выбирается либо через контекстное меню, либо через свойства соответствующего элемента управления. Далее *GUI-конструктор* автоматически создает приемник события. На следующем примере показаны оба варианта и сгенерированный с помощью *GUI-конструктора* исходный код.

Вариант 1: через контекстное меню



Вариант 2: через свойства



Далее генерируется следующий исходный код:

В методе `initComponents` этот приемник добавляется автоматически. Далее для обработки события вызывается метод `jButtonAbsendenActionPerformed`. В этом методе добавляется исходный текст для обработки события.

```
jButtonAbsenden.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        jButtonAbsendenActionPerformed(evt);
    }
});
```

Автоматически сгенерированный исходный код `jButtonAbsendenActionPerformed` служит для собственного программирования события.

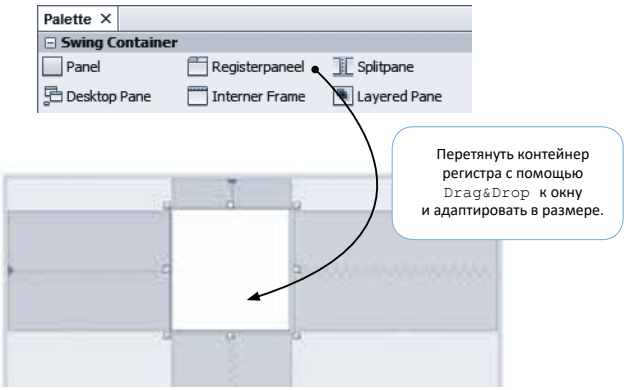
```
    private void jButtonAbsendenActionPerformed(java.awt.event.
                                                ActionEvent evt) {
    // TODO add your handling code here:
    }
```

14.2 Интеграция более сложных элементов управления

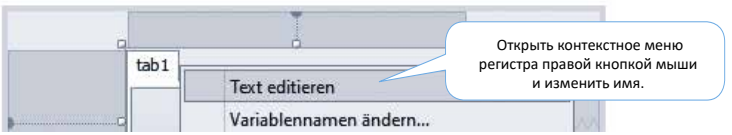
Кроме элементов управления меню также предлагает встраивание других контейнеров, которые затем, в свою очередь, принимают элементы. Далее представим три таких контейнера.

14.2.1 Контейнер регистра (панель регистра)

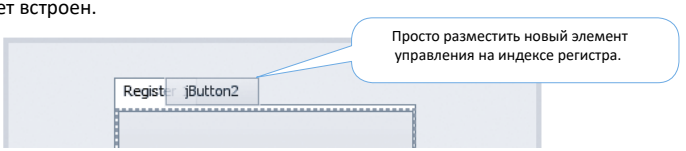
Этот контейнер предлагает любое количество регистров, которые соответственно могут быть заполнены любыми элементами управления. Так можно представить много функций приложения в хорошо структурированном виде.



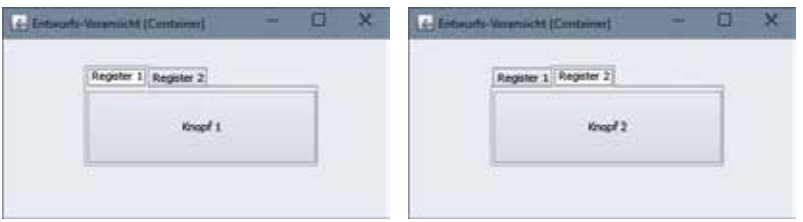
Далее элементы управления можно перетянуть в контейнер регистра. С первым элементом управления (в следующем примере `Button`) отображается индекс регистра. Имя индекса регистра можно изменить через контекстное меню.



Для создания других регистров необходимо просто перетянуть новый элемент управления к индексу регистра. Так автоматически откроется новый регистр, и элемент управления будет встроен.

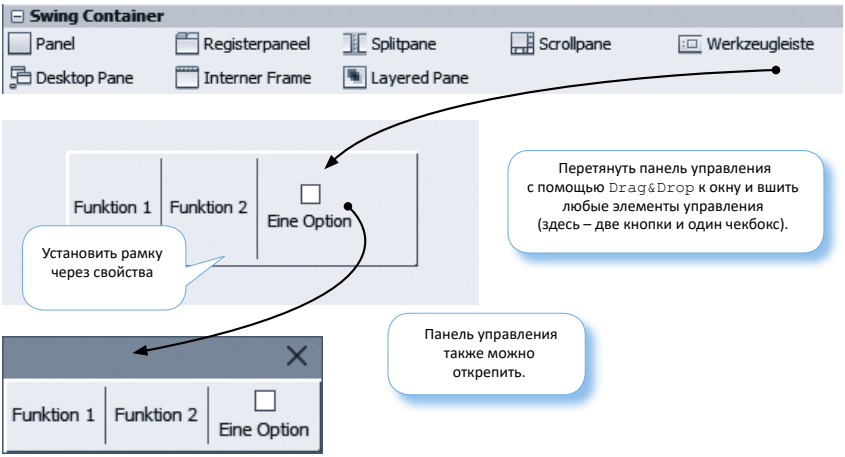


В результате будет два регистра с одной кнопкой `Button` для каждого из них в качестве элемента управления:



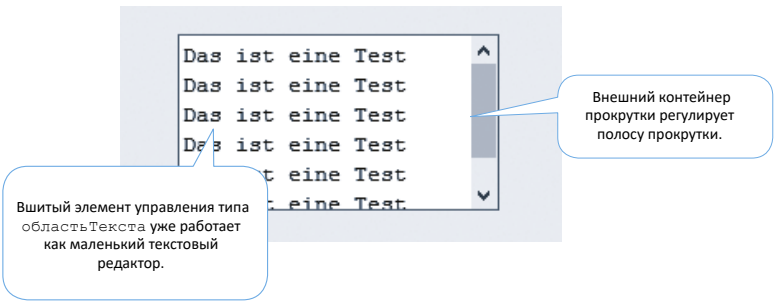
14.2.2 Панель инструментов (Toolbar)

Панель инструментов – это контейнер, который отвечает за простые элементы управления, которые часто нужны пользователю приложения. Это могут быть кнопки, выпадающие списки или также чекбоксы. Особенность панели инструментов в том, что она может быть свободно размещена пользователем. Панель может быть закреплена внутри окна, а также в собственном окне.



14.2.3 Контейнер прокрутки (Scrollpane)

В предыдущих главах уже был использован элемент управления `JScrollPane` для снабжения любых элементов управления полосой прокрутки. С помощью *меню* этот процесс упрощается. Необходимо перетянуть контейнер типа `Scrollpane` в окно и заполнить любым элементом управления. Этот элемент автоматически имеет полосу прокрутки. На следующем примере показан элемент типа ОбластьТекста, вшитый в контейнер прокрутки:



15 Соединение с базой данных

15.1 Доступ к базе данных с помощью Java

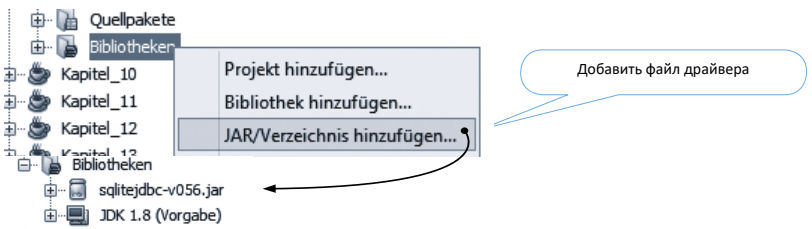
Сохранение данных приложение может осуществлять самостоятельно с помощью файловых операций. Для некоторых данных это, возможно, и лучший выбор при разработке приложения, так как оно относительно независимо. Однако когда необходимо сохранить много данных (или наборов данных) и данные имеют сложную структуру, следует рассмотреть сохранение в базе данных. Большое преимущество соединения с базой данных в том, что приложение не зависит от технической реализации сохранения данных. Этим занимается база данных на втором плане. Также изменение или удаление данных удобно производить с помощью некоторых операций базы данных (SQL-операции). Язык структурированных запросов **SQL** (Structured Query Language) играет здесь важную роль. Поэтому при выполнении следующих операций требуется знание основ SQL.

15.1.1 Соединение с базой данных с помощью JDBC

Java предлагает множество классов для реализации соединения с базой данных. Эти классы объединены в общем понятии **JDBC** (Java Database Connectivity). Для большинства баз данных существуют так называемые JDCB-интерфейсы, преобразующие доступ к базе данных из Java-приложения в соответствующие операции базы данных. Поэтому для программиста Java не имеет значения, какая база данных работает на втором плане – доступ одинаковый. На следующей схеме представлен основной принцип этого доступа:

Далее представлен доступ к базе данных SQLite. Однако принцип можно перенести на другие реляционные базы данных, например, MySQL или Oracle. При этом необходимо импортировать пакет `java.sql`. В этом пакете хранятся все важные классы для доступа к базе данных.

После загрузки желаемого драйвера (например, *sqlitejdbc-vxx.jar*) файл типа Java-архив интегрируется в проект:



После успешного добавления драйвера можно производить соединение с классом `Class`:

```
String базаданных = "jdbc:sqlite:/путь/базаданных";
```

Указать путь и файл базы данных

```
Class.forName("org.sqlite.JDBC");
```

Загрузить драйвер

Осуществить соединение

```
Connection соединение;
соединение = DriverManager.getConnection(datenbank,"","");
```

Основная база данных *Клиенты*.sqlite для этого примера находится в папке "C:\temp". Она имеет примера таблицы в качестве Клиенты с атрибутами ID (тип число) и имя (тип VARCHAR):

ID	Имя
1	Майер
2	Кнудсен
3	Кайзер
4	Франден
5	Кноблох

Совет: База данных ***SQLite*** – бесплатная и портативная база данных, соединяющая всю структуру базы данных и сами данные в одном файле. Таким образом, возможна передача Java-программ с собственной базой данных. Для небольших проектов с относительно малым объемом данных это отличная альтернатива большим базам данных типа Oracle или MySQL. Особенно удобно то, что существует расширение (*Add-on* ***SQLite Manager***) для ***Firefox-Browser***, с помощью которого можно создавать и управлять этими маленькими базами данных.

```
package глава_15;
import java.sql.*;

public class БДдоступ {
      public static void main(String[] args) {
         try {
```

Определить строку соединения с помощью данных драйвера и источника данных.

```
String базаданных = jdbc:sqlite:/c:/temp/клиенты.sqlite";
```

Загрузить драйвер.

```
Class.forName("org.sqlite.JDBC");
```

Запросить объект соединения с помощью статического метода getConnection.

```
Connection соединение
          DriverManager.getConnection(БазаДанных,"","");
```

Через объект соединения создается объект для SQL-операции.

```
Statement sqlоперация = verbindung.createStatement();
```

Через объект соединения создается объект для SQL-операции.

```
ResultSet результат =
      sqlоперация .executeQuery("SELECT * FROM Клиенты;");
```

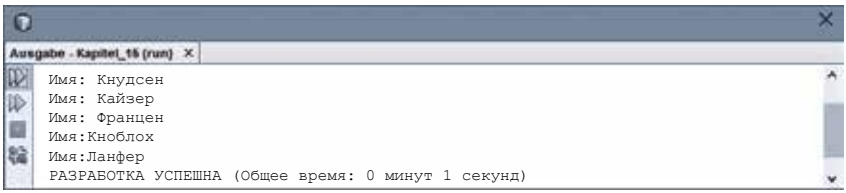
Объект события предлагает методы для вызова таблицы результатов. Метод `next` отображает, есть ли другие вводы, метод `getString` считывает следующий ввод из желаемого столбца (здесь Имя).

```
        while (событие. next( ) == true) {
                System.out.println("Имя: " +
                                    событие.getString("имя"))

            }
            Соединение.close();
        }
        catch (Exception e) {
                System.out.println(e.getMessage());
        }
    }
}
```

ВАЖНО: Соединения с базой данной должны быть снова закрыты.

ВНИМАНИЕ: При вызове баз данных очень важно использовать обработку исключений!

После запуска экран выглядит так:



База данных вызвана в нормальном режиме, все имена клиентов считаны и отображены.

Примечание:

Доступ к значениям столбцов таблицы с объектом событий производится в зависимости от того или иного типа данных. Для каждого типа данных существует специальный метод, перенимающий либо индекс, либо имя столбца:

- `getString( int индекс столбца )`
- `getString ( String имя столбца )`
- `getDouble( int индекс столбца)`
- `getDouble ( String имя столбца)`
- `getInt( int индекс столбца)`
- `getInt ( String имя столбца)`
- … другие типы

Например, первый столбец таблицы Клиенты может быть считан с помощью метода `getInt`, так как речь идет о целочисленных числовых типах:

```
    while (результат.next() == true) {
          System.out.println("ID: " +
                              результат.getInt(0));

    }
```

15.1.2 Сброс операций

Считывание любой таблицы может производиться с помощью вышеописанных операций. Если же нужно не выбирать, а добавлять, изменять или удалять, то можно сбросить так называемую операцию `executeUpdate`. Ранее желаемую SQL-операцию необходимо было создавать в строке символов. В следующем примере добавляется новая строка в таблицу Клиенты, существующая строка изменяется, одна строка удаляется:

```
package глава_15;
import java.sql.*;

public class БДдоступ {
   public static void main(String[] args) {
      try {

   String база данных = "jdbc:sqlite:/c:/temp/клиенты.sqlite";

   Class.forName("org.sqlite.JDBC");

   Connection соединение =
          DriverManager.getConnection(база данных,"","");
   Statement sqlоперация = соединение.createStatement();
```

Сброс SQL-операции и возврат количества соответствующих строк.

SQL-операция для добавления строки.

```
   int anzahl = sqlBefehl. executeUpdate ("INSERT INTO Клиенты
                        VALUES(7,'Кениг');");

   System.out.println("Количество импортированных строк: "
                     + количество);
```

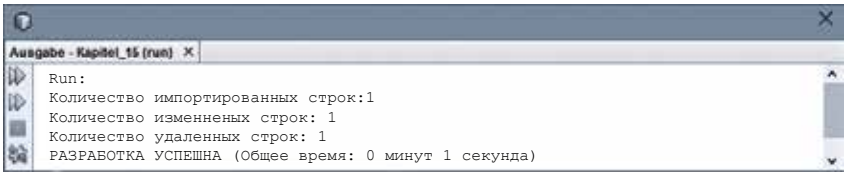
Операция UPDATE

```
   anzahl = sqlBefehl. executeUpdate ("UPDATE Kunden SET Name =
               'Кнобляух' WHERE Name = 'Кноблох';");

   System.out.println("Количество измененных строк: "
                     + количество);
```

Операция DELETE

```
   количество = sqlоперация.executeUpdate("DELETE FROM Клиенты WHERE
                     имя = Кнудсен;");

   System.out.println("Количество удаленных строк: "
                      + количество);
   соединение.close();
  }
  catch (Exception e) {
        System.out.println(e.getMessage());
  }
 }
}
```

После запуска три *невыбранные-SQL-операции* сбрасываются и выводится количество соответствующих строк:



Сравните: таблица Клиенты до и после SQL-операций:

До:

ID	Имя
1	Майер
2	Кнудсен
3	Кайзер
4	Франсен
5	Кноблох
6	Лауфер

После:

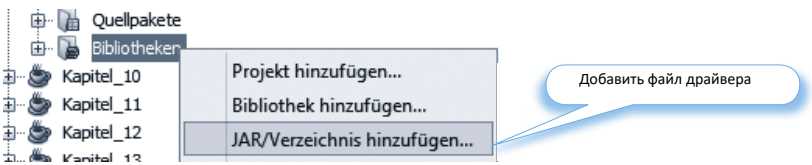
ID	Имя
1	Майер
3	Кайсер
4	Франсен
5	Кнобляух
6	Лауфер
7	Кениг

15.2 Доступ к другим базам данных

Для большинства баз данных существуют драйверы для осуществления к ним доступа с помощью Java и JDBC. Обычно драйвер нужно просто загрузить и добавить к проекту. С помощью класса `Class` и метода `foreName` драйвер базы данных можно загрузить непосредственно в виртуальную машину.

15.2.1 Добавление драйвера

После скачивания необходимого драйвера (например, *mysql-connector-java-XXX-bin.jar*) Java-архив интегрируется в проект:



После успешного добавления драйвера MySQL можно производить соединение с классом `Class`:

```
String база данных = "jdbc:mysql://ИмяСервера/базаданных";
```

Указать имя сервера для базы данных MySQL. При локальной инсталляции просто ввести localhost.

Указать имя базы данных.

```
Class.forName("com.mysql.jdbc.драйвер");
```

Загрузить драйвер.

Создать соединение с базой данных (с указанием имени пользователя и пароля).

```
Connection соединение;
соединение = DriverManager.getConnection(БазаДанных,
"пользователь","Pwd");
```

15.2.2 Другие драйверы баз данных

В следующей таблице представлен обзор стандартных баз данных и соответствующих названий драйверов Java. Соответствующий файл драйвера, как описано выше, необходимо скачать с веб-страницы производителя, либо получить его другим путем.

База данных	Название драйвера Java
Borland Interbase	`interbase.interclient.Driver`
DB2/Derby	`COM.ibm.db2.jdbc.app.DB2Driver`
Microsoft SQL Server	`com.microsoft.jdbc.sqlserver.SQLServerDriver`
MySQL	`com.mysql.jdbc.Driver`
Oracle	`oracle.jdbc.driver.OracleDriver`
PostgreSQL	`org.postgresql.Driver`

Примечание:

Перед соединением с базой данных обычно не остается альтернативы для поисков в Интернете или обзора соответствующей специальной литературы о базе данных.

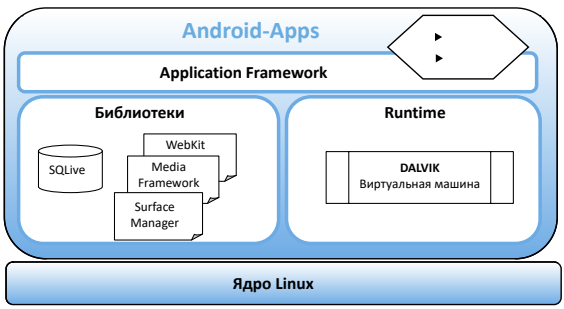
16 Разработка приложений Android

16.1 Основы приложений Android

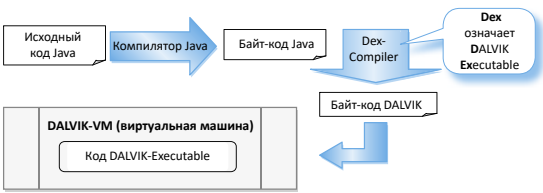
Android – это операционная система для таких мобильных устройств, как смартфоны или планшеты. Android основан на Linux, но был модифицирован для использования на мобильных устройствах. Компания Google разработала данную операционную систему и предоставляет ее для пользования бесплатно. Производители мобильных устройств могут адаптировать соответствующие устройства. Также частные пользователи могут использовать систему для модификаций. Приложения для Android называются ***Apps*** (прикладные программы) и очень просто устанавливаются на мобильные устройства. Google предлагает онлайн-магазин (Play-Store) для скачивания прикладных программ бесплатно, либо с оплатой через различные системы (PayPal или кредитная карта). Каждому пользователю доступны миллионы прикладных программ.

Прикладные программы Android разрабатываются в Java. Для разработки интегрируются различные библиотеки, и в конечном итоге, App разрабатываются в специальной виртуальной машине (DALVIK). Прикладная программа Android доступна как для Android*.***apk-файл** (для Android). Этот файл можно скопировать и установить на мобильное устройство и без использования онлайн-систем типа Play-Store (например, через USB[1]).

На схеме представлена структура прикладной программы Android:



Процесс создания прикладной программы похож на создание приложения Java, как показано на схеме ниже:

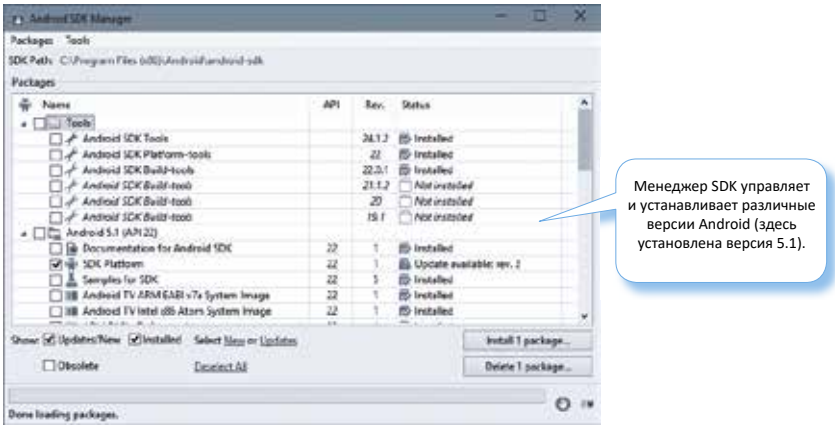


[1] Для копирования файла apk через USB обычно требуется, чтобы в настройках была разрешена установка других прикладных программ Play-Store-Apps. Затем можно выбрать файл apk для установки.

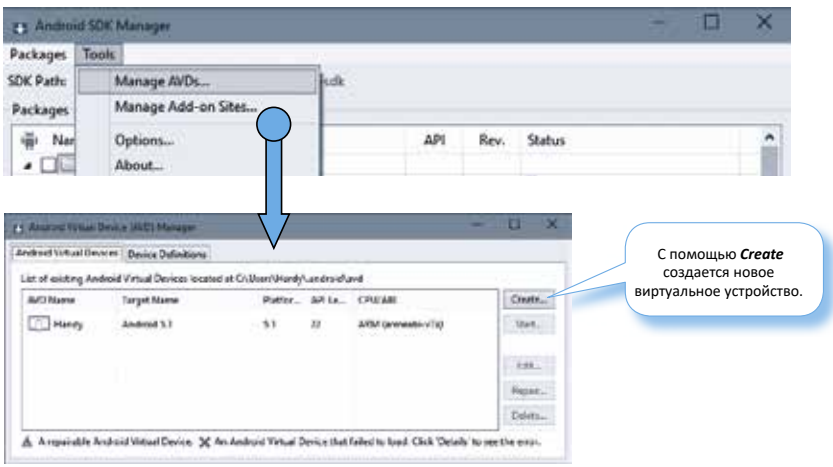
16.1.1 Установка Android SDK

Разработка прикладных программ предполагает Android SDK (Software Development Kit). Это набор инструментов и программ для создания прикладной программы Android, а также для запуска в эмуляции. Эмуляция похожа на моделирование мобильного устройства на ПК для тестирования функций программы.

После скачивания и установки бесплатного SDK с соответствующей Интернет-страницы (http://developer.android.com/sdk/installing/index.html) SDK менеджер запускается в следующем окне:



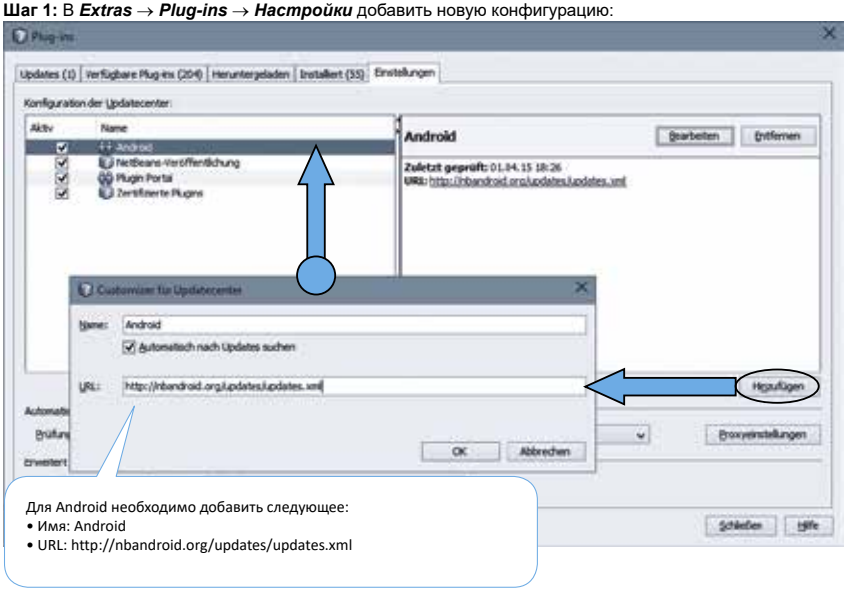
Следующий шаг – создание виртуального устройства (эмуляция) через менеджер AVD:



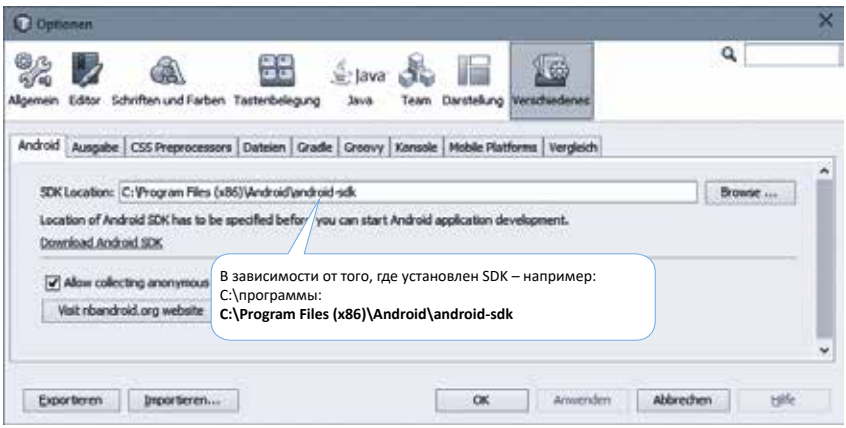
16.1.2 Подготовка NetBeans для проектов Android

После установки Android SDK и создания виртуального устройства необходимы еще следующие настройки в среде разработки NetBeans:

Шаг 1: В ***Extras*** → ***Plug-ins*** → ***Настройки*** добавить новую конфигурацию:



Шаг 2: В ***Extras*** → ***Опции*** → ***Android*** указать ***SDK Location***:

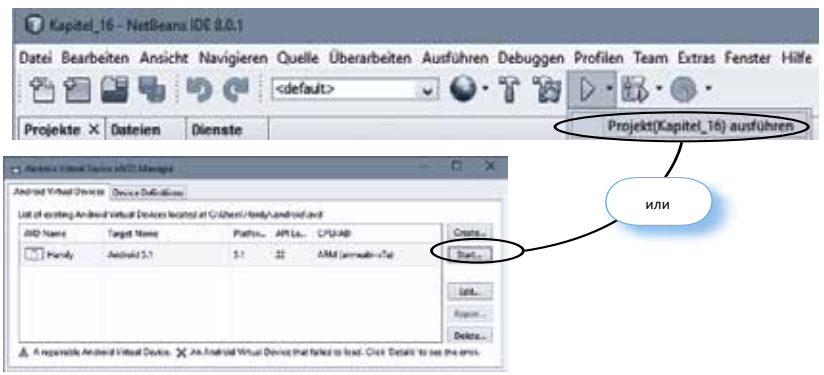


16.2 Разработка Android-Apps

16.2.1 Создание проекта Android

После подготовки NetBeans теперь может предложить новую проектную форму, а именно, ***проект Android:***

Теперь может быть запущен первый проект. Для этого NetBeans автоматически запускает виртуальное устройство или предлагает выбор виртуальных устройств, если их несколько. Устройство также может быть запущено через менеджер AVD:



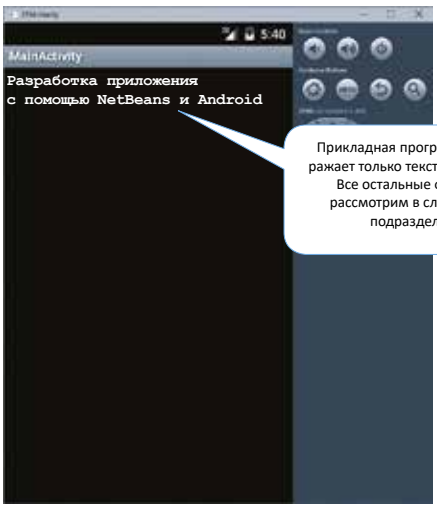
После запуска виртуального устройства экран может выглядеть так:

Примечания:

► Запуск виртуального устройства может может занять несколько минут в зависимости от производительности ПК. Очень малопроизводительные ПК могут быть перегружены.

► Чтобы прикладная программа отображалась в устройстве, проект в NetBeans, возможно, следует запустить повторно!

После успешного запуска проекта экран выглядит так:



Примечание:

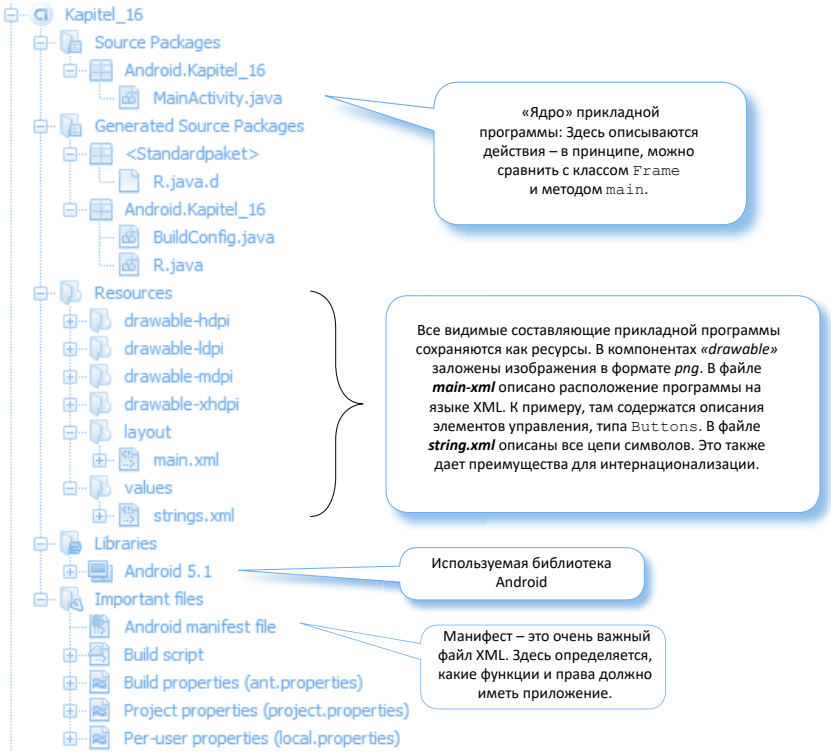
Для вышеприведенной программы был адаптирован только текст вывода. Это происходит в файле *main.xml*

В области расположения:

```
      :
      <TextView
      :
android:text="Разработка приложения с помощью NetBeans и Android"
      />
```

16.2.2 Элементы прикладной программы Android

После создания проекта Android NetBeans генерирует множество папок и файлов. Здесь представлены основные данные:



В основных файлах содержатся следующие данные:

► **„MainActivity.java“**

```
package Android.глава_16;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{

        @Override
        public void onCreate(Bundle savedInstanceState)
        {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);
        }
    }
```

Некоторые действия производятся от базового класса `Activity`.

Этот метод вызывается при создании приложения.

Вид описывается через шаблон расположения (из файла ***main.xml***).

► **main.xml“**

```
<?xml версия="1.0" encoding="utf-8"?>
<LinearLayo t
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    androi:layout_height="fill_parent"
    >
<TextView
    androi:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Разработка приложения с помощью NetBeans
                                                и Android"
    />
</LinearLayout>
```

Версия XML

Определить расположение – похоже на контейнер в Swing

Определить элементы управления – здесь вид текста

► **„strings.xml”**

```
<?xml версия="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">MainActivity</string>
</resource
>
```

Все строки описываются как ресурсы

► **„Android manifest file“**

```
<?xml версия="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
     package="Android.глава_16"
     android:versionCode="1"
     android:versionName="1.0">

     <application android:label="@string/app_name"
              android:icon="@drawable/ic_launcher">

     <activity android:name="MainActivity"
                  android:label="@string/app_name">
          <intent-filter>
          <action android:name="android.intent.action.MAIN" />
                  <category android:name="
                        android.intent.category.LAUNCHER" />
              </intent-filter>
         </activity>
    </application>
</manifest>
```

Android Package

Указать ресурсы приложения

Определить основную деятельность

16.2.3 Адаптация расположения прикладной программы

Файл XML *main.xml* описывает расположение прикладной программы. Внутри контейнера расположения типа `LinearLayout` могут быть вшиты элементы управления. При этом элементы управления всегда должны иметь связь с контейнером. На следующем примере `TextView` и `Button` вшиваются в контейнер:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >

<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Первое приложение с Button"
/>

<Button
android:layout_width="fill_parent"
android:layout_height="wrap_conte
android:text="ЩЕЛЧОК"
/>

</LinearLayout>
```

Описывается кнопка. Ширина указывается с помощью „fill_parent". Это означает, что кнопка внутри контейнера расположения принимает полную ширину. С помощью „wrap_ content" элемент управления принимает высоту, соответствующую содержанию.

Примечание:

Тексты элементов управления здесь сразу указываются как цепи символов. Для начала это нормально, однако более целесообразно описывать ресурсы строки:

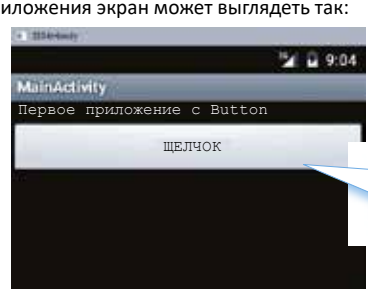
strings.xml:

```
<resources>
      <string name="button_text">ЩЕЛЧОК</string>
</resources>
```

main.xml:

```
<Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/button_text"
/>
```

После запуска приложения экран может выглядеть так:



Кнопка отображается, однако пока не имеет функций.

16.2.4 Другие контейнеры расположения и элементы управления

Для упорядочивания элементов управления существуют различные контейнеры расположения, которые в основном работают так же, как и в программировании Swing:

Контейнер расположения	Описание
Linearlayout	Самый простой Layout. Элементы располагаются просто друг за другом.

RelativeLayout	Этот Layout предлагает возможность определить внутри отдельных элементов управления, какой будет следующий.
	Пример:
	`<RelativeLayout`
	
	`<Button`
	`android:id="@+id/button_1"`
	
	`:`
	`android:layout_centerHorizontal="true"`
	`android:layout_centerVertical="true"`
	`android:text="Button 1" />`
	
	`<Button`
	`android:id="@+id/button_2"`
	`android:layout_above="@+id/button_1"`
	`android:text="Button 2" />`
	`</RelativeLayout>`

GridLayout	Этот Layout располагает элементы управления в виде таблицы.
	Пример:
	`<GridLayout xmlns:android=`
	`"http://schemas.android.com/apk/res/android"`
	`android:layout_width="wrap_content"`
	`android:layout_height="wrap_content"`
	`android:columnCount="2"`
	`android:rowCount = "2" >`
	
	`<Button`
	`android:layout_column="0"`
	`android:layout_row="0"`
	`android:text="Button 1" />`
	
	`<Button`
	`android:layout_column="1"`
	`android:layout_row="0"`
	`android:text="Button 2" />`
	
	`<Button`
	`android:layout_column="0"`
	`android:layout_row="1"`
	`android:text="Button 3" />`
	
	`<Button`
	`android:layout_column="1"`
	`android:layout_row="1"`
	`android:text="Button 4" />`
	`</GridLayout>`

Элементы управления	**Описание**
TextView	Является своего рода меткой для отображения текста.
Button	С помощью этого элемента пользователь может вызвать действие («Щелчок»).
EditText	Соответствует текстовому полю для принятия ввода пользователя.
CheckBox	Служит для выбора опции.
RadioBox	Служат для выбора нескольких (исключающих) опций.

16.2.5 Программирование элементов управления

До сих пор элементы управления были определены только в файле XML, соединение с исходным кодом Java отсутствует, чтобы иметь возможность реагировать на события и т. п. Поэтому необходимо, чтобы все элементы управления получили однозначный ID. С помощью ID в Java можно инстанцировать соответствующий объект. С помощью этого объекта можно программировать элемент управления (например, реагировать на события). В следующем примере показан доступ к кнопке:

Сначала следует адаптировать файл расположения:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/
                                                    res/android"

<Button
android:id="@+id/моякнопка"
:
/>
</LinearLayout>
```

Кнопка получает однозначный ID *"моякнопка"*.

Далее можно получить доступ к кнопке в файле Java:

```
package Android.глава_16;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
      :
Button b = (Button) findViewById(R.id.МояКнопка);

b.setOnClickListener(new OnClickListener() {
@Override
      public void onClick(View v) {
            ((Button)v).setText("OK, щелчок")
      }
   });
}
}
```

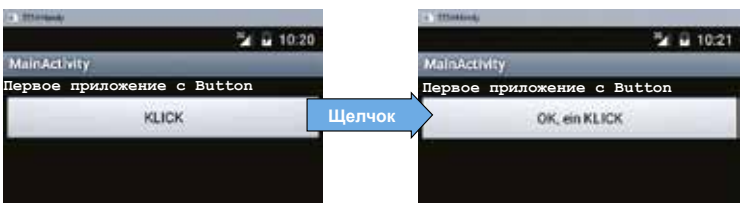
ВАЖНО: интегрировать пакеты

Инстанцирование кнопки с привязкой к ID.

Создать приемник.

Описать метод `onClick`. Передаваемый параметр типа `View` необходимо преобразовать в соответствующий тип (здесь в `Button`).

После запуска приложение выглядит так:



Примечание:

► Альтернативно в файле XML также можно указать метод события, который затем нужно описать в классе деятельности:

```
<Button
      android:id="@+id/моякнопка"
      :
      android:onClick="Щелчок"
/>

public class MainActivity extends Activity
{
           :
           public void Щелчок(View view) {
                  Button b = (Button) view;
                  b.setText("OK, Щелчок"

          }
}
```

Указать метод щелчка

Описать метод щелчка

Программирование других элементов управления происходит аналогично `Button`. Например, ввод пользователя в элементе управления `EditText` можно вызвать так:

Сначала снова адаптируется файл расположения:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
                                                          android"
:
>

<EditText
android:id="@+id/editВводТекста"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>

<Button android:id="@+id/buttonкопировать"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/button_text"
/>

<TextView
android:id="@+id/textViewПоказ"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>

</LinearLayout>
```

Далее доступ к элементам управления в файле Java:

```
package Android.глава_16;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity
{
   private Button b;
   private TextView t;
   private EditText e;

 @Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);


   b = (Button) findViewById(R.id.buttonкопирование);
   t = (TextView) findViewById(R.id.textViewпоказ);
   e = (EditText) findViewById(R.id.editтекстввод);

   b.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v)
       t.setText(e.getText());
  }
  });
  }
}
```

Ссылки на элементы управления

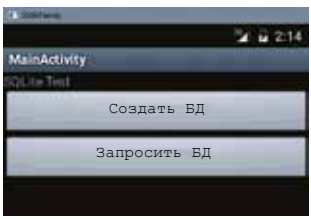
Инстанцирование с помощью ID

Щелчок по кнопке отвечает за то, что введенный текст пользователя вызывается через `t.setText()` и передается элементу `TextView`.

После запуска приложение может выглядеть так:

16.2.6 Запрос базы данных SQLite

С базой данных SQLite вы уже знакомы из главы о базах данных. В целях практики Android SDK уже интегрировал драйвер SQLite и соответствующие пакеты, так что создать и вызвать базу данных очень просто. На следующем примере показано, как создается, заполняется и запрашивается база данных SQLite. Для этого создаются две кнопки *(создать БД и Запросить БД)* и `TextView` для отображения результатов вызова. Приложение запускается с представления результатов:



```
package Android.SQLite;

import android.app.Activity;
import android.os.Bundle;
import android.database.*;
import android.database.sqlite.*;
import android.view.View;
import android.widget.TextView;


public class MainActivity extends Activity
{
        private SQLiteDatabase db;
        private TextView t;

       @Override
       public void onCreate(Bundle savedInstanceState)
       {

         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);

         t = (TextView)findViewById(R.id.БДрезультат);
}

public void БД_создать(View view) {
     try {

db = openOrCreateDatabase
            ("Test.sqlite", MODE_PRIVATE, null);

db.execSQL("CREATE TABLE IF NOT EXISTS Лица
                        id NUMBER, name VARCHAR);");

db.execSQL("INSERT INTO Лица VALUES(1,'Петер');");
db.execSQL("INSERT INTO Лица VALUES(2,'Клаус');");
db.execSQL("INSERT INTO Лица VALUES(3,'Маркус');");


db.close();
}
catch(Exception e){
     t.setText( e.getMessage() {
 }
}


public void  DB_вызов(View view) {
   try {
   db = openOrCreateDatabase
                ("Test.sqlite", MODE_PRIVATE, null);
```

Интегрировать пакеты

Ссылка на объект БД SQLite

TextView для отображения данных

Метод «щелчок» для первой кнопки

Действие предлагает метод для создания БД. MODE_PRIVATE означает, что БД создается эксклюзивно для данного приложения.

С помощью метода execSQL объекта БД выполняется операция SQL.

Добавить три набора данных с помощью INSERT

Метод «щелчок» – для второй кнопки

Перехват ошибки и отображение в TextView

```
Курсор c = db.rawQuery("SELECT * FROM Лица", null);
```

Этот метод дает результат вызова в форме курсора, который можно обработать в дальнейшем.

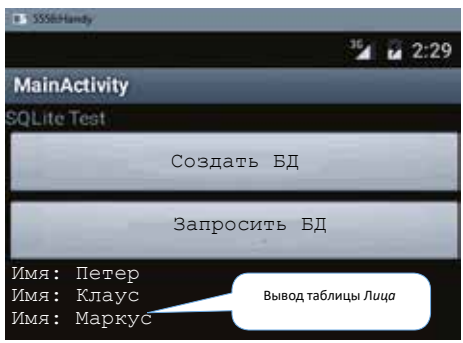
Возвращает количество строк с результатом.

```
    if(c.getCount()==0) {
      t.setText("Не найдено наборов данных");
      return;
    }

    String dummy = "";

    while(c.moveToNext()){
          dummy += ("имя: " + c.getString(1)+ "\n");
    }
    t.setText(dummy);
}
catch(Exception e){
        t.setText( e.getMessage());
}
}
}
```

С помощью `MovetoNext` курсор переходит от одной строки к другой.

С помощью метода `getString` и указания столбца (ВНИМАНИЕ: начинается с нуля) составляется вывод на экран.

После щелчка по кнопке «Создать БД» далее с помощью кнопки «Запрос БД» можно видеть результат в `TextView`:



Примечания:

В эмуляции база данных находится только в памяти и удаляется после повторного запуска. Однако если приложение установлено на мобильное устройство, то база данных сохраняется и может быть использована приложением в дальнейшем.

Данная глава предлагает введение в мир программирования приложений. С помощью этих основ в дальнейшем можно расширить знания, используя дополнительную литературу или онлайн-помощник Android SDK. Остальные аспекты многогранны и трудоемки. Также в будущем можно перейти к среде разработки, которая предлагает GUI-конструктор для приложений – NetBeans в данной версии, к сожалению, не имеет поддержки.

Часть 2
Задания

1 Задания к главе «Введение в технологию Java» 226
2 Задания к главе «Первая программа Java» 226
3 Задания к главе «Ввод и вывод в Java» 227
4 Задания к главе «Операторы в Java» 228
5 Задания к главе «Селекция и итерация» 230
6 Задания к главе «Понятие классов в Java» 234
7 Задания к главе «Наследование в Java» 237
8 Задания к главе «Массивы в Java» 240
9 Задания к главе «Файловые операции в Java» 245
10 Задания к главе «Темы Java для продвинутого уровня» 251
11 Задания к главе «GUI-программирование с помощью AWT» 253
12 Задания к главе «Элементы управления с помощью AWT классов Swing» 256
13 Задания к главе «Меню, диалоги и апплеты» 258
14 Задания к главе «GUI-конструктор NetBeans» 259
15 Задания к главе «Соединение с базой данных» 261
16 Задания к главе «Разработка приложения» 262

Задания

1 Задания к главе «Введение в технологию Java»

Задание 1.1

Найдите в информационном блоке и в интернете (или других открытых источниках) следующие основные понятия Java:

- JDK
- JRE
- JVM
- garbage collector
- Байткод
- HotSpot
- Парадигма программирования

Задание 1.2

Проведите поиск в Интернете или других источниках и составьте схему языков программирования, в которой представлены основные языки (Java, C++, C#, Delphi, BASIC, COBOL и др.) и их создание с примерными временными рамками.

Задание 1.3

Представьте наглядный процесс разработки программы в Java (от исходного кода до работающей программы) в форме диаграммы.

2 Задания к главе «Первая программа Java»

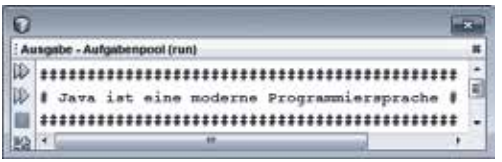
Задание 2.1

Проанализируйте следующую программу Java. Какие ошибки допущены?

```
package java_it_berufe;

public class Java_IT_Berufe ()

       public static void main(String[] args) {
              System.out.println('Привет, Java!');
}
```

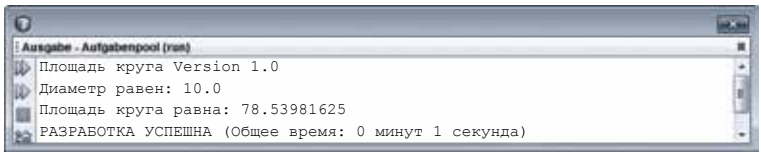
Задание 2.2

Напишите программу Java, имеющую следующий вывод на экран:



Задание 2.3

Напишите программу Java, рассчитывающую площадь круга. Используйте для этого постоянную переменную `PI`, которая инициализируется со значением `3,14159265`. Далее определите в программе диаметр круга. После запуска на экране должен появиться следующий вывод (задан диаметр 10):



3 Задания к главе «Система ввода-вывода в Java»

Задание 3.1

Необходимо рассчитать конечный капитал сберегательного вклада. Для этого следует ввести с клавиатуры стартовый капитал и размер процентной ставки. Капиталовложение всегда осуществляется в течение трех лет. Учтите эффект сложного процента. Далее необходимо вывести конечный капитал. Выберите соответствующие типы данных для переменных.

Пример:

- Стартовый капитал: 1.000 (евро)
- Процентная ставка: 5 (процентов)
- Конечный капитал: 1.157,625 (в евро через 3 года)

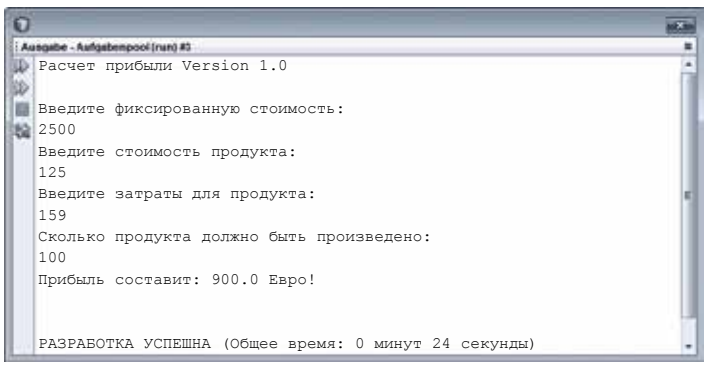
Задание 3.2

Исходная ситуация:

На предприятии производится продукция. Предприятие заложило для продукции фиксированные цены от **X** евро и производственные затраты для каждого продукта в размере **Y** евро. Ожидается доход в размере **Z** евро с каждого продукта.

Задание:

Напишите программу Java, считывающую данные фиксированных цен, расходов на продукт и доход с продукта. Далее необходимо рассчитать прибыль для заданного количества продукции. После запуска программа может выглядеть так:

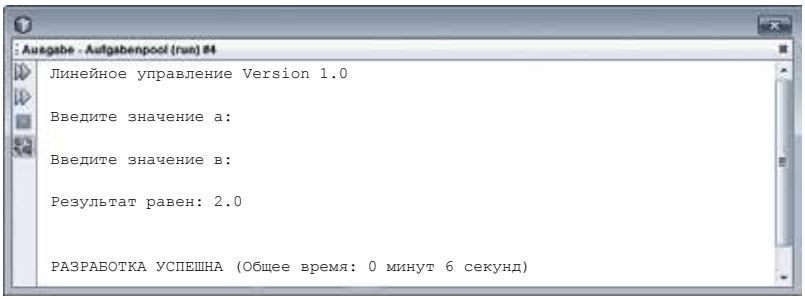


Задание 3.3

Напишите программу Java, которая может решать линейные уравнения. Для этого пользователь должен указать значения a и b. Затем программа рассчитывает результат.

Общая формула линейного уравнения: **ax + b = 0**

После запуска программа может выглядеть так:



4 Задания к главе «Операторы в Java»

Задание 4.1

Определите значение переменной x. Действительны следующие условия:

```
int a = 10;
int b = 20;
int x;
```

- `x = 3 * (a + b) - b/8;` x =_____
- `x = (a++) + (++b);` x =_____
- `x = (a % b ) % ( b % (++a) );` x =_____

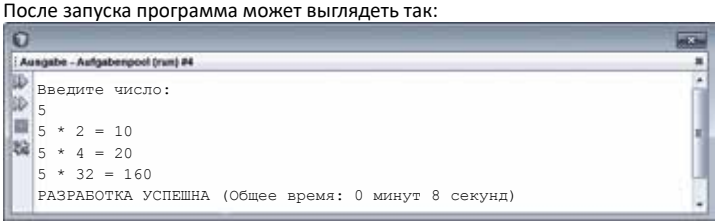
Задание 4.2

Напишите программу Java, которая считывает два целых числа с клавиатуры и далее выводит остаток деления на экран. Напишите программу без использования оператора модуля.

Задание 4.3

Напишите программу Java, которая считывает целое число. Далее число нужно умножить на 2, 4 и 32 без использования мультипликативного оператора.

После запуска программа может выглядеть так:



Задание 4.4

В сетевом оборудовании выделение **подсети** используется для создания других подсетей в классе сети. При этом IP-адреса связывают так называемой маской **(маской подсети)** и в результате получают сетевые адреса. Если этот сетевой адрес одинаковый для двух IP-адресов, то оба IP-адреса относятся к одной подсети (Subnet).

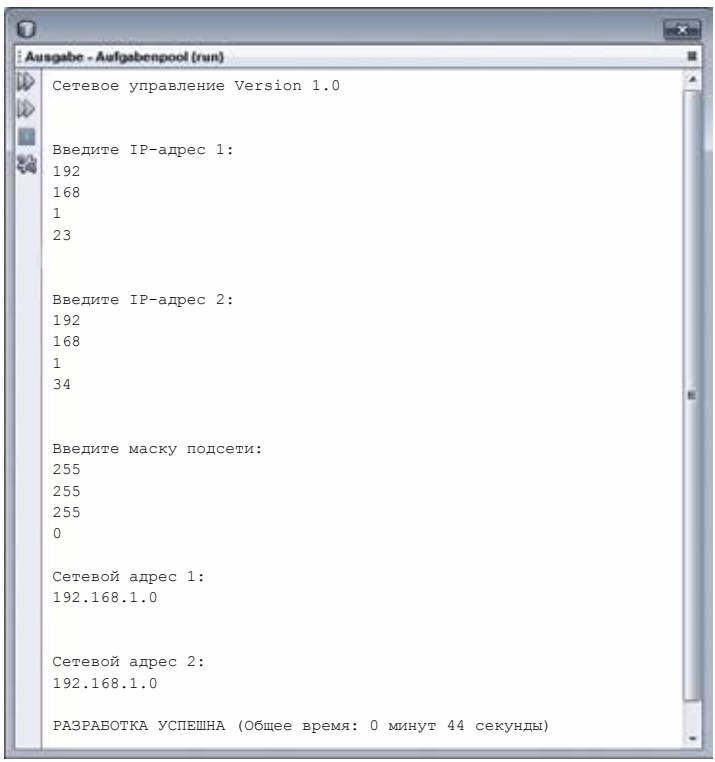
Пример:

```
IP-адрес 1: 192.168.1.23
IP-адрес 2: 192.168.1.34
Маска подсети: 255.255.255.0
```

Отдельные компоненты IP-адресов теперь соединяются посредством побитового И-оператора & с помощью маски подсети. Результатом является сетевой адрес.

```
IP-адрес 1: 192.168.1.23
     &            &
Маска подсети: 255.255.255.0
-------------------------------------------------------------
Сетевой адрес 1: 192.168.1.0



IP-адрес 2: 192.168.1.34
      &            &
Маска подсети: 255.255.255.0
-------------------------------------------------------------
Сетевой адрес 2: 192.168.1.0
```

Напишите программу Java, которая считывает два IP-адреса (по 4 целых переменных) и маску подсети с клавиатуры и затем выводит сетевые адреса на экран. После запуска программа может выглядеть так:



5 Задания к главе «Селекция и итерация»

Задание 5.1

Напишите программу Java, имеющую следующие функции:
Необходимо считать с клавиатуры три числа (тип данных `double`). Далее следует вывести на экран минимум и максимум обоих чисел.

Пример вывода на экран:
Пожалуйста, введите первое значение: 5
Пожалуйста, введите второе значение: 33
Пожалуйста, введите третье значение: 22
Максимум: 33
Минимум: 5

Дополнение: При выполнении используйте ровно три оператора `if` (без else).

Задание 5.2

Разработайте программу Java, которая проверяет корректность введенной даты. Дата вводится в формате день, месяц и год, где значения следует сохранить в трех целых переменных:

Примеры:

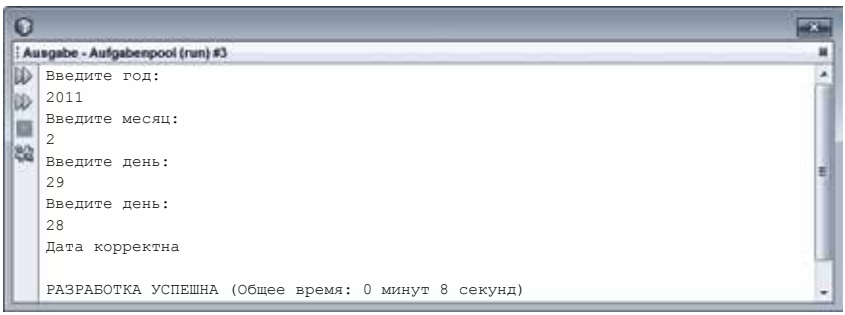
- 10 5 2005 Корректная дата
- 15 13 2001 Некорректная дата
- 29 2 2000 Некорректная дата (високосный год)

Программа также должна уметь проверять особенность високосного года (в високосном году в феврале 29 дней). При некорректном вводе необходим повторный ввод.

Совет:

Год является високосным, если
- он делится на 4, но на 100.
- он делится на 4, на 100 и на 400.

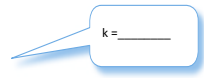
После запуска программа может выглядеть так:

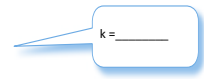


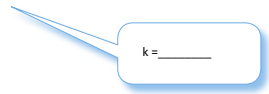
Задание 5.3

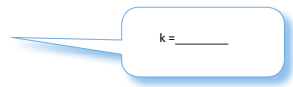
Проанализируйте следующие циклы `for`. Определите значение k после окончания цикла.

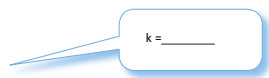
```
int i, j, k;
k = 0;
for (i = 1; i < 10; i = i + 1)k = k + i;
System.out.println("Значение k: " + k);

k = 0;
for (i = 2; i < 10; i = i + 2)k = k + i;
System.out.println("Значение k: " + k);

k = 0;
for (i = 1, j = 5; (i < 5) && (j > 1); i++, j--) k = k + i * j;
System.out.println("Значение k: " + k);

k = 0;
for (i = 1; i < 5; i++)
{

      if (i == 3) continue;
      k = k + i;
}
System.out.println("Значение k: " + k);

k = 0;
for (i = 1; i < 10; i++)
{
     k = k + i;
     if (i == 6) break;
}
System.out.println("Значение k: " + k);
```











Задание 5.4

Постарайтесь получить следующий вывод на экран **с помощью двух циклов for.**

```
10 9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
6 5 4 3 2 1 0
5 4 3 2 1 0
4 3 2 1 0
3 2 1 0
2 1 0
1 0
0
```

Задание 5.5

Используйте цикл `for` для решения следующих проблем:

► Необходимо ввести с клавиатуры одно целое число и затем вывести на экран все натуральные числа до этого числа.

Пример:

Ввод: 8 Вывод: 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8

► Необходимо ввести с клавиатуры одно целое число и затем вывести на экран все четные натуральные числа, начинающиеся от этого числа до 2.

Пример:

Ввод: `12` Вывод: `12 , 10 , 8, 6 , 4 , 2`

► Напишите программу, которая считает от 1 до 10 и сразу наоборот. Программа может использовать только один цикл `for`.

Пример:

Вывод: `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1`

Задание 5.6

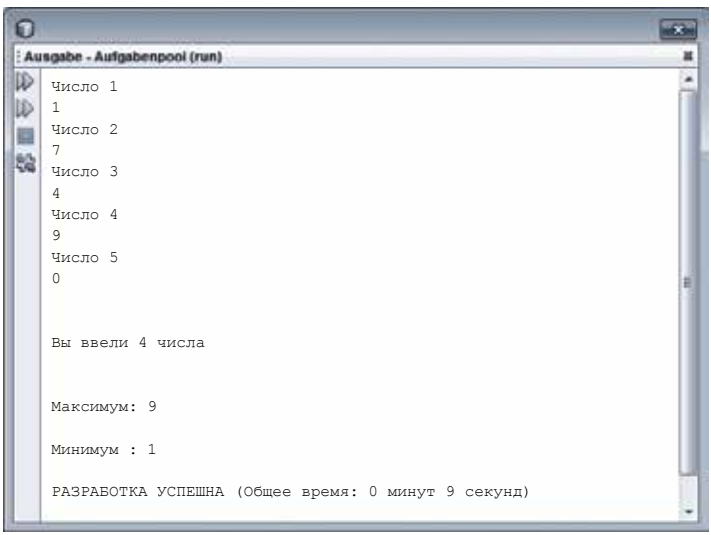
Напишите программу Java, имеющую следующие возможности:

Пользователь может вводить любое количество положительных целых чисел (целые значения). При вводе нуля программа должна вывести на экран количество введенных чисел, а также наибольшее и наименьшее число.

Необходимо использовать только следующие переменные:

- `int вводчисло;`
- `int количество;`
- `int min;`
- `int max;`

После запуска программа может выглядеть так:



Задание 5.7

Пользователю необходимо ввести друг за другом четыре отдельных символа (**не** `String`), которые должны быть паролем. Эти четыре символа должны быть проверены по следующим критериям:

Оригинальный пароль (или последовательность символов): `P R O G`

Если пользователь ввел эти символы, на экране должно появиться следующее сообщение: «ЛОГИН корректный». Тем не менее, пользователю разрешается ввести четыре символа в любой последовательности прописными или строчными буквами.

Например, допускаются следующие варианты ввода:

- `R P G O`
- `P G o r`
- `O R p g` и др.

Пользователь может ввести пароль максимум три раза, в противном случае программа должна выдать сообщение об ошибке.

Советы:

► Считывание отдельных символов можно произвести так:

```
BufferedReader считывание =
        new BufferedReader(new InputStreamReader(System.in));
char a = считывание.readLine().charAt(0);
```

► Преобразование в прописные буквы можно осуществить с помощью статического метода `toUpperCase`:

```
char a = 'x'
a = Character.toUpperCase(a);     // a == 'X'
```

Задание 5.8

Напишите программу Java, проверяющую ввод пользователя. Для этого пользователь вводит код в форме целого числа, который затем необходимо проверить по следующим критериям:

► Число должно быть пятизначным.

► Число не должно делиться на 3, 5 или 7.

► Если число начинается на 1, то последняя цифра также должна быть 1.

► Пятая цифра является контрольной. Она должна равняться остатку деления суммы первых четырех цифр на 7.

Примеры:

► Ввод:	12345	Некорректный ввод (последняя цифра не 1)
► Ввод:	56442	Некорректный ввод (делится на 3)
► Ввод:	23456	Некорректный ввод (контрольная цифра неверна)
► Ввод:	45454	Корректный ввод

Если ввод корректный, программа должна завершиться с соответствующим сообщением («Ввод корректный»). В противном случае ввод должен повторяться.

Примечания:

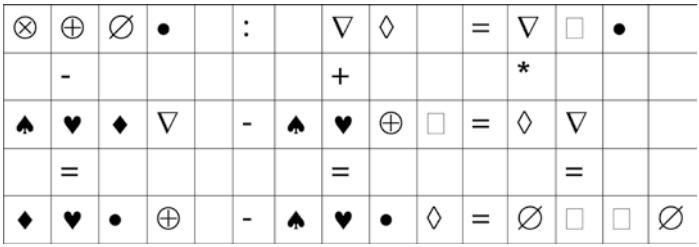
• Можно использовать только простые типы данных (но не строковый тип) и операторы.

• Активно используйте арифметические операторы (вкл. оператор модуля) и учтите, что деление целых чисел не имеет количества разрядов после запятой.

Задание 5.9

Интересная проблема, которую можно решить с помощью селекции и итерации, это задание, которое используется в рубрике «Загадки» в различных развлекательных журналах.

Проблема: Числовой ребус

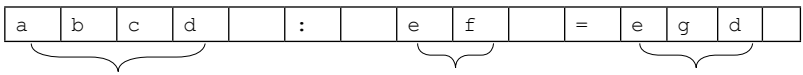


Каждый символ используется для одной цифры. Всего существует шесть уравнений с десятью неизвестными цифрами. С точки зрения математики здесь нет однозначного решения. Поэтому тот, кто разгадывает загадку, пытается решить проблему с помощью размышлений.

Напишите программу Java, которая моделирует все варианты распределения цифр и каждый раз осуществляет шесть расчетов. Когда верные цифры будут найдены, они должны быть выведены на экран.

Совет:

Для проверки расчетов отдельные цифры (переменные) необходимо свести к одному числу:



(a · 1000 + b · 100 + c · 10 + d) : (e · 10 + f) = (e · 100 + g · 10 + d)

6 Задания к главе «Понятие классов в Java»

Задание 6.1

Напишите класс `Точка`, который будет представлять точку в системе координат. Для этого необходимо создать атрибуты для координат x и y.

► Кроме стандартного конструктора за инициализацию точки должны отвечать два конструктора с параметрами:

```
public Точка() { . . . }

public Точка(double a, double b)   { . . . }

public Точка(точка p) { . . . }
```

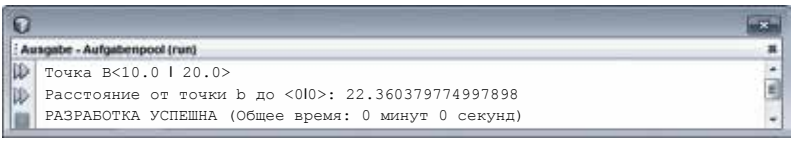
► Метод `Расстояние` должен рассчитать и вернуть значение расстояния точки от начала координат.

В основном методе класс может быть использован следующим образом:

```
public static void main(String[] args) {
        Точка a = new Точка (10, 20);
        Точка b = new Точка (a);
        Точка c = new Точка ();

        c.setX(30);
        c.setY(40);
```

```
System.out.println("Точка B<" + b.getX() + "|" + b.getY() + ">");
System.out.println("Расстояние от точки b до <0|0>: "
                   + b.расстояние());
}
```

После запуска экран выглядит так:



Примечание:

Класс `Math` предлагает некоторые статические методы типа `sqrt` для расчета, к примеру, квадратного корня.

Задание 6.2

Напишите класс `Число` в Java со следующими функциями:

Класс имеет целое число в качестве атрибута (создать соответствующие методы и конструкторы). Число может принимать значения только от 0 до 999, метод `spell` должен выводить целое число как полностью написанное число.

Пример:

```
Число z = new Число(213);
z.spell();
```

После запуска экран выглядит так:

Задание 6.3

Разработайте класс `IPадрес` для сохранения `IP-адреса`. При этом IP-адрес должен быть сохранен в частном атрибуте типа `String`. Перед сохранением необходимо проверить IP-адрес. В случае если IP-адрес не корректный, необходимо вывести сообщение об ошибке и сохранить адрес «`0.0.0.0`».

Необходимо создать следующие перегруженные конструкторы:

- `public IPадрес (){...}`
- `public IPадрес (int a, int b, int c, int d){...}`
- `public IPадрес (String s) ){...}`

Далее необходимо исполнить методы Get и Set, и, если IP-адрес не действителен, вернуть значение false.

- `public boolean setIP(int a,int b, int c, int d) {...}`
- `public boolean setIP(String s) {...}`
- `public string getIP(){...}`

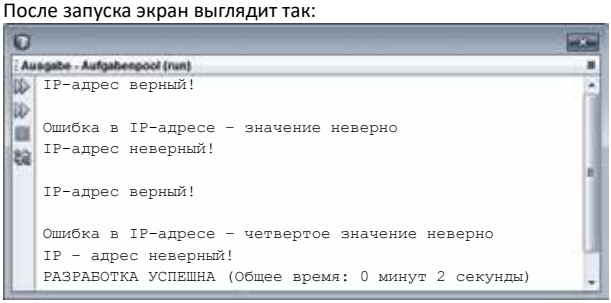
Примечания:

- Необходимо использовать только простые типы строк.
- Напишите частные методы для осуществления проверки IP-адресов.

Тестовая программа:

```
public static void main(String[] args) {

   IPадрес iP = new IPадрес ();

   if (iP.setIP("12.111.222.123")==true)
       System.out.println("IP-адрес верный!");
   else System.out.println("IP-адрес неверный!");

   System.out.println();
   if (iP.setIP("..0.000") == true)
       System.out.println("IP-адрес верный!");
   else System.out.println("IP-адрес неверный!");

   System.out.println();
   if (iP.setIP("012.1.10.000") == true)
       System.out.println("IP-адрес верный!");
   else System.out.println("IP-адрес неверный!");

   System.out.println();
   if (iP.setIP("123.12.0.") == true)
       System.out.println("IP-адрес верный!");
   else System.out.println("IP-адрес неверный!");
}
```

После запуска экран выглядит так:



Примечание:

Активно используйте такие методы, как `substring` или `indexOf` класса `String`.

Задание 6.4

Для занятий по математике в профессиональной школе требуется разработать класс, имеющий в распоряжении некоторые полезные методы и атрибут. Исполните для этого следующие статические методы и статический атрибут в классе `математика`:

► Метод ***степени***: этот метод возводит в степень переменную типа `double` с заданным показателем. Рассчитанное значение степени должно возвращаться.

► Метод ***факториала***: этот метод принимает целое значение, рассчитывает факториал и возвращает значение.

Примечание:

Факториал натурального числа равен произведению всех чисел от 1 до этого числа. Пример: 5! (! означает факториал) = 1 · 2 · 3 · 4 · 5 = 120

► Метод ***суммы цифр числа***: Этот метод определяет и возвращает сумму цифр переданного целого числа.

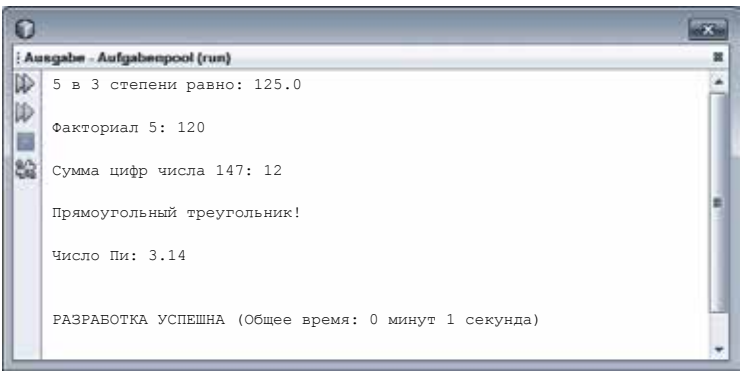
► Метод ***треугольника***: Этот метод принимает три стороны треугольника и проверяет, прямоугольный ли это треугольник. Если да, метод возвращает значение `true`, в противном случае – `false`.

► Постоянная `Пи`: эта постоянная представляет число Пи (3.14).

Пример использования класса в основном методе:

```
public static void main(String[] args) {
   System.out.println("Третья степень 5 равна: "
                      + математика.степень(5, 3));
   System.out.println();
   System.out.println("Факториал 5: " + математика.факториал(5));
   System.out.println();
   System.out.println("Сумма цифр числа 147: "
                      + математика.сумма цифр числа(147));
   System.out.println();
   if (математика.треугольник(3,4,5) == true)
      System.out.println("Прямоугольный треугольник!");
   else
      System.out.println("Не прямоугольный треугольник!");
System.out.println();
System.out.println("Число Пи: " + математика.Пи );
System.out.println();
}
```

После запуска экран выглядит так:



7 Задания к главе «Наследование в Java»

Задание 7.1

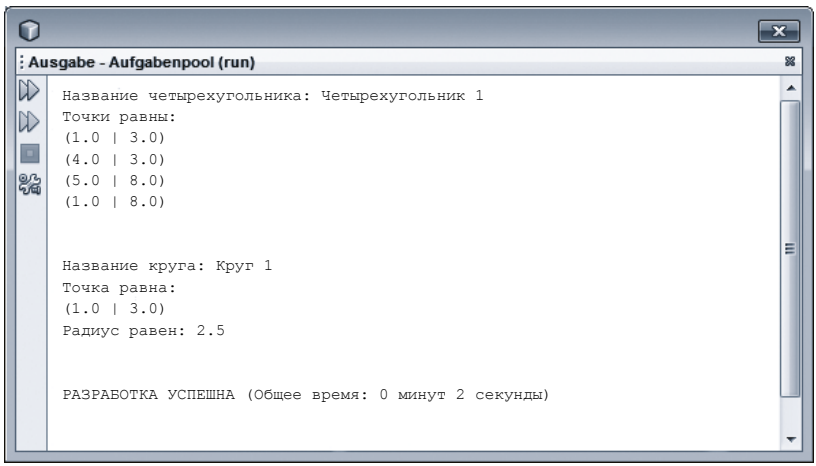
Создайте класс `ОсновнаяФорма`, который может служить базовым классом для основных геометрических форм. Классу нужен только атрибут (`String`) для обозначения. Создайте два других производных класса от основной формы: `четырехугольник` и `круг`. Эти два класса также должны иметь атрибуты, отражающие свойства четырехугольника и круга: четырехугольник можно описать с помощью четырех точек в системе координат, а круг с помощью центральной точки и радиуса.

Используйте класс `Точка` из задания 6.1 или создайте класс `Точка`, который представляет точку в системе координат (координаты x и y). Используйте эти классы для атрибутов классов форм. Класс должен иметь соответствующий метод для установки, чтения атрибутов и вывода на экран.

Пример использования:

```
public static void main(String[] args) {

        четырехугольник v = new четырехугольник ("четырехугольник 1");
        круг k = new круг("круг 1");

        точка p1= new точка (1,3);
        точка p2= new точка (4,3);
        точка p3= new точка (5,8);
        точка p4 = new точка (1, 8);
        double радиус = 2.5;

        v.setточки(p1,p2,p3,p4);
        k.setточкарадиус(p1,радиус);

        v.вывод(); System.out.println();
        k.вывод(); System.out.println();
}
```

После запуска экран выглядит так:

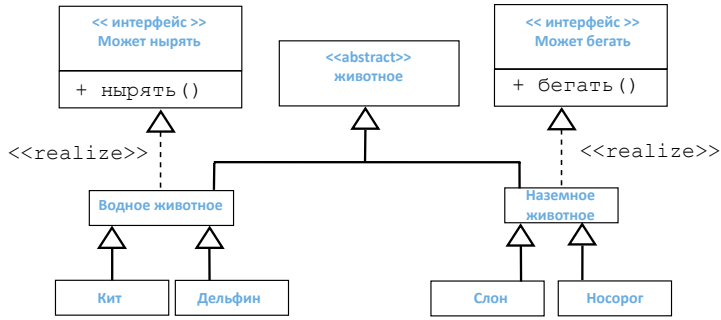


Задание 7.2

Для зоопарка необходимо написать ПО, «управляющее» следующими животными: слоны, носороги, дельфины и киты. Напишите для всех животных класс в Java. Каждое животное должно иметь атрибут `Имя`. Также для всех животных, живущих в воде, необходимо исполнить метод `Нырять`. Для всех животных, живущих на земле, необходимо исполнить метод `Бегать`. Метод `ИнформационныйЛист` должен выводить данные животных на экран.

Дополнительные критерии для применения:

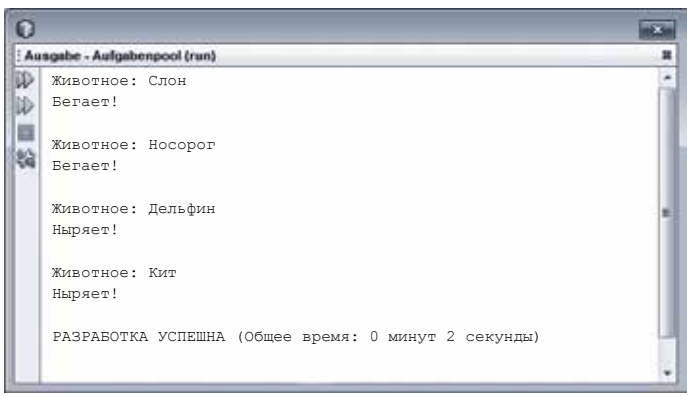
- Целесообразно установить наследование и абстрактные базовые классы.
- Используйте возможности интерфейса для предопределения исполнений.
- Метод информационного листа должен использовать полиморфизм, чтобы любой экземпляр объекта животного можно было присвоить ссылке базового класса.
- При работе ориентируйтесь на схему классов UML.



Главная программа может использовать классы животных так:

```
public static void main(String[] args) {

        Слон e = new слон("Слон");
        Носорог n = new носорог("Носорог");
        Дельфин d = new дельфин("Дельфин");
        Кит w = new кит("Кит");

        животное t;

        t = e;
        t.информационныйлист();
        System.out.println();

        t = n;
        t. информационныйлист ();
        System.out.println();

        t = d;
        t. информационныйлист ();
        System.out.println();

        t = w;
        t. информационныйлист ();
        System.out.println();
}
```

После запуска экран выглядит так:



8 Задания к главе «Массивы в Java»

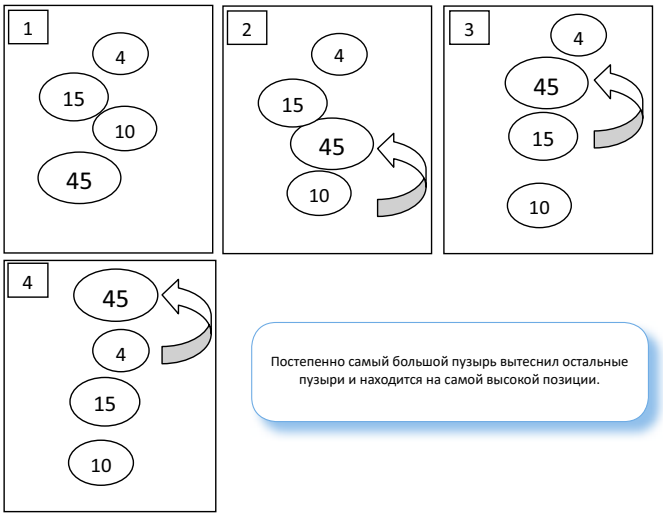
Задание 8.1

Напишите программу Java, которая считывает десять целых значений в массив и затем отображает сумму значений на экране.

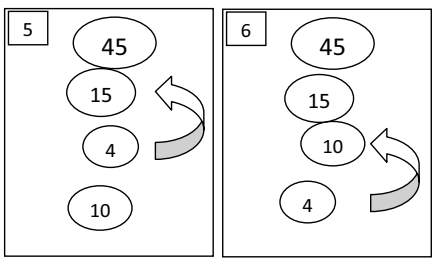
Задание 8.2

Простой алгоритм сортировки массивов – так называемая **Bubblesort**. Алгоритм был назван так, потому что элементы массива можно представить как пузыри (англ. bubbles) в стакане минеральной воды. Большие пузыри (элементы поля) поднимаются до тех пор, пока они не будут задержаны еще большими пузырями, которые, в свою очередь, поднимаются дальше.

На схеме ниже принцип представлен наглядно:



Теперь принцип повторяется с другими пузырями, так что второй по величине пузырь расположен на втором месте, третий на третьем, и четвертый на четвертом, последнем месте. Таким образом, массив полностью отсортирован.



Применение алгоритма в нескольких словах можно описать так:

Массив проходит первый этап от начала до конца. Первый элемент сравнивается и при необходимости меняется со следующим. Затем следующий элемент сравнивается и при необходимости меняется с дальнейшим. Это происходит до конца массива. Так наибольший элемент оказывается в конце массива – то есть наибольший пузырь поднимается вверх.

На следующих этапах применяется тот же принцип. Если массив имеет, например, четыре элемента, то эти этапы должны быть проведены три раза. В целом требуется (N-1) этапов при N элементах. Первый этап проходит до конца массива, второй идет только до предпоследнего элемента, так как наибольший элемент уже находится в конце и т. д.

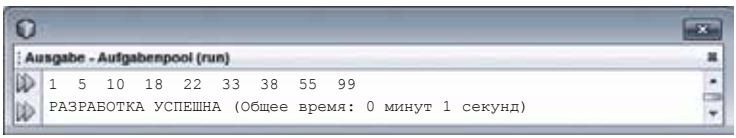
Задание:

Напишите статический метод Bubblesort, который сортирует массив целых значений любого размера по вышеописанному методу.

Главная программа может использовать сортировку Bubblesort следующим образом:

```
public static void main(String[] args) {
   int [] значения = {10, 55, 23, 18, 5, 99, 22, 33, 1, 38};
   bubblesort(значения);
   System.out.println("Сортировка Bubblesort");

   for (int i = 0; i < значения.length ; i++) {
        System.out.print(werte[i] + "     ");
   }
   System.out.println();
}
```

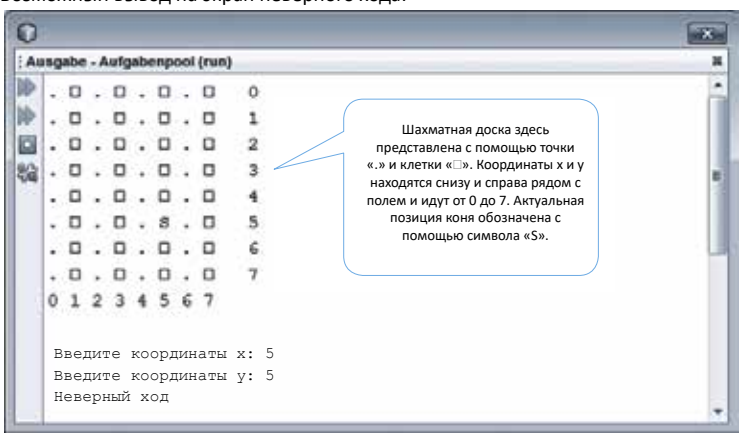
После запуска экран выглядит так:



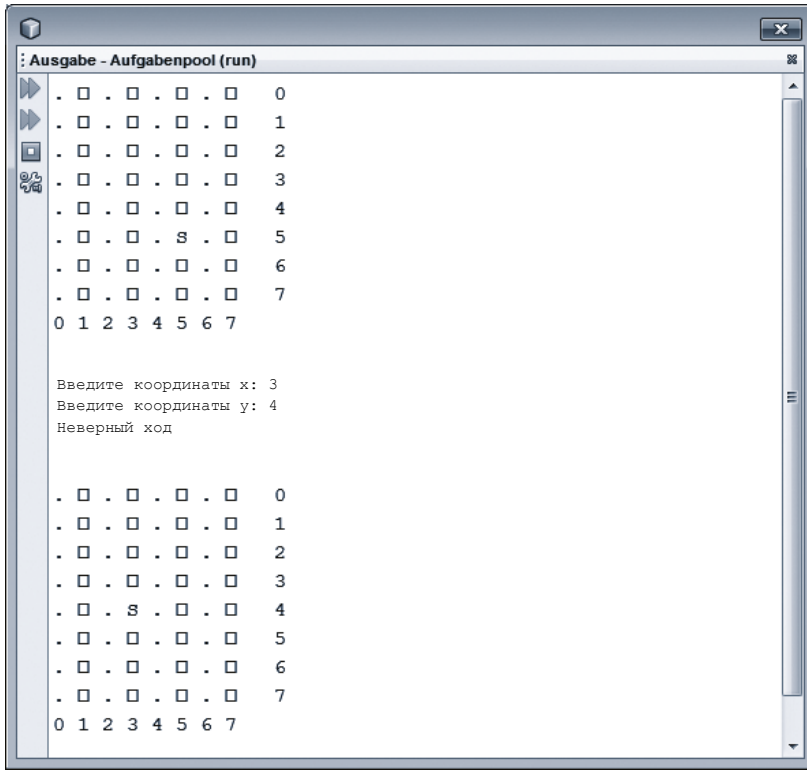
Задание 8.3

Напишите программу Java, «управляющую» шахматной доской с помощью массива. На этой шахматной доске стоит один конь. Пользователь может указать новые координаты для нового хода конем. Этот ход можно выполнить, только если он не нарушает правила шахматной игры. Программа должна выводить на экран шахматную доску и коня – при этом нужно использовать только очень простой вывод символов с помощью `System.out.println`. Для этого напишите класс `Шахматы`, который используется с требуемыми функциями с помощью соответствующих методов.

Возможный вывод на экран неверного хода:



Возможный вывод на экран верного хода:



Примечание:

Поиск возможных символов для представления шахматной доски можно осуществить с помощью следующего цикла:

```
for (int i=48;i < 255; i++) System.out.println(i + " " + (char)i);
```

Задание 8.4

В массиве было сохранено 100 целых значений из серии измерений в электротехнике. Для значений нужно определить различные статические данные. Для этого необходимо создать программу Java, которая имеет следующие функции:

- Расчет минимума измерительных значений
- Расчет максимума измерительных значений
- Расчет среднего значения измерительных значений
- Расчет диапазона разброса измерительных значений
- Расчет среднего отклонения измерительных значений
- Расчет пяти значений, которые встречаются чаще всего (список частотности значений)

Создайте для этого класс `Статистика`, который предлагает соответствующие методы для данных функций. Класс должен предложить меню выбора, из которого можно вызвать методы.

Пояснения к статическим данным:

► **Среднее значение:** Среднее значение это значение из середины массива. Массив должен быть предварительно отсортирован.

Пример:

```
int [] значения = { 3 , 7 , 2 , 9 , 1 };
сортировка: 1 2 3 7 9
среднее значение: 3
```

► Диапазон разброса: диапазон разброса ряда это расстояние между наименьшим и наибольшим элементом ряда.

► Среднее отклонение: среднее отклонение рассчитывается из суммы всех элементов массива за вычетом среднего значения каждого, и делится на количество элементов:

Пример:

```
int [] значения = { 3 , 7 , 2 };
среднее значение: ( 3 + 7 + 2 ) / 3 = 4
среднее откл.: ( | 3 – 4 | + | 7 – 4 | + | 2 – 4 | ) / 3 = ( 1 + 3 + 2 ) / 3 = 2
```

Сумма (положительное отклонение)

► **Частота:** число, указывающее, как часто элемент встречается в ряду.

Пример:

```
int [] значения = { 3, 7, 2, 3, 6, 2, 7, 3, 2, 3 };
```

Значение	2	3	7	6
Частота	3	4	2	1

Примечание:

Измерительные значения можно получить с помощью случайных чисел. В следующей строке программы можно создать случайные значения.

```
int случайноеЧисло = (int) (Math.random() * 100);
```

Метод `random` дает случайное неотрицательное целое число от 0 (вкл.) до 1 (искл.). После умножения на 100 и преобразования в целое число случайное значение находится между 0 и 99.

Задание 8.5

В математике и экономике матрицы являются важной темой. Для учета матриц необходимо разработать класс `Matrix`, который может сохранять матрицу любого размера.

Пример: матрица 3 x 3:

$$Matrix = \begin{bmatrix} 1 & 7 & 2 \\ 3 & 2 & 2 \\ 5 & 6 & 3 \end{bmatrix}$$

Сохранение матрицы должно осуществляться с помощью соответствующего массива. При этом должны применяться следующие функции в классе:

Метод ввода():

Пользователь может указать размер матрицы, далее устанавливаются размеры массива и считываются значения.

Метод вывода():

Матрица (отформатированная) выводится на экран.

Метод транспонирования():

Матрица транспонируется. Это значит, что все значения матрицы `aij` меняют свои индексы (`aij = aji`).

Пример транспонирования:

$$\begin{bmatrix}1 & 7 & 2\\ 3 & 2 & 2\\ 5 & 6 & 3\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 3 & 5\\ 7 & 2 & 6\\ 2 & 2 & 3\end{bmatrix}$$

Складывание статических методов ():

Две переданные матрицы складываются и результат возвращается.

Умножение статических методов ():

Две переданные матрицы умножаются и результат возвращается.

Пояснения к складыванию и умножению:

Две матрицы складываются, каждый элемент матрицы складывается с элементом с тем же индексом другой матрицы.

Пример:

$$\begin{bmatrix}1 & 7 & 2\\ 3 & 2 & 2\\ 5 & 6 & 3\end{bmatrix} + \begin{bmatrix}2 & 6 & 5\\ 7 & 1 & 6\\ 2 & 2 & 7\end{bmatrix} = \begin{bmatrix}1+2 & 7+6 & 2+5\\ 3+7 & 2+1 & 2+6\\ 5+2 & 6+2 & 3+7\end{bmatrix} = \begin{bmatrix}3 & 13 & 7\\ 10 & 3 & 8\\ 7 & 8 & 10\end{bmatrix}$$

Умножение немного сложнее:

Первая строка первой матрицы умножается на первый столбец второй матрицы по элементам, и затем произведения суммируются. Конечный результат это первый элемент матрицы умножения.

Далее первая строка первой матрицы умножается на второй столбец второй матрицы по элементам, и затем произведения суммируются. Конечный результат это второй элемент матрицы умножения и т. д.

Пример:

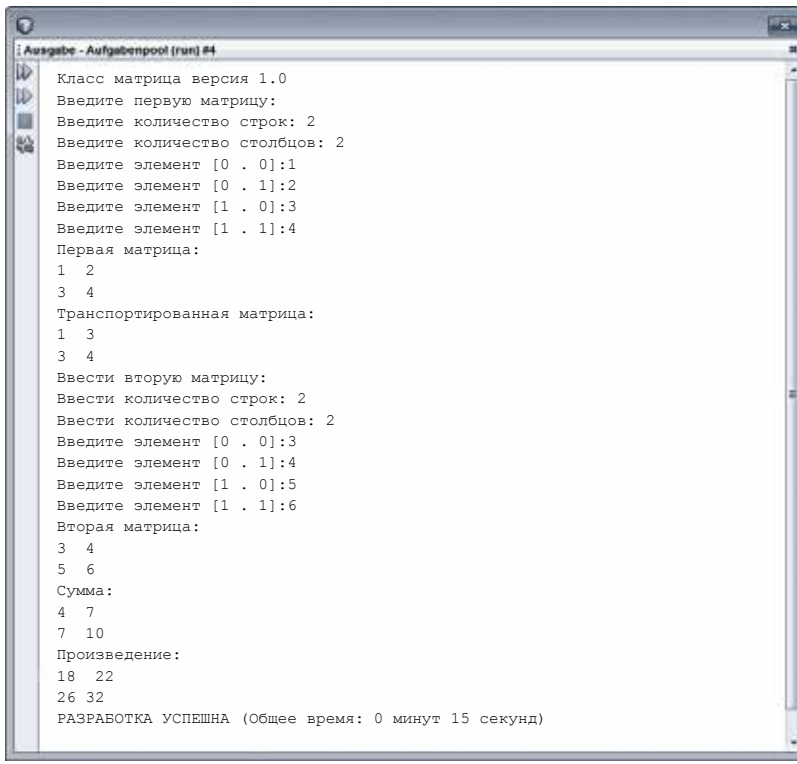
$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \bullet \begin{bmatrix}3 & 4\\ 5 & 6\end{bmatrix} = \begin{bmatrix}1\cdot 3+2\cdot 5 & 1\cdot 4+2\cdot 6\\ 3\cdot 3+4\cdot 5 & 3\cdot 4+4\cdot 6\end{bmatrix} = \begin{bmatrix}13 & 16\\ 29 & 36\end{bmatrix}$$

На следующем примере показано использование класса `Матрица` в главной программе:

```
public static void main(String[] args) {
      Matrix a = new Matrix ();
      Matrix b = new Matrix ();
      Matrix c = new Matrix ();
      System.out.println("Класс Матрица версия 1.0");
      System.out.println("Ввести первую матрицу:");
      a.ввод();
      System.out.println("Первая матрица:");
```

```
        a.вывод();
        a.транспонирование();
        System.out.println("Транспонированная матрица:");
        a.вывод();
        System.out.println("Ввести вторую матрицу:");
        b.ввод();
        System.out.println("Вторая матрица:");
        b.вывод();
        c = матрица.сложение(a, b);
        System.out.println("Сумма:");
        c.ввод();
        c = матрица.умножение(a, b);
        System.out.println("Произведение:");
        c.вывод();
    }
```

После запуска экран выглядит так:

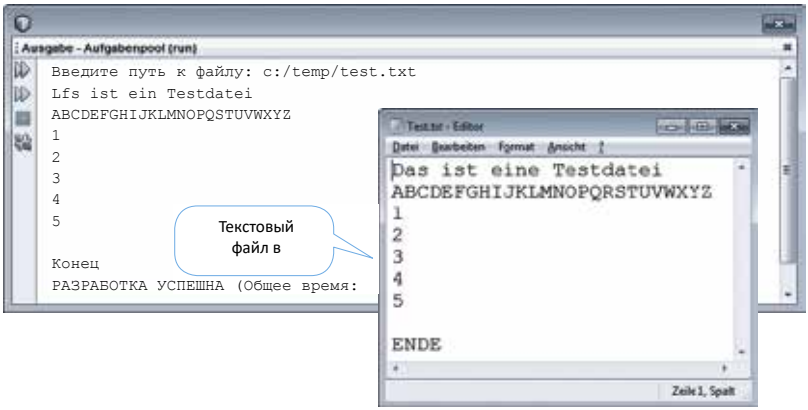


9 Задания к главе «Файловые операции в Java»

Задание 9.1

Напишите программу Java, которая считывает и отображает на экране текстовый файл. Для этого пользователь должен указать имя файла, затем производится вывод на экран.

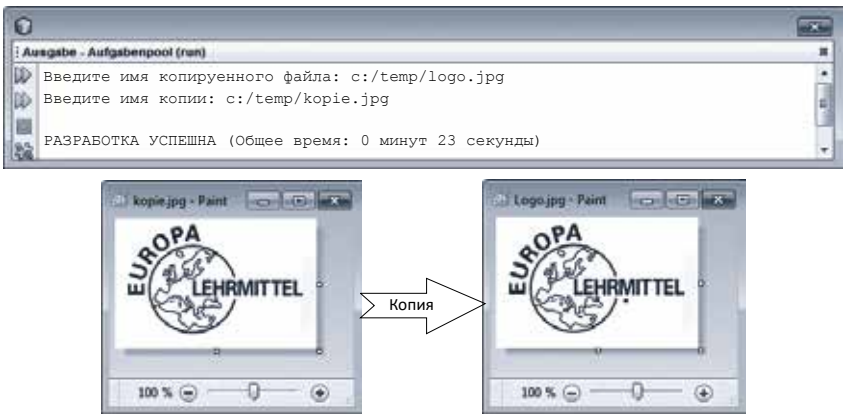
После запуска программа может выглядеть так:



Задание 9.2

Напишите программу Java, которая создает копию любого файла. Для этого пользователь должен указать имя файла, который необходимо скопировать и имя копии.

После запуска программа может выглядеть так:



Примечание:

Бинарное чтение и запись может производиться с помощью классов `FileInputStream` и `FileOutputStream`:

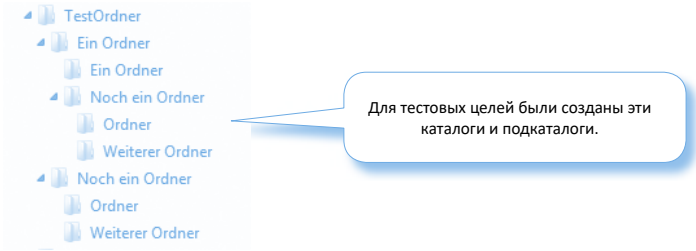
```
FileInputStream в = new FileInputStream("Имя исходного файла");
FileOutputStream из = new FileOutputStream("Имя целевого файла");

byte [] b = new byte[1];

чтение.read(b);

запись.write(b);
```

Одноэлементный массив для бинарного чтения

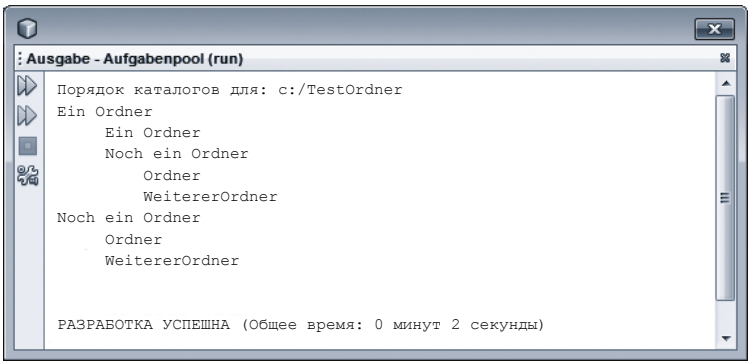
Имя файла с указанием пути

Задание 9.3

Напишите программу Java, которая считывает путь пользователя и затем приводит список всех каталогов и подкаталогов. Для этого используйте метод `listFiles` класса `File`.



После запуска программа может выглядеть так:



Примечание:

Для считывания каталогов предлагается так называемый ***рекурсивный метод***. Такой метод вызывается снова и снова, но с измененными параметрами, иначе это был бы лишь вид бесконечного цикла. «Фишка» *рекурсии* в выборе подходящего передаваемого параметра, который отвечает за то, чтобы *рекурсия* проходила под контролем (то есть, с концом).

Пример 1:

Рекурсия, проходящая без контроля:

```
public static void вывод() {

        System.out.print("Привет");
        вывод();
}
```

Рекурсивный вызов

Метод вызывается постоянно (бесконечно). Так программа рано или поздно вылетает из-за ошибки переполнения стека (`StackOverflow`).

Пример 2:

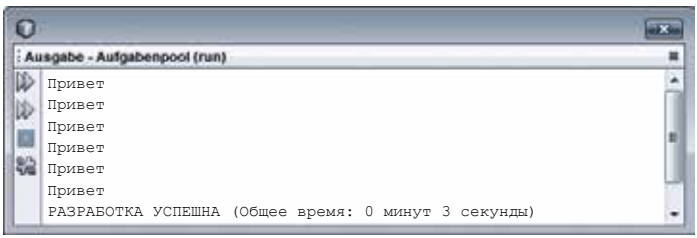
Рекурсия, проходящая под контролем:

Параметр управления

```
public static void вывод(цел число){

  if (счетчик < 5) {
     System.out.println("Привет");
     вывод2(счетчик + 1);
  }
}
```

Вызов с измененным параметром

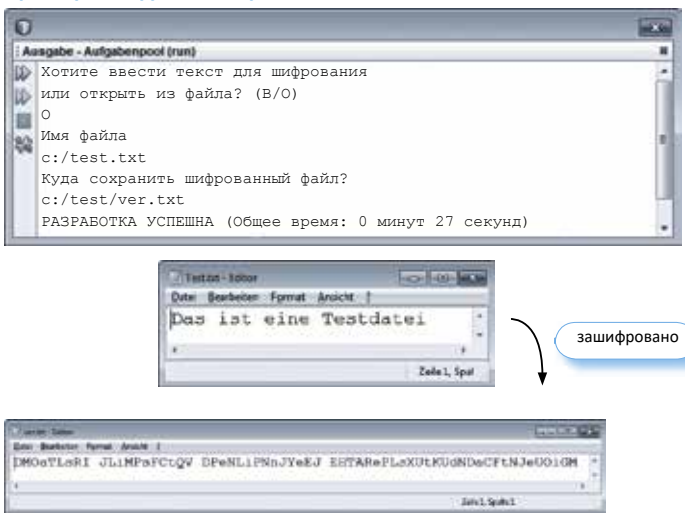
После запуска программы с `вывод 2 (0)` (стартовое значение 0) ***рекурсия*** теперь проходит под контролем:



Задание 9.4

Напишите программу Java, которая может производить простое шифрование цепей символов и текстовых файлов. Для этого пользователю будет задан вопрос, хочет ли он ввести текст, который следует зашифровать, или для этого нужно считать текстовый файл. Далее текст шифруется и записывается в указанный файл. Шифрование относительно простое: после каждого символа текста сначала всегда пишутся два случайных символа. Таким образом, только каждый третий символ зашифрованного файла принадлежит оригинальному тексту. Кроме шифрования должна быть предложена соответствующая расшифровка.

Пример: Шифрование файла



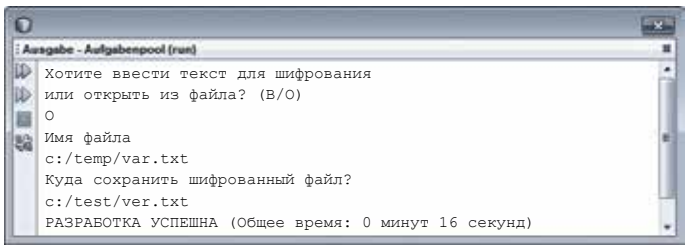
Примечание:

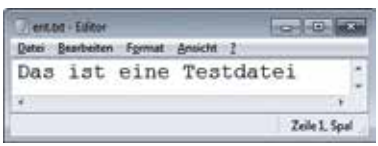
С помощью следующей программной строки можно получить случайные значения.

```
int случайноечисло = (int) (Math.random() * 100);
```

Метод `random` дает случайное неотрицательное целое число от 0 (вкл.) и 1 (искл.). После умножения на 100 и преобразования в целое число случайное значение находится между 0 и 99.

Пример: Расшифровка файла





Расшифровано

Задание 9.5

Создайте класс `Слова` в Java, который будет представлять и вызывать пары слов (немецкий – английский и немецкий – испанский). Класс должен сохранять слова в двух `Hashmaps`. Слова должны считываться в конструкторе класса из файла «`Vokabel.txt`» (см. примечания).

Для обработки слов должны быть доступны следующие методы:

- `Вывод слов`

```
public void вывод()
```

Метод отображает все слова отформатированными на экране.

- `Вызов слов`

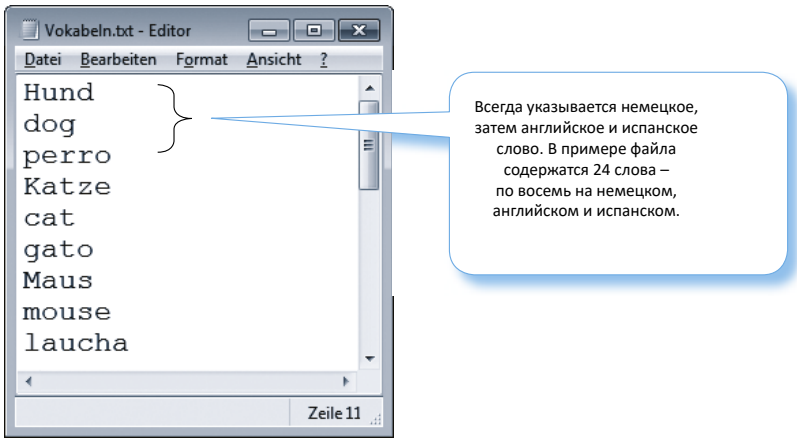
```
public int вызов (String язык)
```

Запускается вызов слов. В зависимости от передаваемого параметра вызов немецкий – английский или немецкий – испанский. Далее вызов слов должен производиться с помощью случайного генератора. Здесь считаются и возвращаются правильные ответы. Каждое слово может или должно быть вызвано только один раз.

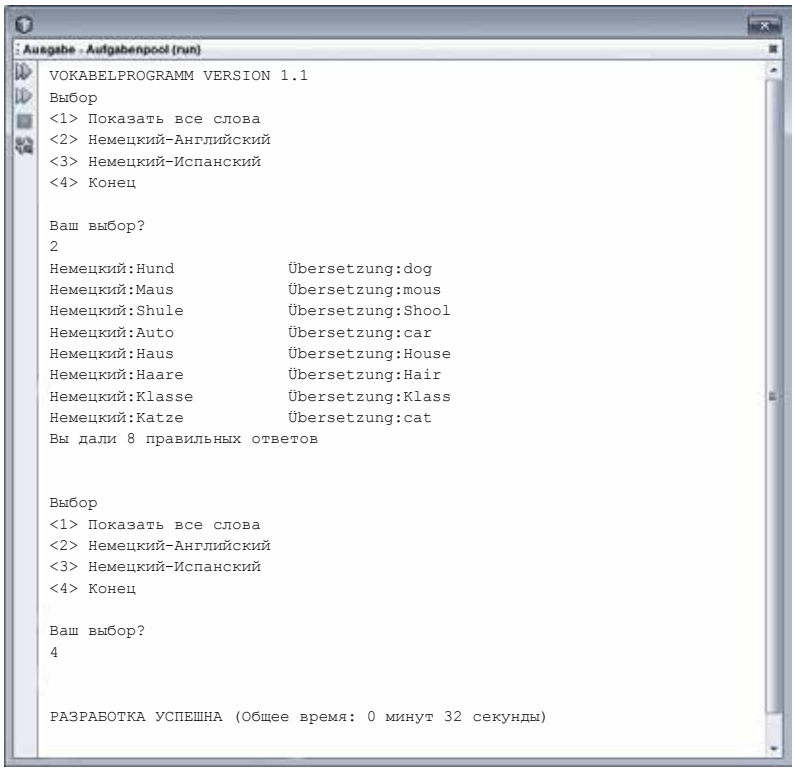
Дополнительно напишите меню выбора в главном меню. Пользователь должен выбрать, будут ли вызваны слова на немецком – английском или немецком – испанском.

Примечания:

- Для получения случайных значений можно использовать метод `random` класса `Math` (см. также задание 9.4)
- Файл «Слова.txt» выглядит так:



После запуска вызов может выглядеть так:



Задание 9.6

Напишите класс `РядИспытаний`, который может принимать любое количество целочисленных положительных измерительных значений в `ArrayList`. Далее создайте интерфейс сериализации и протестируйте успешную сериализацию и десериализацию с помощью следующей копии исходного кода:

```
:
рядиспытаний испытание = new рядиспытаний ();
испытание.ввод();
испытание.вывод();

System.out.println("Объект ряда испытаний сериализуется!");
ObjectOutputStream сериализовать =
  new ObjectOutputStream(
    new FileOutputStream("C:/temp/ИзмерительныеЗначения.dat"));

сериализовать.writeObject(испытание);
сериализовать.close();

System.out.println("Десериализация в другой объект
                  ряда испытаний!");

ObjectInputStream десериализовать =
    new ObjectInputStream(
     new FileInputStream("C:/temp/ИзмерительныеЗначения.dat"));

РядИспытаний НовоеИспытание;
НовоеИспытание = (РядИспытаний) десериализовать.readObject();
НовоеИспытание.вывод();

десериализовать.close();
:
```

Примечание:

Метод `ввод` класса `РядИспытаний` считывает любое количество значений и добавляет их к `ArrayList`. Метод `вывод` выводит значения на экран.

10 Задания к главе „Темы Java для продвинутого уровня“

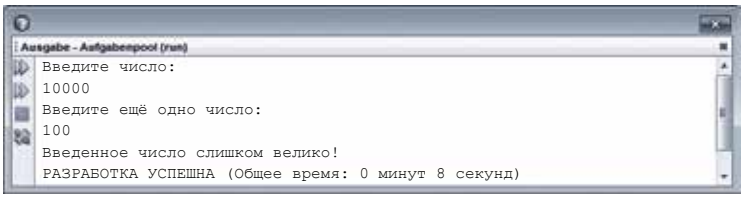
Задание 10.1

Напишите программу Java, которая считывает два числа и затем складывает, вычитает, делит и умножает. Для этого необходимо написать соответствующие методы в классе. Подумайте, какие виды ошибок (ошибка расчета, ошибки ввода, ошибки переполнения буфера…) могут возникнуть, и напишите соответствующие блоки `try` и `catch`. Исполните обработку исключений на различных уровнях (в самом методе, при вызове метода и т. д.) и проанализируйте процесс.

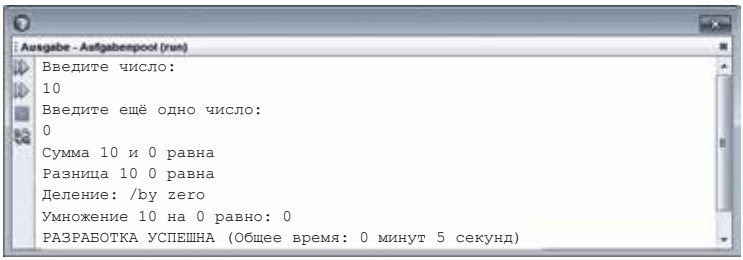
Дополнение:

Введенные числа должны находиться в диапазоне от –9999 до 9999. Исполните для этого собственный Exception-класс, который в случае ошибки ввода будет использован для обработки исключений.

После запуска экран может выглядеть так:



Или так:



Задание 10.2

Напишите обобщенный класс `Liste`. Этот класс должен уметь сохранять любое количество значений типа `T`. Кроме конструктора следует создать методы добавления, удаления и отображения. Дополнительно необходимо создать атрибут количества. С помощью соответствующего метода `get` должно быть возвращено актуальное количество элементов.

В методе `main` класс должен использоваться следующим образом:

```
public static void main(String[] args) {

    список<целое> intListe = new список<целое>();

    System.out.println("Список целых чисел:");

    intListe.добавить(10);
    intListe.добавить(20);
    intListe.добавить(30);
    System.out.println("Количество элементов:
                       " + intListe.getAnzahl());

    intListe.отобразить();
    System.out.println();
    System.out.println("Удаление элемента 2......");
    intListe.удалить(1);

    System.out.println("Количество элементов:
                " + intListe.getAnzahl()); intListe.отобразить();
    System.out.println();
}
```

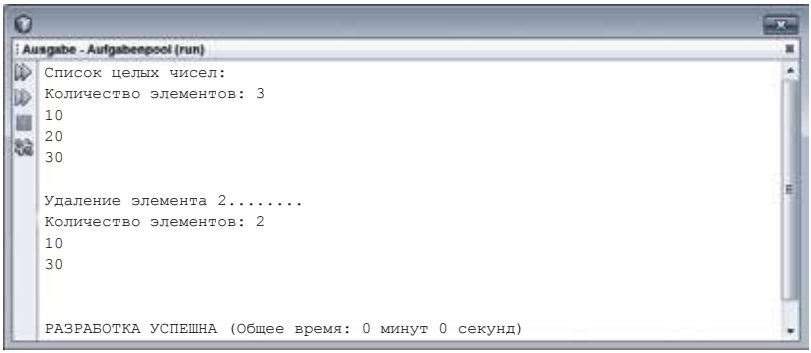
Инстанцировать список для целых значений

Примечание:

Создание обобщенного массива в Java может производиться только с помощью особого ***приема***. Создается массив `Object` и затем сразу преобразуется в обобщенный тип `T`:

```
T[] список = (T[]) new Object[100];
```

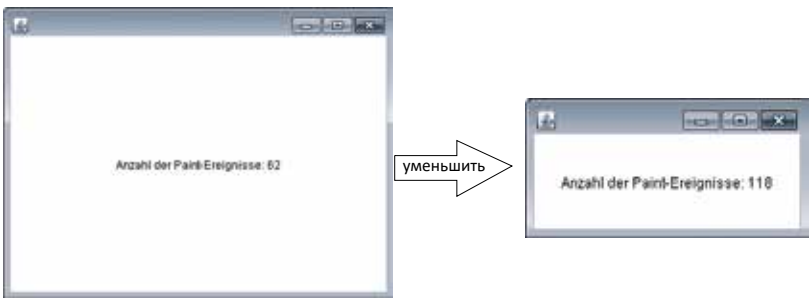
После запуска вывод на экран выглядит так:



11 Задания к главе „GUI-программирование с помощью AWT“

Задание 11.1

Наследуйте собственный класс от класса `Frame`, исполните клиентскую область и перезапишите метод `paint`. Затем создайте счетчик, который считает количество событий Paint. Далее в окне должно отображаться их актуальное количество. При этом текст должен быть отцентрирован как вертикально, так и горизонтально, как показано на скриншотах программы:



Примечание к использованию:

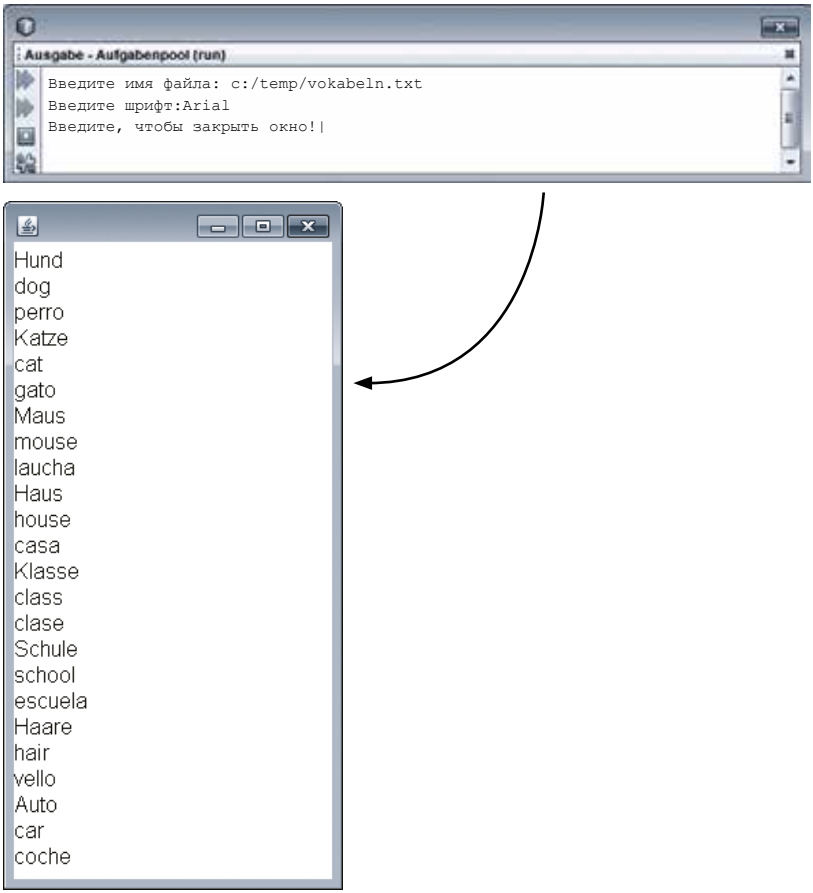
Высоту и ширину строки текста можно определить с помощью следующих операторов:

```
int высоташрифта = this.getFont().getSize();

FontMetrics fm = g.getFontMetrics(this.getFont()); i

int ширинатекста = fm.stringWidth("текст");
```

Задание 11.2

Напишите программу GUI, которая считывает и отображает текстовый файл. При запуске программы с консоли сначала вводится имя файла и желаемый шрифт, затем запускается окно, которое отображает текстовый файл по строкам.

После запуска программа может выглядеть так:



Задание 11.3

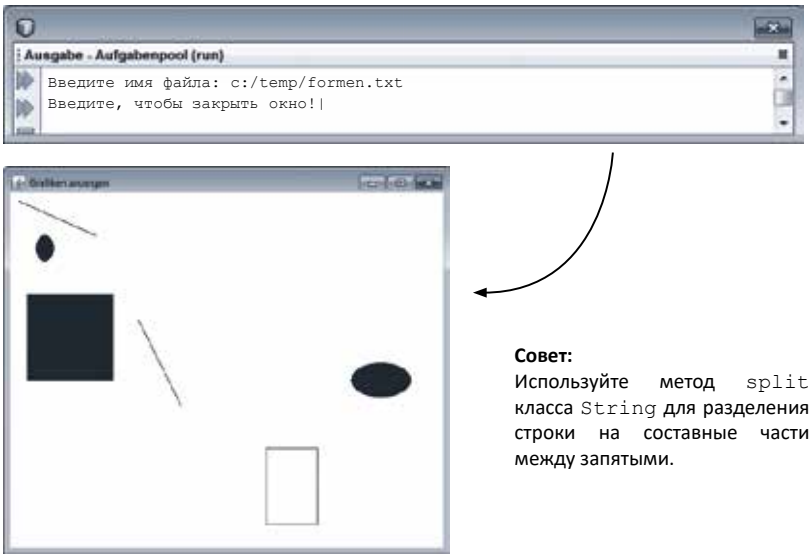
В текстовом файле (см. структура файла) сохранены геометрические формы. Эти формы (линия, эллипс, прямоугольник) должны быть считаны и затем графически представлены в клиентской области окна.

Структура текстового файла:

```
ЛИНИЯ,10,10,100,50,ЧЕРНЫЙ
ПРЯМОУГОЛЬНИК,20,120,100,100,ЧЕРНЫЙ,
ЭЛЛИПС,30,50,20,30,СИНИЙ,СИНИЙ
ЛИНИЯ,150,150,200,250,ЧЕРНЫЙ
ПРЯМОУГОЛЬНИК,300,300,60,90,СИНИЙ
ЭЛЛИПС,400,200,70,40,КРАСНЫЙ,СИНИЙ
```

У линии заданы начальная и конечная точка. У прямоугольника и эллипса задана начальная точка, ширина и высота. Первый цвет - цвет линии, второй - цвет заливки (опционально).

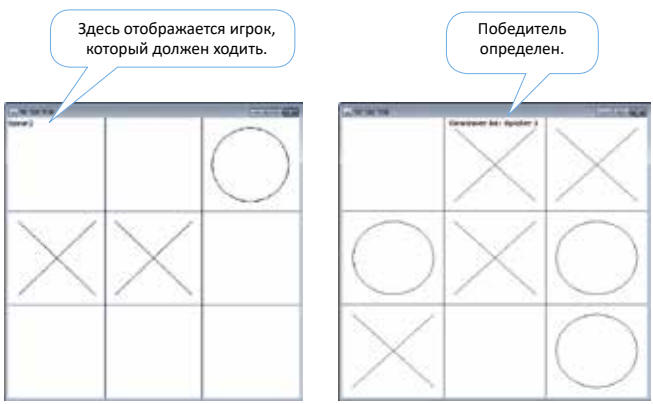
Сохраните вышеуказанные строки в тестовом файле на жестком диске. При запуске программы этот файл должен быть считан, и должны отобразиться соответствующие графические элементы. После запуска программа может отобразить вышеуказанный файл следующим образом:



Совет:
Используйте метод `split` класса `String` для разделения строки на составные части между запятыми.

Задание 11.4

Создайте проект программы GUI, в которой реализуется известная игра ***Крестики-Нолики***. Игра должна быть рассчитана в одном варианте для двух игроков, которые ходят по очереди. Для этого первый игрок щелкает по одному из девяти полей и, таким образом, отмечает это поле крестиком. Затем второй игрок щелкает по пустому полю и отмечает его ноликом. Как только три крестика или нолика будут расположены в одном ряду, столбце или по диагонали, соответствующий игрок выигрывает. Программа с помощью текста показывает, чей ход, и проверяет победителя. На скриншотах ниже представлена возможная реализация программы.



12 Задания к главе „Элементы управления с помощью AWT или классов Swing“

Задание 12.1

Напишите программу GUI, которая реализует простой карманный калькулятор. Карманный калькулятор должен предлагать основные арифметические операции и дополнительную кнопку (кнопка «C») для удаления информации. Калькулятор может выглядеть так:

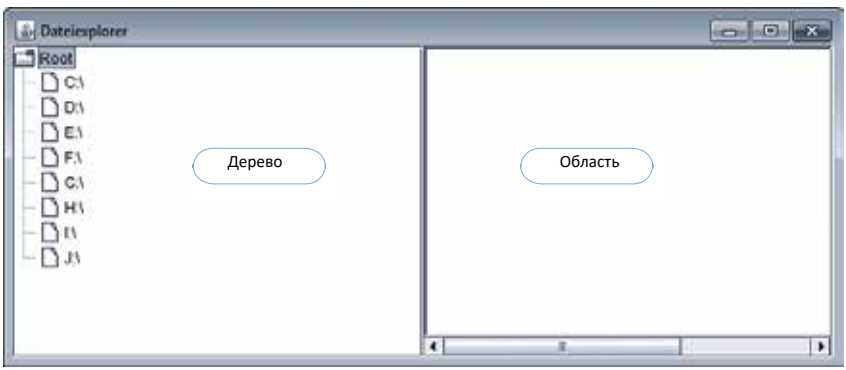
На изображениях выше представлено использование калькулятора: в текстовом поле вводится значение и далее нажимается кнопка операции (в этом случае кнопка «+»). После ввода второго значения результат (10 + 20 = 30) после нажатия кнопки «=» будет отображен в текстовом поле.

Примечание:

Значения должны быть преобразованы из текстового поля с помощью метода преобразования в желаемый тип данных (например, `double`).

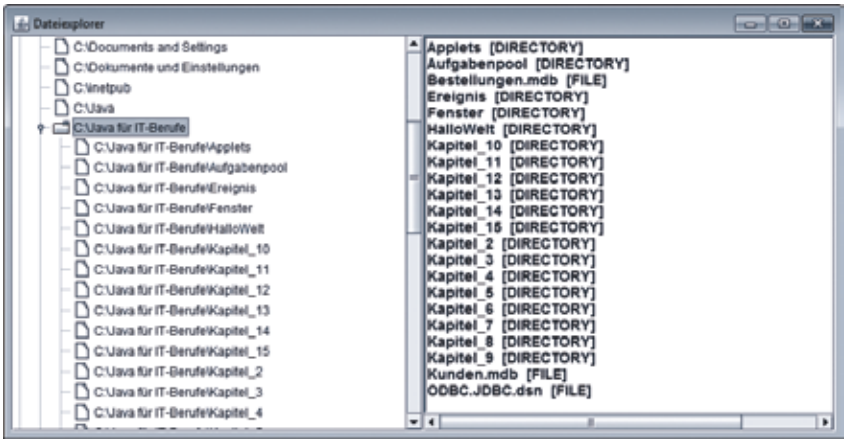
Задание 12.2

Напишите свой собственный проводник. Как в проводнике Windows или другом проводнике в дереве должны быть представлены папки, а также данные текста, которые можно найти в выбранной папке дерева. Программа может выглядеть так:



После запуска считываются все доступные накопители и отображаются в дереве. Кроме дерева создан второй элемент управления (`TextArea`). Этот элемент управления служит для приема детализированной информации об отдельных каталогах. Оба элемента управления имеют полосу прокрутки.

Щелчок по каталогу отображает подкаталоги и параллельно выводит списки всех данных и каталогов в области данных:



Примечания:

- Дерево может быть заполнено двумя способами:

Рекурсия:

Весь каталог (накопитель) просматривается рекурсивно, и создаются все узлы. *Недостаток:* При больших каталогах (накопителях) проходит очень медленно, и существует опасность *StackOverfows*.

Заполнение по требованию:

Только при событии (например, щелчок по папке) считываются каталоги папки и добавляются в структуру дерева. На следующем примере показано, как такое событие можно исполнить:

```
дерево.addTreeSelectionListener(
  new TreeSelectionListener(){
    @Override
    public void valueChanged(TreeSelectionEvent e)

    DefaultMutableTreeNode узлы =
    (DefaultMutableTreeNode)дерево.getLastSelectedPathComponent();
    //TODO: заполнить узлы
    :
    дерево.expandPath(e.getNewLeadSelectionPath());
}
});
```

Выбрать селектированный узел

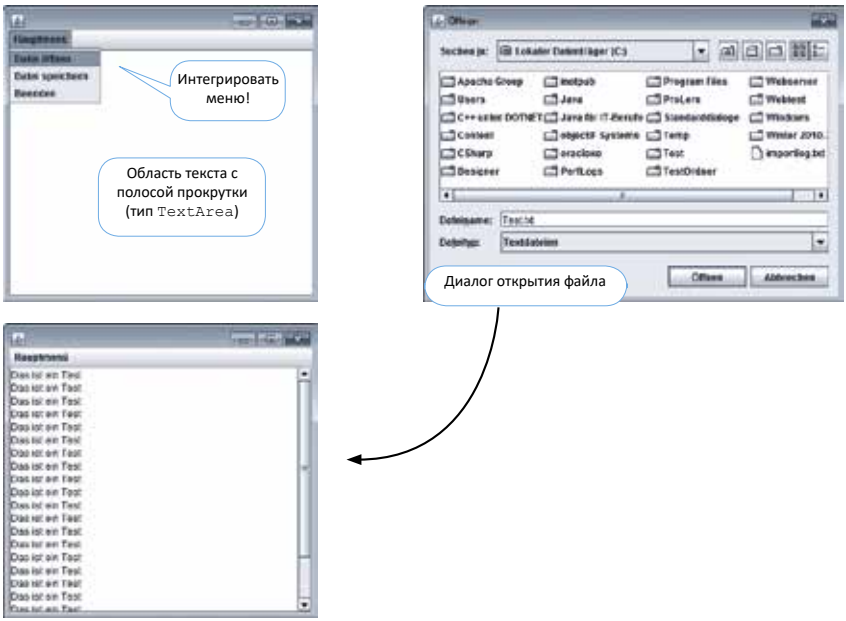
Расширить заполненный узел

13 Задания к главе „Меню, диалоги и апплеты“

Задание 13.1

Напишите программу Java-GUI, которая реализует простое приложение-редактор. Должна быть возможность загружать, редактировать и сохранять текстовые файлы. Меню должно предлагать соответствующие функции. Открытие и сохранение должно быть реализовано с помощью соответствующего диалога (например, `JFileChooser`).

Приложение может выглядеть так:



Примечание:

Диалог файла реализуется с помощью класса `JFileChooser`. На следующем примере показано, как можно считать выбранный файл пользователя:

```
JFileChooser файлдиалог = new JFileChooser("*.txt");
dateiDialog.setFileFilter(
        new FileNameExtensionFilter("ТекстовыеФайлы","txt"));
dateiDialog.showOpenDialog(this);
```

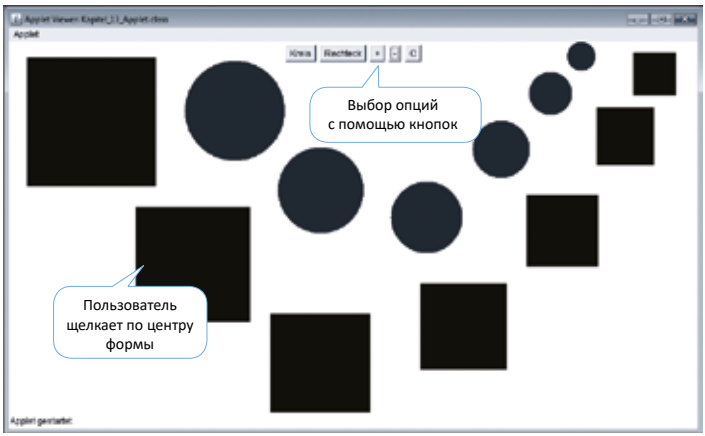
Установить фильтр файла

```
String файл = файлдиалог.getSelectedFile().getPath();
```

Выбранный файл

Задание 13.2

Разработайте апплет Java, который позволяет графически изображать круги и прямоугольники в клиентской области. Для этого пользователь щелкает мышкой в клиентской области и затем изображается одна из форм. Круги изображаются синим цветом, прямоугольники – черным. С помощью кнопки пользователь может выбрать, какую форму изобразить. Кнопка «плюс» увеличивает следующую фигуру, которую нужно изобразить (наоборот – кнопка «минус»). Кнопка с (`clear`) удаляет все формы. Формы должны быть сохранены в списке (например, `ArrayList`), чтобы метод `paint` всегда мог корректно представить все содержание апплета.



Примечание:

Создайте проект классов для форм, все из которых произведены от одного абстрактного базового класса. Так упрощается сохранение форм в `ArrayList`. Классы должны иметь методы, которые выводят форму на экран. Эти методы созданы в абстрактном базовом классе и перезаписываются в производных классах.

14 Задания к главе «GUI-конструктор NetBeans»

Задание 14.1

Напишите приложение Java, которое реализует игру ***Крестики-Нолики*** из задания 11.4 с помощью GUI-конструктора. Для этого создайте девять кнопок и присвойте каждой кнопке приемник события. Логику игры, в принципе, можно перенять из задания 11.4. Крестики и нолики теперь просто представлены шрифтом соответствующего размера как название кнопки («X» или «O»).

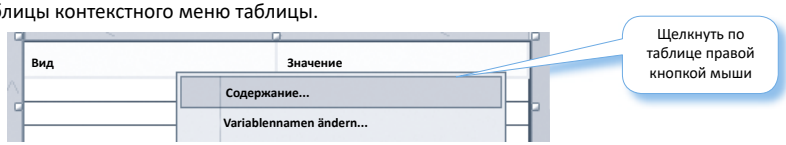
Задание 14.2

Для операторов клиентской поддержки *Банк 42* необходимо разработать приложение, с помощью которого легко можно рассчитать потребительский кредит. С помощью ползунковых регуляторов (`slider`) необходимо определить основные данные кредита. В расположенной ниже таблице параллельно представлены эти данные. Также сразу отображается ежемесячный платеж.

Примечания:

► С помощью атрибутов `минимум` и `максимум` можно установить интервалы. Важно ограничить интервал и умножить значение на соответствующий множитель. Например, интервал ползунка суммы кредита установить на 2 – 100, а значение для расчета умножить на 500.

► Столбцы и строки таблицы удобно создать с помощью пункта меню содержание таблицы контекстного меню таблицы.



Доступ к значениям таблицы происходит в исходном коде с помощью следующего метода:

```
jTable.setValueAt("Значение", 0, 0);
```

Строка и столбец таблицы (ВНИМАНИЕ: начинается с нуля)

Формула расчета ежемесячного платежа выглядит так:

$$\text{Выплата} = \text{Сумма кредита} \cdot \frac{\frac{\%\text{ставка}}{1200} \cdot \left(1+\frac{\%\text{ставка}}{1200}\right)^{\text{Срок}}}{\left(1+\frac{\%\text{ставка}}{1200}\right)^{\text{Срок}} - 1}$$

Формула соответствует формуле ежегодного платежа, пересчитанной на месяцы.

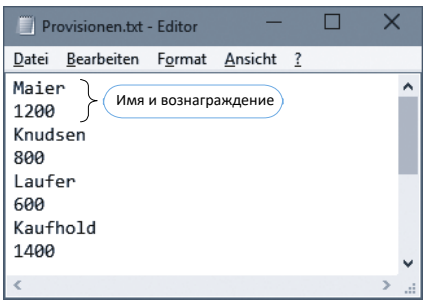
15 Задания к главе „Соединение с базой данных“

Задание 15.1

В компании комиссионное вознаграждение торговых представителей сохраняется в текстовом файле. Напишите простое консольное приложение, которое считывает текстовый файл и сохраняет в таблице базы данных. Таблица базы данных должна быть создана заранее в соответствующей базе данных (например, SQLite) с соответствующими командами SQL. Далее из таблицы необходимо считать следующие статические данные:

- торговый представитель с наибольшим вознаграждением
- торговый представитель с наименьшим вознаграждением
- сумма всех вознаграждений
- среднее вознаграждение

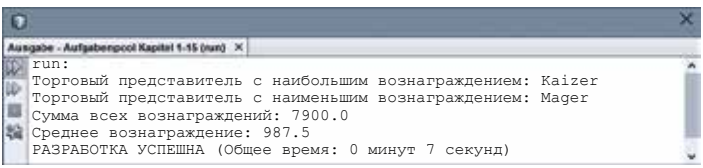
Текстовый файл выглядит так:



Примечание:

- Расчет данных можно производить с помощью соответствующих функций SQL (как СУМ, СРЗН, МИН и МАХ), или непосредственно в программе Java.

После запуска экран может выглядеть так:



Задание 15.2

Исходная ситуация:

В компании данные заказов клиента хранятся в двух таблицах базы данных (например, SQLite). Для сотрудников необходимо написать простую программу приложения Java-GUI, с помощью которой данные заказов клиента можно представить наглядно.

Основные таблицы выглядят так:

Таблица клиентов:

ID	Имя
1	Майер
2	Кнудсен
3	Кайзер
4	Франсен
5	Кноблох

Связь таблиц:

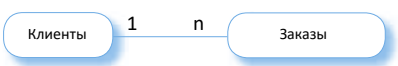


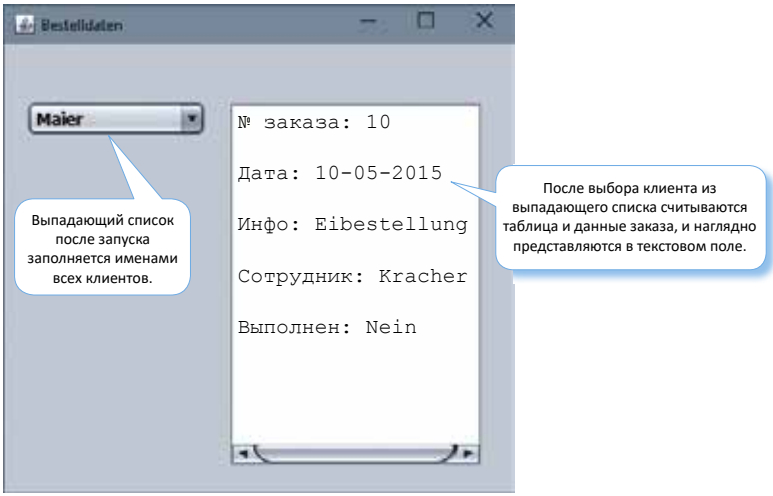
Таблица заказов:

ID Клиента	№ заказа	Дата	Инфо	Сотрудник	Выполнен
1	10	10.05.2015	Eilbestellung	Kracher	true
3	11	10.06.2015	gefährliche Fracht	Klauber	true
4	12	10.07.2015	guter Kunde	Hütter	false

Таблица заказов имеет внешний ключ ID клиента, который реализует связь «1:n» – обеих таблиц.

Задание:

Создайте обе таблицы в соответствующей базе данных (например, SQLite) и заполните таблицы соответствующими значениями. Затем создайте приложение Java, которое имеет доступ к базе данных и считывает таблицы. Оболочка приложения должна выглядеть так:



16 Задания к главе «Разработка приложений Android»

Задание 16.1

Для занятий по математике необходимо разработать приложение, с помощью которого можно определить форму вершины кривой квадратичной функции. Для этого пользователь может указать условия функции, и приложение определит вершину кривой. Приложение может выглядеть так:

Ввод условий функции в форме:
ax^2 + bx + c
Примеры:
5x^2+3x-2 (соответствует: $5x^2+3x-2$)
2x^2+0x+5 (соответствует: $2x^2+5$)
x^2+2x+0 (соответствует: x^2+2x)

Кнопка «рассчитать»

Вывод вершины кривой

Примечания:

► Размер шрифта (также вид или цвет) текстового вывода можно определить в файле main.xml:

```
<TextView
:
android:textSize="12pt"
/>
```

►Преобразование `String s` в `double d` может происходить так:

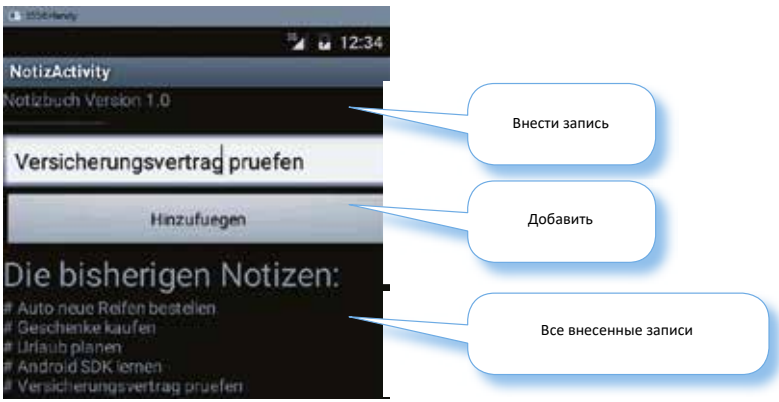
```
double d =        Double.parseDouble(s);
```

►С помощью метода `Split` строкового типа ввод в параметры может быть распределен:

```
Строковой тип [] части = функция.split("x");
```

Задание 16.2

Напишите прикладную программу Android, которая представляет своего рода записную книжку. Пользователь может добавлять любое количество записей (короткие тексты), которые затем сохраняются в базе данных SQLite. Все внесенные записи должны отображаться в хронологическом порядке в нижней области приложения.



Примечание:

TextView можно снабдить полосой прокрутки, если записей больше, чем может быть показано на дисплее:

```
<TextView android:layout_width="match_parent"
android:layout_height="match_parent"
android:scrollbars = "vertical"
/>
```

Затем в исходном тексте дополнительно следует создать элемент управления «с прокруткой»:

```
tView.setMovementMethod(new ScrollingMovementMethod());
```

Часть 3
Учебные ситуации

Учебная ситуация 1:
Презентация с вводной информацией о языке Java
(на русском или английском) 266

Учебная ситуация 2
Подготовка клиентской документации для реализации среды разработки в Java
(на русском или английском) 269

Учебная ситуация 3:
Разработка шифра для внутренней системы памяти
отдела поддержки сетевой компании 269

Учебная ситуация 4:
Планирование, внедрение и анализ электронной анкеты 270

Учебная ситуация 5:
Разработка программного обеспечения для представления
метеорологических данных с помощью схемы
«Модель – Вид – Контроллер» 273

Учебная ситуация 6:
Разработка приложения для решения головоломки «Судоку» 276

Учебная ситуация 1:

Презентация с вводной информацией о языке Java (на русском или английском)

Исходная ситуация:

Вы начали учиться на специалиста по програмированию в небольшой компании по разработке ПО `ProSource`. Также компания проводит курсы в разных сферах IT.

Запланировано обучение по языку программирования Java. В связи с тем, что Java-разработчики компании очень заняты, подготовка обучения проблематична. Поэтому Вы получаете задание по организации вводного информационного блока обучения. Эта часть должна занять примерно 15 минут. Кроме исторических данных необходимо представить интересные аспекты языка – например, расположение языка Java среди структурированных и объектно-ориентированных языков. Презентацию необходимо использовать также в иностранных представительствах компании, поэтому она должна быть продублирована на английском языке.

Работа индивидуально и в паре:

Планирование:

Определите средства презентации (презентация PowerPoint, раздаточный материал и т. д.). Подумайте над объемом презентации (временные рамки – 15 минут).

Найдите информацию об основах Java в информационном блоке данной книги и других источниках в Интернет.

Если возможно, поработайте над презентацией также на занятиях по английскому языку.

Выполнение:

Сделайте привлекательное оформление слайдов, не перегружая их. Формулируйте тексты слайдов кратко и емко. Ведите презентацию на русском или английском языке, если можно организовать междисциплинарное занятие.

Контроль:

Проведите презентацию для Вашего партнера или другой учебной группы. Партнер или слушатель оценивает презентацию с учетом приведенного ниже списка критериев, который потом может стать основой для критического анализа.

Каталог критериев для оценивания презентации

Аспекты содержания:

- Компоненты/структура доклада
- Логическое оформление доклада
- Соответствующее использование профессионального языка
- Четкое объяснение взаимосвязей,
- Подведение итогов (выводы)
- Четкое выделение частей доклада

Личные аспекты:

- Спокойная четкая речь/произношение
- Выдерживание пауз
- Движения и жесты
- Зрительный контакт со слушателями

Учебные цели:

► Ознакомление с важными аспектами языка Java.
► Получение опыта в составлении и проведении профессиональной презентации в сфере IT.
► Улучшение навыков английского языка и знакомство с основными профессиональными понятиями на английском.
► Улучшение навыка целенаправленного оценивания других докладов.

Учебная ситуация 2:
Подготовка клиентской документации для реализации среды разработки в Java (на русском или английском)

Исходная ситуация:

Компания по разработке ПО **ProSource** разрабатывает индивидуальное ПО для клиентов. Для некоторых клиентов важно иметь возможность самостоятельно совершенствовать программы или изменять существующие интерфейсы.

Поэтому `ProSource` предлагает своим клиентам подробную документацию для каждого проекта. Для клиента (Банк45), который предлагает в основном Интернет-банкинг, `ProSource` разработал интерфейс для передачи данных по оборотам по счету с веб-сервера на внутреннюю систему обработки.

Этот интерфейс был разработан в Java. Чтобы Банк45 имел возможность модифицировать интерфейс с помощью своих разработчиков, ему необходима бесплатная среда разработки. Как практикант компании `ProSource` Вы получаете заказ на подготовку клиентской документации для данной среды разработки.

Цель – вводное описание среды разработки и описание того, как может быть создан определенный проект (так называемое консольное приложение). Для улучшения понимания следует представить небольшой пример программирования. Объем данной документации должен быть не более пяти страниц. Документацию также следует использовать в иностранных представительствах банка и поэтому она должна быть продублирована на английском языке.

Работа индивидуально и в паре:

Планирование:

Подумайте над концепцей данной клиентской документации. Ориентируйтесь на следующие вопросы:

► Из каких частей может состоять документация (общее описание, создание проекта, пример программирования)?

► С помощью какой программы возможна соответствующая реализация программы (Editor, Word и т. п.)?

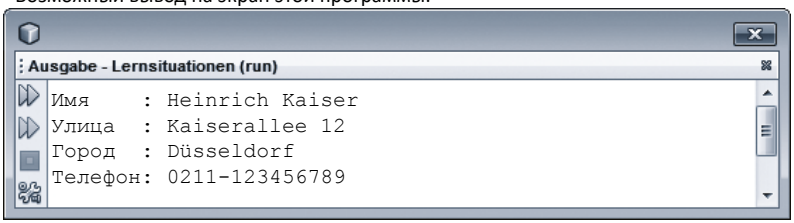
Выберите среду разработки. Предлагается бесплатная среда разработки ***NetBeans***. Если Вы в Вашем учебном блоке работаете с другой средой, Вы можете использовать ее.

Если возможно, поработайте над документацией также на занятиях по английскому языку.

Для примера программирования Вы получаете следующее задание от Вашей компании:

Необходимо написать программу Java, которая отображает на экране своего рода визитную карточку программиста.

Возможный вывод на экран этой программы:



При работе над этой проблемой необходимо учесть следующие вопросы:

► Как создается проект Java в среде разработки?

► Как выглядит «главная программа» и с помощью какого оператора выполняется вывод на экран?

Используйте информационный блок данной книги и другие источники сети Интернет для проработки необходимых знаний.

Выполнение:

Оформите клиентскую документацию с соответствующей сменой текста и графика (скриншоты среды разработки).

Отформатируйте исходный текст другим шрифтом.

Контроль:

Проверьте правописание Вашего текста. Если возможно, дайте кому-нибудь прочитать Вашу документацию, кто не знаком со средой разработки – это покажет, ведут ли Ваши пояснения к цели.

Учебные цели:

► Ознакомление со средой разработки *NetBeans*.

► Проработка основной структуры программы Java.

► Узнавание обязанностей для запуска программы Java.

► Улучшение знаний английского языка и знакомство с основными профессиональными понятиями на английском..

Учебная ситуация 3:
Разработка шифра для внутренней системы памяти отдела поддержки сетевой компании

Исходная ситуация:

Как специалист по системной интеграции компания **NetSolution** предлагает своим клиентам полный сервис. В этот сервис также входит бесплатная поддержка всех клиентов в течение 8 месяцев.

Все запросы клиентов (по телефону или E-Mail) сохраняются сотрудниками отдела поддержки в форме записей или Memo-файлов. Эта система памяти – простой веб-продукт компании **ProSource**, доступный всем клиентам **ProSource** бесплатно. Возможные расширения функционала системы (конечно, платно) можно заказать в ProSource. **NetSolution** желает сделать заказ на такое расширение функционала. Memo-файлы по умолчанию сохраняются в незашифрованном тексте. В системе памяти необходимо создать простое, но относительно надежное шифрование. После оценки требований безопасности и расходов представители компании выбрали шифрование, основанное на Квадрате Полибия* с дополнительным кодовым словом. Как опытный практикант компании ProSource вы получаете заказ на реализацию модуля шифрования.

* Полибий был греческим писателем и историком (200–120 до н. э.). Уже тогда он занимался техниками шифрования.

Работа индивидуально и в паре:

Планирование:

Техника шифрования, которую необходимо создать, представлена здесь схематично.

1. Выбор ключевого слова: например, PROGRAMMIEREN
2. Заполнение кодирующей матрицы: Все буквы ключевого слова вводятся в матрицу, но без повторений. Затем заполняются остальные буквы алфавита.

	1	2	3	4	5
1	P	R	O	G	A
2	M	I	E	N	B
3	C	D	F	H	J
4	K	L	Q	S	T
5	U	V	W	X	Y
6	Z	Пустые символы			

3. С помощью этой матрицы цепи символов шифруются. Каждая буква получает двузначное число (строка и столбец). Так эти числа дают шифрование цепи символов.

Пример: DAS IST EIN TEST

Шифр: **32 15 44 62 22 44 45 62 23 22 24 62 45 23 44 45**
Реализация шифрования предполагает работу с одномерными и многомерными массивами из информационного блока.

Выполнение:

Исполните подходящий класс с методами, которые могут выполнить шифрование. При этом зашифрованные цепи символов должны быть сохранены в массиве типа данных `int`. Класс должен шифровать и расшифровывать любое количество ключевых слов.

Контроль:

Каждая команда разработчиков составляет письменную матрицу шифрования с самостоятельно выбранным ключевым словом. Эти матрицы служат основой теста для контроля корректного шифрования и расшифровки.

Протестируйте модуль по условиям заказа. Зашифруйте и расшифруйте очень длинные цепи символов, которые по объему соответствуют Memo-тексту. Это может быть около 300 слов или 2000 символов.

Учебные цели:

- Знакомство с интересным применением программирования – техникой шифрования.
- Проработка необходимых знаний об одномерных и многомерных массивах в Java.
- Знакомство с особенностями проработки цепей символов в Java.

Учебная ситуация 4:

Планирование, внедрение и анализ электронной анкеты

Исходная ситуация:

Психологический факультет большого немецкого университета проводит исследования с помощью компьютера. В специальном исследовании, которое измеряет когнитивные способности при стрессе (нехватка времени), часть этого исследования должна проводиться за компьютером.

Компания ***ProSource*** получает заказ на разработку программы для этой части исследования.

В программе участникам эксперимента должно задаваться пять вопросов. На каждый вопрос предусмотрено три ответа, из которых нужно выбрать один. Кроме данного ответа программа также должна сохранять время ответа (в миллисекундах). Оформление вывода на экран должно быть очень простым, чтобы не отвлекать участников без необходимости. Поэтому руководство проекта ***ProSource*** выбирает консольное приложение. Как практикант компании ProSource вы уже разработали несколько консольных приложений. Поэтому задание поручается Вам.

Работа индивидуально и в паре:

Планирование:

Вы получаете каталог вопросов с заданными ответами от психологического факультета. Правильный ответ выделен жирным.

Вопрос 1: Какой роман написал Томас Манн?
► Ответ 1: Чума
► **Ответ 2: Волшебная гора**
► Ответ 3: Верноподданный

Вопрос 2: В какой битве Наполеон был окончательно побежден?
► Ответ 1: **Ватерлоо**
► Ответ 2: Маренго
► Ответ 3: Аустерлиц

Вопрос 3: С какой скоростью распространяется звук в воздухе?
► Ответ 1: 33 км/ч
► **Ответ 2: 330 м/с**
► Ответ 3: 3300 м/с

Вопрос 4: Какое открытие не делал Томас Алва Эдисон?
► Ответ 1: Электролампа
► Ответ 2: Фонограф
► **Ответ 3: Энигма**

Вопрос 5: Какой архитектор называл себя Ле Корбюзье?
► **Ответ 1: Шарль-Эдуа́р Жаннере́-Гри**
► Ответ 2: Фрэнк Райт
► Ответ 3: Мис вад дер Роэ

Программа должна иметь меню выбора, с помощью которого руководитель управляет исследованием:

```
Психологический институт II

<1> Запуск нового исследования
<2> Оценка исследования
<3> Конец
Ваш выбор: ?
```

После выбора пункта «Запуск нового исследования» сначала нужно указать номер участника. Потом может быть запущено само исследование.

Участник самостоятельно запускает исследование с помощью нажатия клавиши. С этого момента отсчитывается время, которое участнику требуется для ответа на каждый вопрос.

Далее участнику последовательно задаются пять вопросов.

После последнего вопроса участник переводится в конец исследования.

Руководитель исследования снова берет на себя управление программой и после нажатия клавиши появляется вышеупомянутое меню выбора.

Оценивание исследования может проводиться в любое время и должно отображать различные статистические данные.

Возможный вывод на экран:

```
                Психологический институт II
                 Исследование: 5 вопросов
                 ******ОЦЕНИВАНИЕ**********

Процент верных ответов на вопрос 1: 40%
Процент верных ответов на вопрос 2: 20%
Процент верных ответов на вопрос 3: 50%
Процент верных ответов на вопрос 4: 60%
Процент верных ответов на вопрос 5: 30%
Процент верных ответов на все вопросы: 40%

Среднее время ответа на вопрос 1: 4500 мс
Среднее время ответа на вопрос 2: 6200 мс
Среднее время ответа на вопрос 3: 2600 мс
Среднее время ответа на вопрос 4: 3700 мс
Среднее время ответа на вопрос 5: 4900 мс

Минимальное время ответа на вопрос 1: 1500 мс
Минимальное время ответа на вопрос 2: 2100 мс
Минимальное время ответа на вопрос 3: 900 мс
 Минимальное время ответа на вопрос 4: 1100 мс
Минимальное время ответа на вопрос 5: 1800 мс

Максимальное время ответа на вопрос 1: 9600 мс
Максимальное время ответа на вопрос 2: 6700 мс
Максимальное время ответа на вопрос 3: 4500 мс
Максимальное время ответа на вопрос 4: 6700 мс
Максимальное время ответа на вопрос 5: 8800 мс

Назад в меню выбора – Пожалуйста, нажмите клавишу. . . .
```

Выполнение:

Разработайте программу Java для вышеописанного задания.

Убедитесь в том, что ошибочный ввод пользователя невозможен.

Разработайте класс, в котором можно сохранять данные участника. Поскольку исследование можно проводить неограниченное количество раз, сохранение данных должно происходить динамично.

Вопросы и ответы в программном коде должны легко редактироваться – то есть сохраняться в соответствующей форме. Программа должна быть создана так, чтобы она позволяла добавлять дополнительные вопросы.

Примечание:

Отсчитывание времени можно осуществить с помощью метода `currentTimeMillis`. Этот метод дает миллисекунды с 1 января 1970 г.

```
long ВремяНачала = System.currentTimeMillis();
// действия
long Времяокончания = System.currentTimeMillis();
```

Контроль:

Протестируйте программу с помощью техники тестирования ***Black-Box***. Участники тестирования не знают внутреннюю структуру программы и тестируют только функциональность и результат. Контролируйте отсчет времени также вручную и моделируйте данные как можно большего количества участников теста.

Учебные цели:

- Вы учитесь планировать и реализовывать сложные задачи.
- Вы прорабатываете углубленные знания о понятии классов.
- Вы тестируете свою программу с помощью общей техники тестирования, ***места Black-Box.***

Учебная ситуация 5:

Разработка программного обеспечения для представления метеорологических данных с помощью схемы «Модель – Вид – Контроллер»

Исходная ситуация:

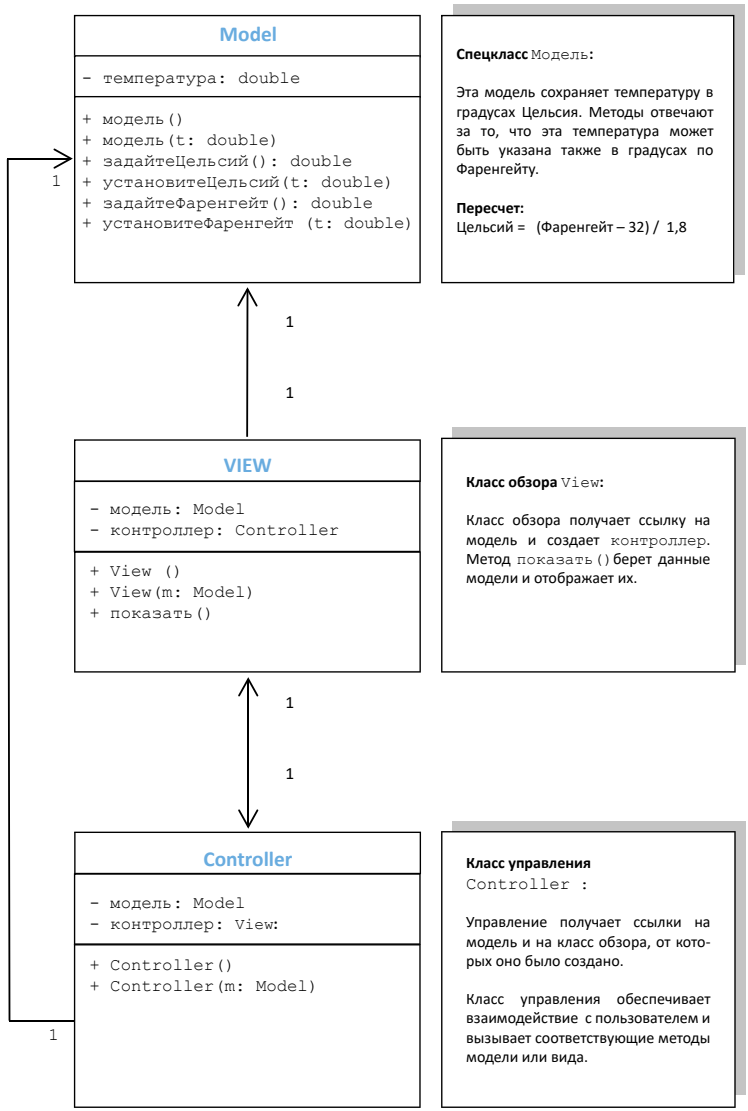
Компания ***WetterCom*** предоставляет своим клиентам данные измерений своих измерительных станций. До настоящего времени данные отправлялись как файловые вложения в текстовом формате. Этот несовременный вид отправки данных следует заменить новой системой ПО. Эта система должна быть реализована с помощью архитектуры **MVC** (архитектуры Model – View – Controller).

В первом образце необходимо создать локальное приложение, реализующее простую модель MVC. Компания **ProSource** получает заказ на разработку этого образца приложения на языке Java как консольного приложения. Как практикант компании **ProSource** Вы уже разработали несколько консольных приложений. Поэтому задание поручается Вам.

Работа индивидуально и в паре:

Планирование:

Объектно-ориентированный анализ дает следующую схему классов. Она описывает структуру классов и их взаимосвязь.


Model
- температура: double
+ модель()
+ модель(t: double)
+ задайтеЦельсий(): double
+ установитеЦельсий(t: double)
+ задайтеФаренгейт(): double
+ установитеФаренгейт (t: double)
1
Спецкласс Модель:
Эта модель сохраняет температуру в градусах Цельсия. Методы отвечают за то, что эта температура может быть указана также в градусах по Фаренгейту.
Пересчет:
Цельсий = (Фаренгейт – 32) / 1,8
1
1
VIEW
- модель: Model
- контроллер: Controller
+ View ()
+ View(m: Model)
+ показать()
Класс обзора View:
Класс обзора получает ссылку на модель и создает контроллер. Метод показать() берет данные модели и отображает их.
1
1
Controller
- модель: Model
- контроллер: View:
+ Controller()
+ Controller(m: Model)
1
Класс управления Controller :
Управление получает ссылки на модель и на класс обзора, от которых оно было создано.
Класс управления обеспечивает взаимодействие с пользователем и вызывает соответствующие методы модели или вида.

Примечание:

Вышеприведенная модель – это простой вариант архитектуры MVC, так как необходимо инстанцировать только одну модель, один вид и управление. Обычно бывает несколько видов или управлений, которые имеют доступ к модели.

Вывод на экран консольного приложения может выглядеть так:

```
Выбор:
<1> Задать новую температуру в градусах Цельсия
<2> Задать новую температуру в градусах по Фаренгейту
<3> КОНЕЦ
1
Пожалуйста, укажите температуру в градусах Цельсия: 10
```
Controller

```
ТЕМПЕРАТУРА - ПЕРЕСЧЕТ <Версия 1.0>
температура в градусах Цельсия:   10
температуру в градусах по Фаренгейту: 50
```
View

```
Выбор:
<1> Задать новую температуру в градусах Цельсия
<2> Задать новую температуру в градусах по Фаренгейту
<3> КОНЕЦ
1
Пожалуйста, укажите температуру в градусах Цельсия: 20
```
Controller

```
ТЕМПЕРАТУРА - ПЕРЕСЧЕТ <Версия 1.0>
температура в градусах Цельсия:   20
температуру в градусах по Фаренгейту: 68
```
View

Выполнение:

Перед реализацией выполните поиск информации по темам схемы классов ***MVC*** и ***UML*** в Интернет или других источниках, чтобы понять структуру этой архитектуры и символику ***UML*** в целом.

Затем создайте три класса в соответствии со схемой классов. При этом логика ввода (меню и ввод пользователя) должна применяться как метод класса управления.

Вывод на экран реализуется как метод класса обзора, который вызывается классом управления, как только пользователь сделал новый выбор. Класс модель используется как классом управления (для сохранения новых значений), так и классом обзора для получения данных для вывода на экран.

«Главная программа» в Java определена заранее:

```
public static void main(String[] args) throws IOException {
        Model модель = new Model (10);
        View вид = new View(модель);
}
```

Контроль:

Проверьте корректное исполнение программы путем ввода нескольких значений, которые необходимо пересчитать и представить.

Учебные цели:

- Ознакомление с основами важной архитектуры – архитектуры ***Model – Vier – Controller***
- Проработка знаний по реализации **схем классов UML.**

Учебная ситуация 5:

Разработка приложения для решения головоломки «Судоку»

Исходная ситуация:

Известное издательство журналов с головоломками желает повысить ценность своих продуктов с помощью бесплатного приложения Android. К различным журналам с головоломками (кроссворды, судоку и др.) через платформы типа Google play нужно предлагать бесплатные приложения. Так клиенты будут сильнее привязаны к продуктам. Компания **ProSource** получает заказ от издательства на разработку приложения, которое самостоятельно может решать судоку. В первом варианте должны решаться судоку 4x4.

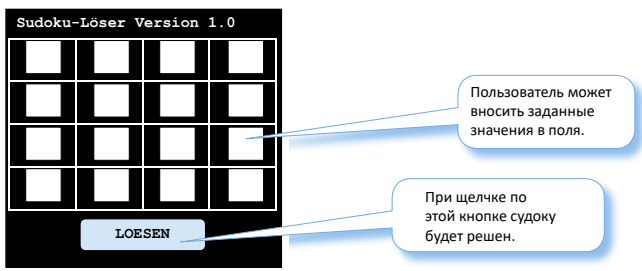
Как практикант компании Вы получаете задание на разработку этого приложения.

Работа индивидуально и в паре:

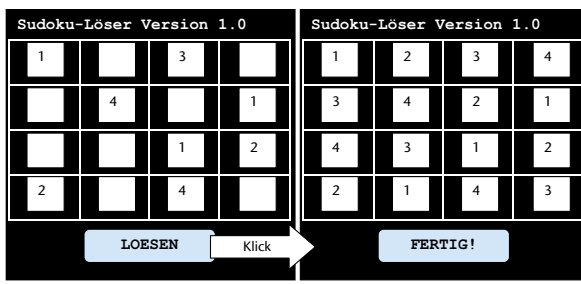
Планирование:

Издательство представило проект, который должен стать основой разработки:

Проект:



Пример приложения:



Выполнение

► Разработайте приложение как проект Android.

► Используйте `GridLayout` для расположения отдельных полей ввода (`EditText`).

► Выполните поиск информации об алгоритмах, с помощью которых можно решить такие проблемы (проблемы в играх).

Примеры таких алгоритмов:

- **Метод исчерпывания:** систематическая проба всех вариантов
- **Поиск с возвратом:** неверные решения ведут к обратному ходу
- **Человеческая логика:** постпрограммирование образа действия человека

► Если время позволяет, расширьте функции приложения до возможности решения судоку 9x9.

Контроль:

► Проведите подробные тестирования Blackbox с практикантами.

Учебные цели:

► Проработка перспективной темы программирования Java – разработка приложения.

► Ознакомление с различными алгоритмами для решения комплексных проблем.

Приложение A: Структурированная техника документации

Логическая схема программы:

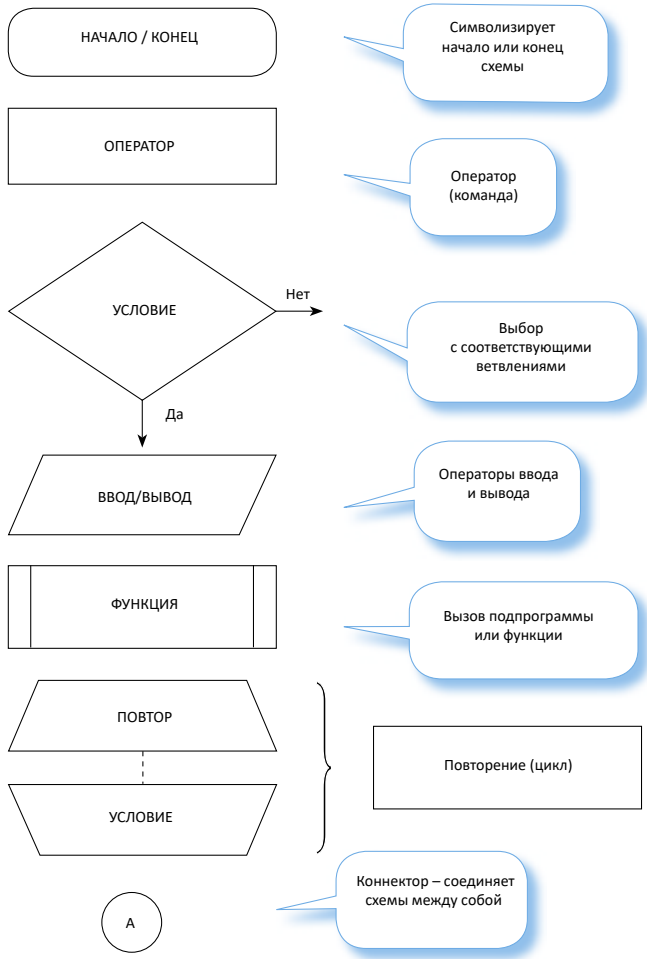
Логическая схема программы – это графическое представление алгоритма. Она является основой для реализации алгоритма в языках программирования типа Java.

Символы логической схемы программы описаны в стандарте DIN 66001.

Внимание:

В схеме не должно быть элементов, специфических для одного языка программирования. Схема находится на более высоком уровне и может применяться для любого структурированного языка программирования.

Символы:

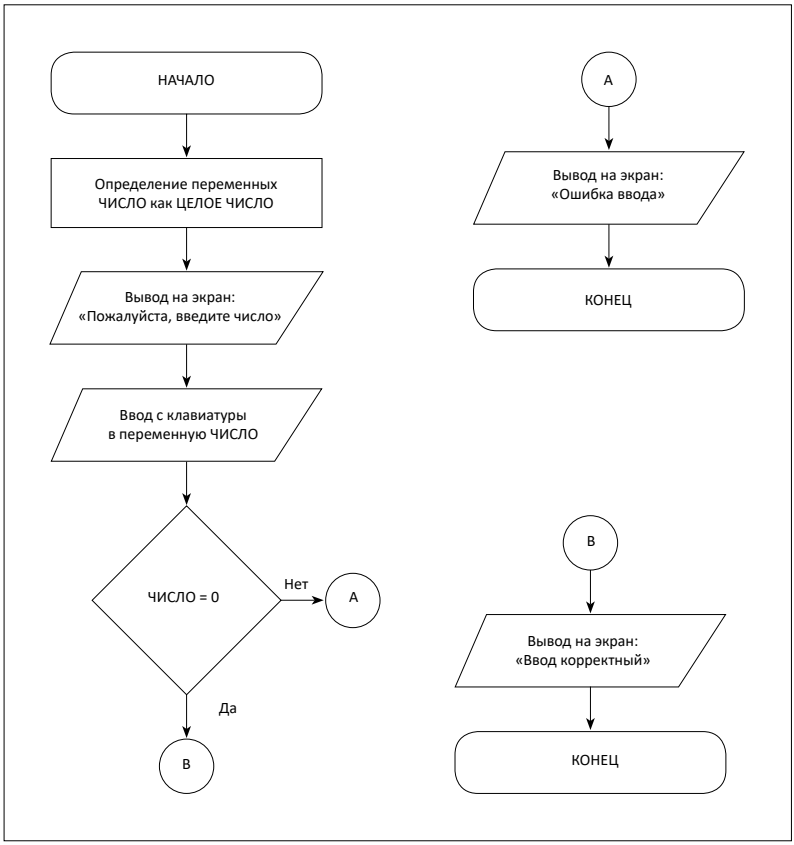


Пример логической схемы программы:

Проблема:

Пользователь должен ввести число с клавиатуры. Если он вводит ноль, на экран должно быть выведено сообщение «Ошибка ввода», в ином случае «Ввод корректный»

Схема:



Примечание:

Схема представлена с помощью свободной программы ПО `Dia` (ссылка: http://www.gnome.org/projects/dia/). С помощью этой программы можно графически представлять многие типы диаграмм (включая диаграммы UML).

Серия «Профессиональное образование»

Java для IT-профессий

Учебник

Редактор С. Уралова
Технический редактор Э. Заманбек
Художественный редактор А. Кунтуова
Корректор В. Селезнева
Компьютерная верстка Д. Скаковой

Подписано к печати 15.10.2019.
Формат 60x90 $^{1}/_{16}$. Бумага офсетная.
Печать офсетная. Усл. п.л. 17,5.
Тираж 50 экз. Заказ №0198.

Издательство «Фолиант»
010000, г. Нур-Султан, ул. Ш. Айманова, 13

Отпечатано в типографии «Регис-СТ Полиграф»
010000,г. Нур-Султан, ул. Ш. Айманова, 13