Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ. ЧАСТЬ II

Методические указания к лабораторным работам и
организации самостоятельной работы
для студентов направления
«Программная инженерия»
(уровень бакалавриата)

2018

Морозова Юлия Викторовна
Информатика и программирование. Часть II: Методические указания к лабораторным работам и организации самостоятельной работы для студентов направления «Программная инженерия» (уровень бакалавриата) / Ю.В. Морозова. – Томск, 2018. – 82 с.

Оглавление

1 Введение

Целью дисциплины «Информатика и программирование» является формирование у студентов объектно-ориентированного мышления и объектно-ориентированного (ОО) подхода, в том числе к анализу предметной области и использование объектно-ориентированной методологии программирования при разработке программных продуктов.

В ходе изучения дисциплины решаются следующие задачи:

- изучение техники объектно-ориентированного анализа;
- изучение приемов объектно-ориентированного программирования (ООП);
- изучение технологии проектирования архитектуры информационных систем;
- изучение основ проектирования информационно-коммуникационных технологий (ИКТ) и основ управления ИКТ-проектами.

В результате изучения дисциплины студент должен:

- *знать* методы обработки и способы реализации основных структур данных в объектно-ориентированных программных средах.
- *уметь* разрабатывать объектно-ориентированные программы в современных инструментальных средах.
- *владеть* практическими приемами объектно-ориентированного программирования.

Данные методические указания предназначены для организации самостоятельной работы и выполнения лабораторных работ для бакалавров направления подготовки «Программная инженерия», изучающих дисциплину «Информатика и программирование».

2 Методические указания к проведению лабораторных работ

2.1 Правила выполнения лабораторных работ

В ходе выполнения лабораторной работы студент должен строго выполнять весь объем самостоятельной подготовки, указанный в описаниях соответствующих лабораторных работ. Выполнению каждой работы предшествует проверка готовности студента, которая проводится преподавателем.

Лабораторные занятия выполняются студентами самостоятельно, преподаватель в ходе занятия осуществляет научное и методическое руководство действиями студентов.

Форма отчетности: отчет с листингом программы и комментариями.

После выполнения работы студент должен представить отчет о проделанной работе с обсуждением полученных результатов и выводов.

1. Тему и цель лабораторной работы.
2. Вариант задания на лабораторную работу.
3. Краткие теоретические сведения и описание алгоритма работы программы.
4. Листинг разработанной программы с подробными комментариями.
5. Результаты работы программы.
6. Выводы.

Защита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов, демонстрации полученных навыков в ответах на вопросы преподавателя.

Отчет оформляемся согласно образовательному стандарту вуза. Титульный лист представлен в приложении А. Изложение должно быть последовательным, логичным, конкретным. В текст отчета могут быть включены небольшие фрагменты программного кода, обязательно с комментариями. Рекомендуемый шрифт для выполнения фрагмента кода – Courier New, размер 12пт. На материалы, взятые из литературы и других источников должны быть даны ссылки с указанием номера источника по списку использованной литературы. В приложениях размещаются листинг, схемы программы, скриншоты интерфейса. Приложения нумеруются русскими буквами в порядке появления ссылок на них в основном тесте документа.

2.2 Лабораторная работа «Знакомство с объектно-ориентированным языком Java и IDE Eclipse»

Цель работы: познакомиться с языком программирования Java и средой Eclipse. Изучить настройки среды, представления Projects, Packages, Debug. Написать простейшую программу на языке Java, скомпилировать ее и отладить.

Теоретические основы

Изучение принципов объектно-ориентированного программирования будет проходить с помощью языка программирования Java. Язык программирования Java является полностью объектно-ориентированным.

Минимальный комплект для разработки программ на Java

- JDK (Java Development Kit) – комплект разработки программного обеспечения (компилятор, стандартные библиотеки и т.п.).
- JRE (Java Runtime Environment) – это программа для запуска и исполнения программ (среда выполнения Java).
- среда программирования Eclipse.

Самые новые версии системного программного обеспечения JRE, JDK можно загрузить с сайта компании Sun (http://java.sun.com/).

Eclipse – один из лучших инструментов Java, созданных за последние годы. Eclipse представляет собой интегрированную среду разработки (IDE, Integrated Development Environment) с открытым исходным кодом. Поддержкой и разработкой Eclipse в настоящее время занимается организация Eclipse Foundation и сообщество Eclipse, информацию о которых можно найти на официальном сайте в сети Интернет (http://www.eclipse.org).

Запуск Eclipse

1. Чтобы запустить Eclipse IDE, нужно открыть файл **eclipse.exe**, находящийся в папке **C:\Program Files\eclipse** (рис. 2.1).

2. В простейшем случае рабочее пространство (workspace) – это каталог для проектов пользователя, в котором располагаются файлы проекта. Все, что находится внутри этого каталога, считается частью рабочего пространства. При запуске откроется окно, предлагающее выбрать рабочую область (Workspace) (рис. 2.2), где будут храниться программные файлы проекта. Указываем удобную для нас директорию и нажимаем **OK**.

Рис. 2.1 – Заспуск eclipse

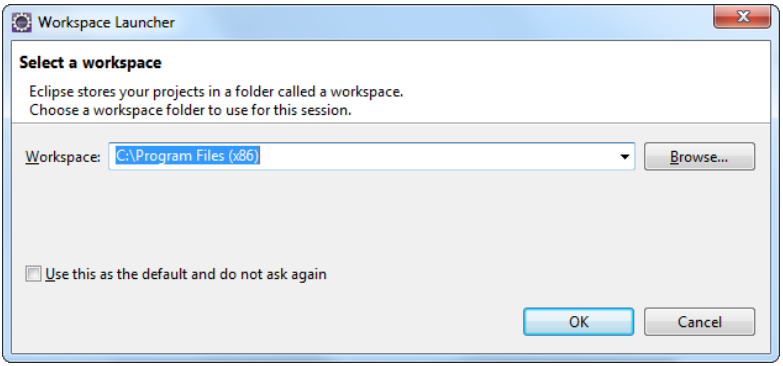

Рис. 2.2 – Workspace

Инструментальные средства Eclipse становятся доступны сразу после запуска приложения. Это по существу сама платформа с набором различных функциональных возможностей главного меню, где прежде всего выделяется набор операций по управлению проектом. Фактическая обработка, как правило, осуществляется дополнениями (плагинами), например, редактирование и просмотр файлов проектов осуществляется JDT, и т.д.

К *инструментам* (workbench) относится набор соответствующих редакторов и представлений, размещенных в рабочей области Eclipse.

Для конкретной задачи определенный набор редакторов и представлений называют перспективой или компоновкой.

Компоновка (perspective) – это набор представлений и редакторов, расположенных в том порядке, который вам требуется.

В каждой компоновке присутствует свой набор инструментов, некоторые компоновки могут иметь общие наборы инструментов. В определенный момент времени активной может быть только одна компоновка. Переключение между различными компоновками осуществляется нажатием клавиш <Ctrl+F8>.

Используя компоновки, вы можете настроить свое рабочее пространство под определенный тип выполняемой задачи. В Eclipse имеется также возможность создавать свои компоновки. Открыть компоновку можно командой Window / Open Perspective (рис. 2.3).

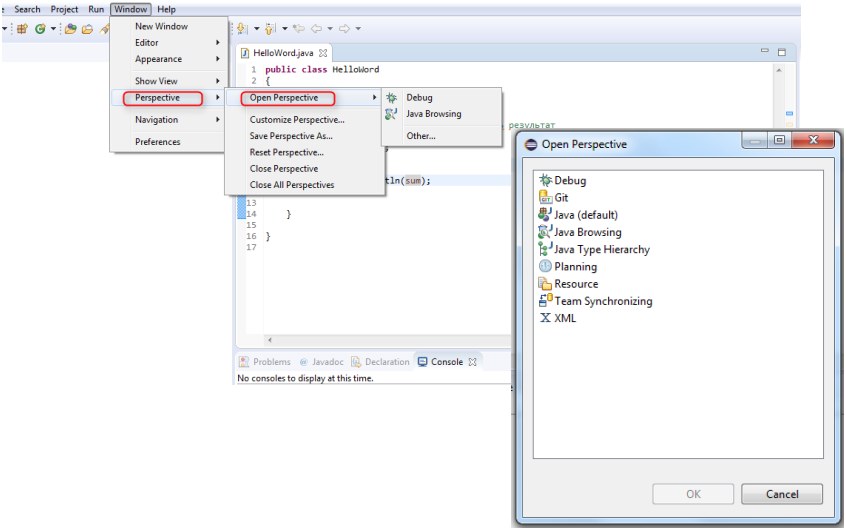

Рис. 2.3 Perspective

Редакторы (editors) представляют собой программные средства, позволяющие осуществлять операции с файлами (создавать, открывать, редактировать, сохранять и др.).

Представления (views) по существу являются дополнениями к редакторам, где выводится информация сопроводительного или дополнительного характера, как правило, о файле, находящемся в редакторе. Открыть представления можно командой Window / Show View (рис. 2.4).

Проект (project) представляет собой набор файлов приложения и сопутствующих дополнений.

Дополнением (plug-in) называют приложение, которое дополнительно может быть установлено в Eclipse.

Примером дополнения может выступать JDT.

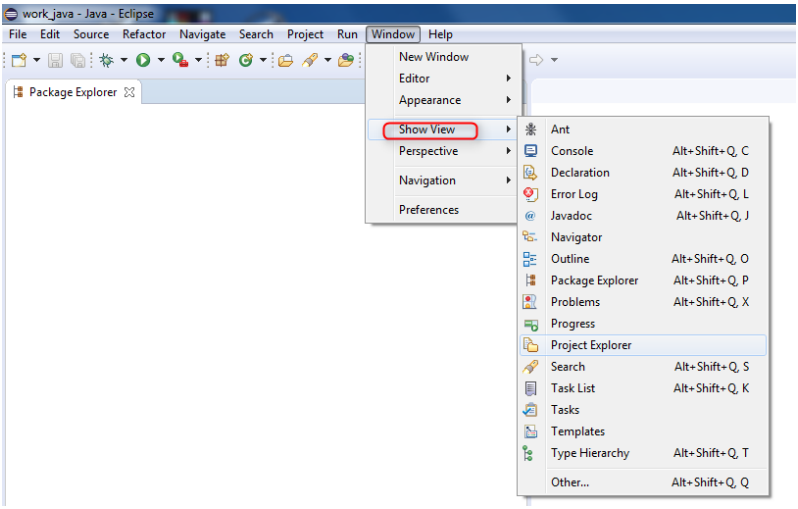

Рис. 2.4 – Views

Проект Java development tools (JDT) с помощью JDT-плагина обеспечивает среду разработки Java-приложений, включая создание Eclipse-плагинов. JDT-плагин добавляет перспективу Java в панель инструментов и Java-группу шаблонов в команду New меню File, а также предоставляет набор окон, редакторов и других инструментов для работы с Java-кодом.

Eclipse-плагины добавляют к Eclipse-платформе новые типы редакторов, представлений и перспектив. К существующим редакторам, представлениям и перспективам могут добавляться новые действия в меню и панелях инструментов.

После того, как вы нажмете кнопку «OK» на окне приветствия, появится страница приветствия (рис. 2.5), на которой имеется 5 графических кнопок:

- Overview – обзор, содержащий ссылки на обучающие интернет-ресурсы eclipse;
- Tutorials – уроки, содержит несколько примеров создания простейших приложений Java;
- What’s new – содержит обзор основных нововведений;
- Samples – примеры, содержит несколько примеров разработки, которые должны быть предварительно установлены для того, чтобы их можно было просмотреть;
- Workbench – «рабочий стол» — это рабочая область программиста.

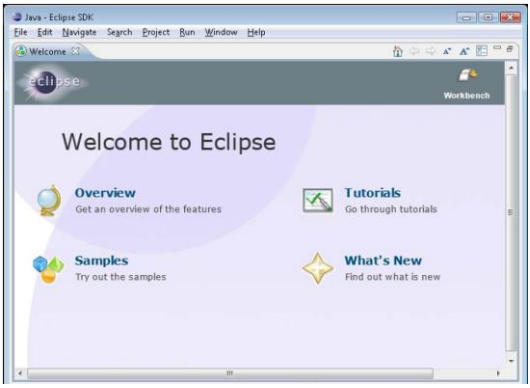

Рис. 2.5 – Окно приветствия eclipse

Перспектива Java содержит окно редактора и представления Package Explorer, Outline, Problems, Javadoc, Declaration (табл. 2.1). Перспектива Debug содержит окно редактора и представления Debug, Breakpoints, Console, Tasks (табл. 2.1).

Таблица 2.1 – Package Explorer, Outline, Problems, Javadoc, Declaration

Представления	**Описание**
Package Explorer	Отображает Java-проект с его структурой, определяемой сборкой проекта, в виде узлов папок и библиотек, пакетов, файлов с их внутренней структурой
Outline	Отображает компилируемую структуру редактируемого в данный момент Java-файла
Problems	Отображает ошибки и предупреждения сборщика проекта
Javadoc	Отображает документацию выбранного в данный момент Java-элемента
Declaration	Отображает исходный код выбранного в данный момент Java-элемента
Console	Отображает системный вывод выполнения Java-кода
Debug	Обеспечивает управление процессом отладки и запуска Java-кода
Tasks	Отображает список маркеров задач проекта
Breakpoints	Отображает список контрольных точек отладки Java-кода

Теперь создадим новый проект. Для этого выберем меню File->New->Project. В открывшемся окне выберем «Java Project» и нажмем «Next» (рис. 2.6).

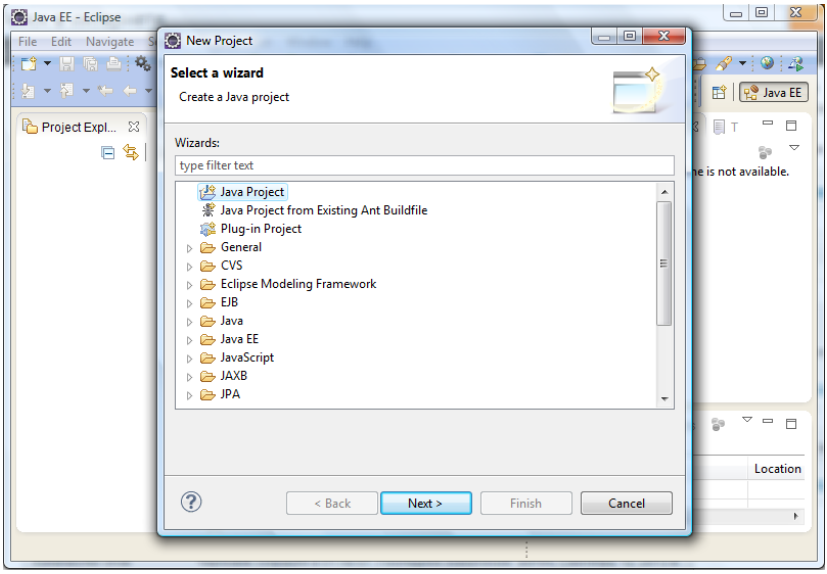

Рис. 2.6 – Создание нового проекта

В следующем окне введем имя нашего проекта и нажмем «Finish» (рис. 2.7).

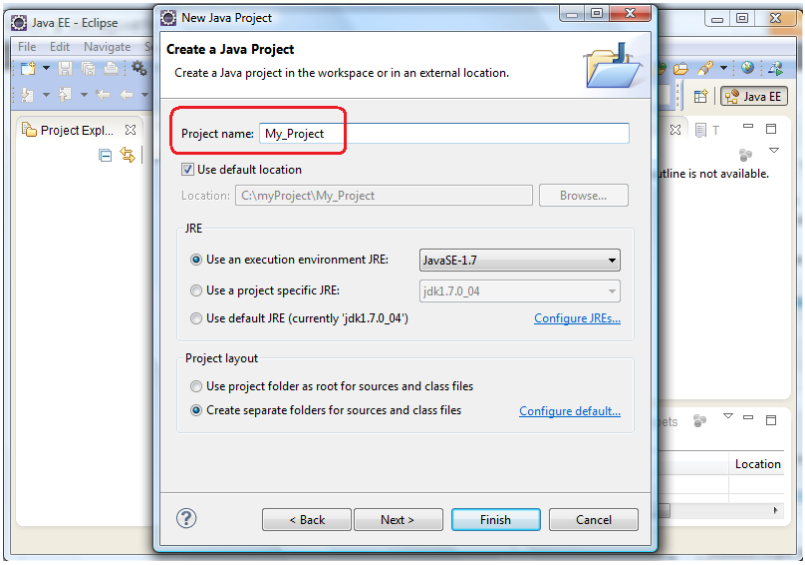

Рис. 2.7 – Создание нового проекта

Проект отобразится в левой части экрана и должен в себе содержать элемент JRE System Library/. Это представление называется Обозреватель Пакетов (Package Explorer) (рис. 2.8).

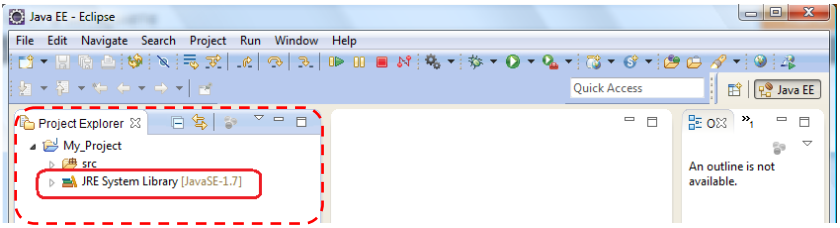

Рис. 2.8 – Package Explorer

Далее рассмотрим создание программы *Hello World* в Eclipse. Первым делом необходимо создать класс. Нажмем правой кнопкой на папке с проектом и выберем из контекстного меню New -> Class (рис. 2.9).

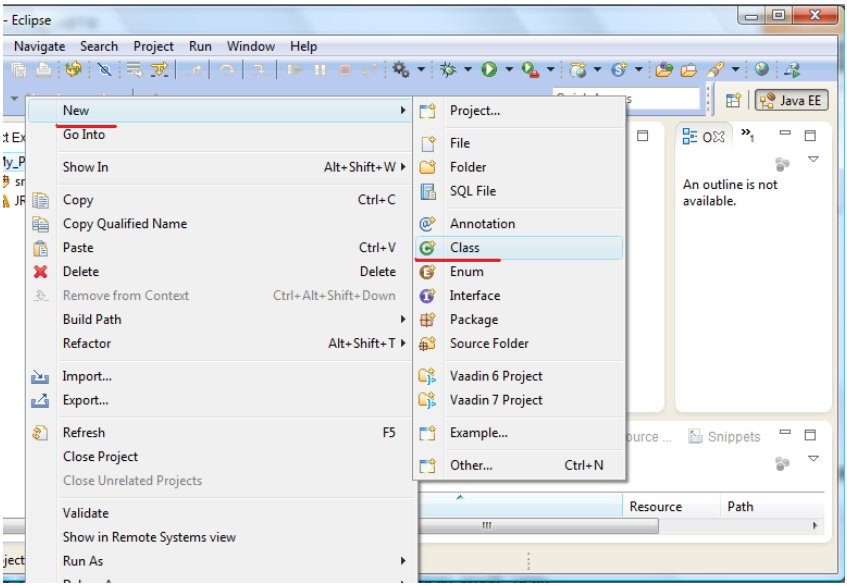

Рис. 2.9 – Создание нового класса

В открывшемся окне «New Java Class» введем имя класса проекта «HelloWorld» и установим флажок для метода public static void main(String[] args), далее «Finish» (рис. 2.10). В итоге, Eclipse создаст новый класс Hello World. Откроем созданный класс и завершим нашу программу. Добавим в метод main() следующий код (рис. 2.11) System.out.println("Hello World");

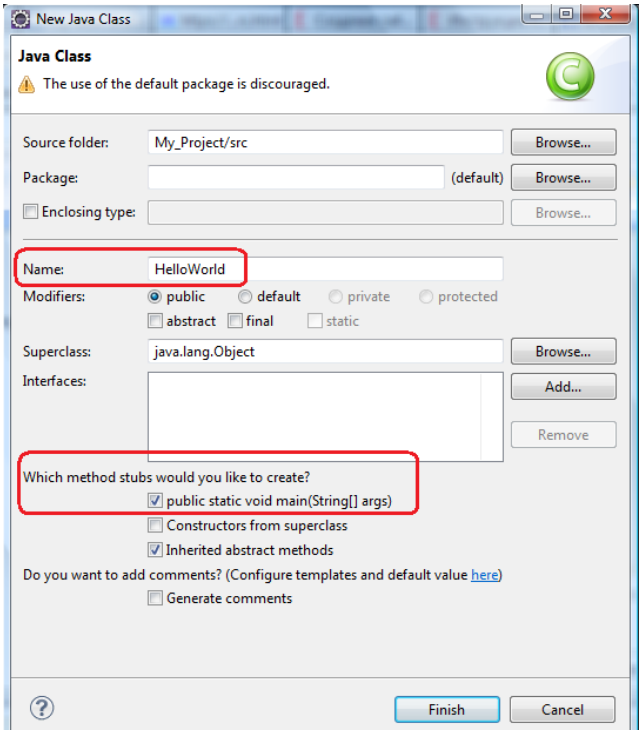

Рис. 2.10 – Создание нового класса

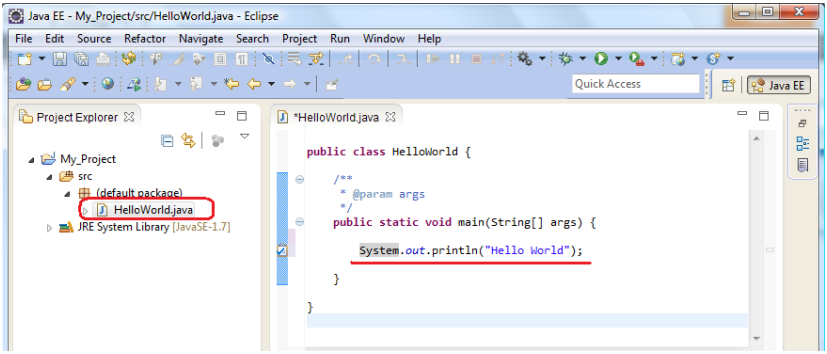

Рис. 2.11 – Создание нового класса

Сохраним изменения с при помощи клавиш <Ctrl+S> или специального значка вверху на панели инструментов.

Далее запустим наш проект, для этого в меню выберем Run -> Run Configurations. В открывшемся окне в левой части 2 раза кликнем на «Java Application» после чего, будет создан новый под элемент с именем «New_configuration», которое впоследствии в правой части можем изменить (рис. 2.12). В правой части также заполним поля Project и Main Class. Project должен содержать имя проекта, Main Class – имя главного класса, в нашем случае – HelloWorld. После чего нажмем «Apply» и «Run».

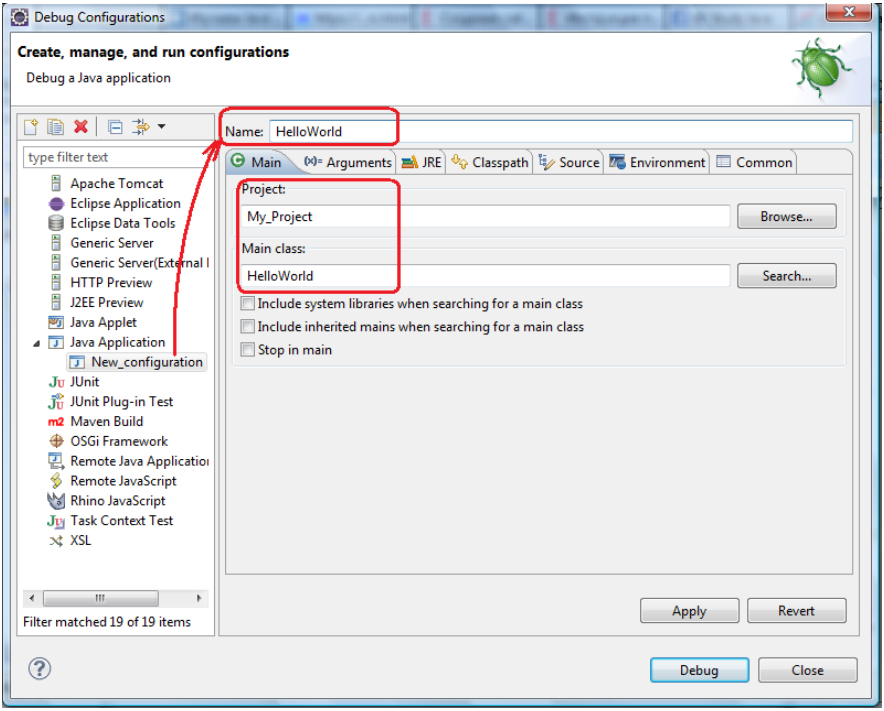

Рис. 2.12 – Запуск проекта

В результате, в консоли будут напечатаны слова *Hello World* (рис. 2.13).

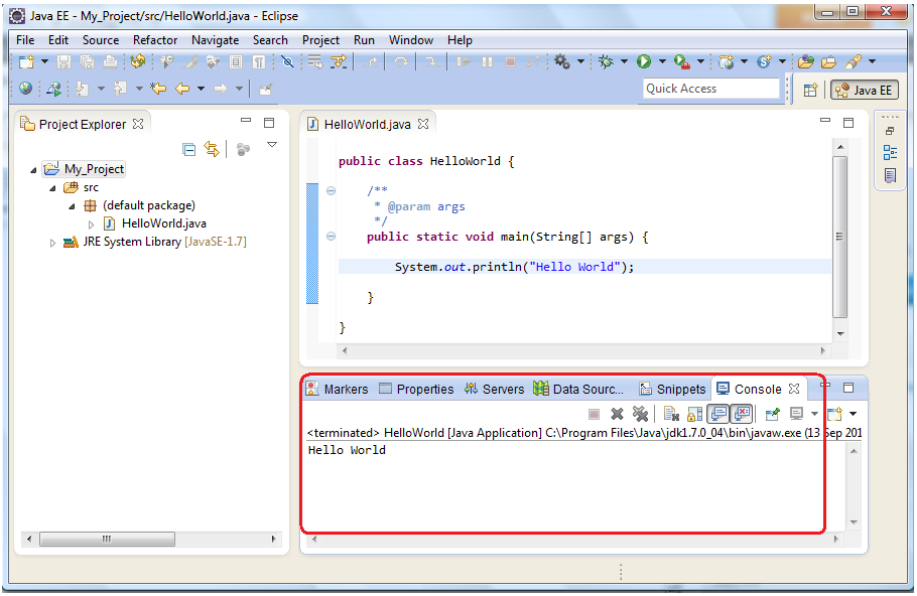

Рис. 2.13 – Console

Для запуска программы в дальнейшем, достаточно нажимать специальный значок на панели инструментов, выбрав «Hello World» (рис. 2.14).

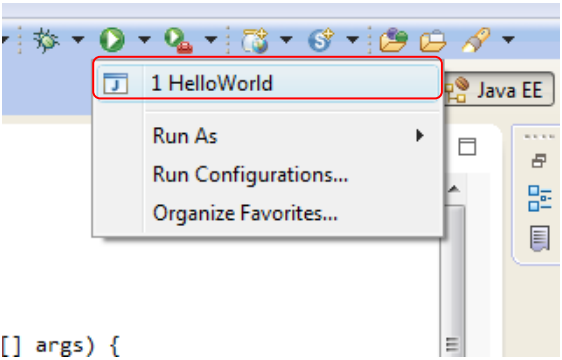

Рис. 2.14 – Запуск проекта

Можно запустить этот файл и через командную строку. Переходим в каталог, где лежит данный файл, и выполняем команды:

javac HelloWorld.java

В данной папке появится файл «HelloWorld.class». Значит программа скомпилирована. Чтобы запустить:

```
java -classpath . HelloWorld
```

В Package Explorer, вы увидите массу папок. Все они содержат файлы, требуемые для создания и последующего распространения библиотеки. Рассмотрим их подробнее:

- **src** содержит исходный код библиотеки на Java.
- **JRE System Library** содержит ссылки на файлы из Java Runtime Environment, необходимые для компиляции нашей библиотеки.
- **Referenced Libraries** содержит ссылку на основной файл Processing – core.jar, который мы добавили.
- **data** содержит изображения, звуки и все, что нужно для библиотеки.
- **distribution** содержит, все, что нужно для распространения библиотеки.
- **examples** используется для хранения примеров скетчей для нашей библиотеки.
- **lib** содержит сторонние файлы .jar для библиотеки, если вы их добавили.
- **resources** содержит файлы для процесса сборки.
- **web** содержит шаблон html.

По давней традиции, восходящей к языку Си, учебники по языкам программирования начинаются с программы "Hello, World!". Не будем нарушать эту традицию.

```
class HelloWorld {
public static void main(String[] args)  {
System.out.println("Hello, World!");
} }
```

Всякая программа, написанная на языке Java, представляет собой один или несколько классов. Начало класса отмечается служебным словом class, за которым следует имя класса, выбираемое произвольно, в данном случае это имя HelloWorld. Все, что содержится в классе, записывается в фигурных скобках и составляет тело класса (class body) (рис. 2.15).

Все действия в программе производятся с помощью методов обработки информации (method). Методы используются в объектно-ориентированных языках вместо функций, применяемых в процедурных языках.

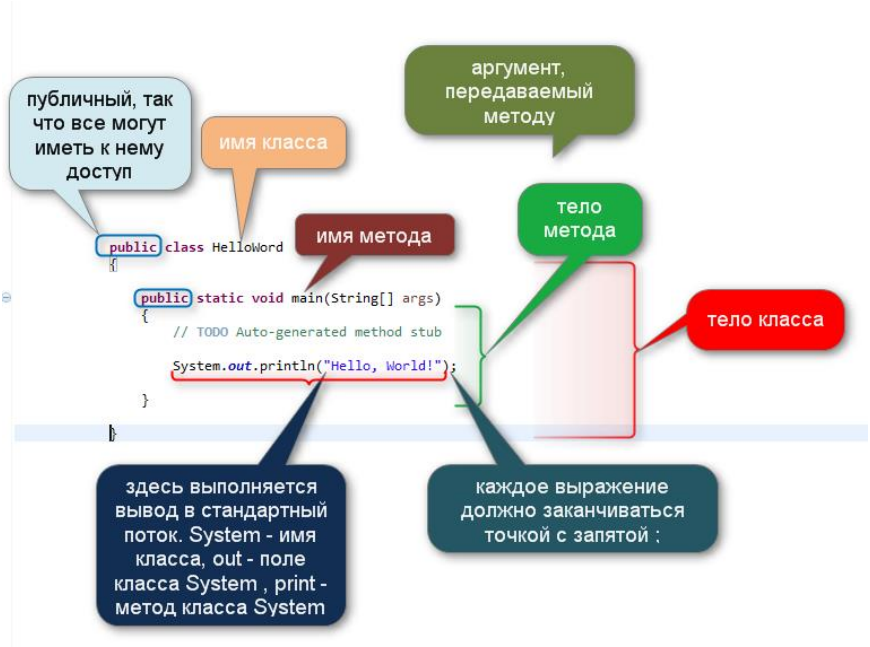

Рис. 2.15 – Программа, написанная на языке Java

Методы различаются по именам и параметрам. Один из методов обязательно должен называться main(), с него начинается выполнение программы. В нашей простейшей программе только один метод, а значит, имя его main (исключение составляют апплеты – у них метода main() нет). Метод main() иногда называют главным методом программы, поскольку во многом именно с этим методом отождествляется сама программа.

Ключевые слова public, static и void перед именем метода main() означают буквально следующее: public – метод доступен вне класса, static – метод статический и для его вызова нет необходимости создавать экземпляр класса (то есть объект), void – метод не возвращает результат. Модификаторы и уровни доступа мы тоже рассмотрим немного позже.

Инструкция String[] args в круглых скобках после имени метода main() означает тип аргумента метода: формальное название аргумента args, и этот аргумент является текстовым массивом (тип String).

Все, что содержит метод, тело метода (method body), записывается в фигурных скобках.

Фигурными скобками в языке программирования Java (как и C++ и C#) отмечаются блоки программного кода. Программный код размещается между открывающей (символ {) и закрывающей (символ }) фигурными скобками.

Единственное действие, которое выполняет метод main() в нашем примере, заключается в вызове другого метода со сложным составным именем System.out.println() и передаче ему на обработку одного аргумента – строчного литерала "Hello, World!". Строковые литералы записываются в кавычках, которые являются только ограничителями и не входят в текст.

Составное имя System.out.println() означает, что в классе System, входящем в Java API, определяется переменная с именем out, содержащая экземпляр одного из классов Java API, класса PrintStream, в котором есть метод println(). Все это станет ясно позднее, а пока просто будем писать это длинное имя.

Язык Java различает строчные и прописные буквы, имена main, Main, MAIN различны с «точки зрения» компилятора Java. В примере важно писать String, System с заглавной буквы, а main – со строчной.

В именах нельзя оставлять пробелы. Свои имена можно записывать как угодно, можно было бы дать классу имя helloworld или helloWorld, но между Java-программистами заключено соглашение, называемое «Code Conventions for the Java Programming Language» (http://www.oracle.com/technetwork/java/codeconv-138413.html).

Имя файла должно в точности совпадать с именем класса, содержащего метод main(). Данное правило очень желательно выполнять. При этом система исполнения Java будет быстро находить метод main() для начала работы, просто отыскивая класс, совпадающий с именем файла. Расширение имени файла должно быть java.

В нашем примере сохраним программу в файле с именем «HelloWorld.java» в текущем каталоге (рис. 2.16). Затем вызовем компилятор, передавая ему имя файла в качестве аргумента:

javac HelloWorld.java

Компилятор создаст файл с байт-кодами, даст ему имя «HelloWorld.class» и запишет этот файл в текущий каталог.

Осталось вызвать интерпретатор байт-кодов, передав ему в качестве аргумента имя класса (а не файла!):

java HelloWorld

На экране появится строка:

Hello, World!

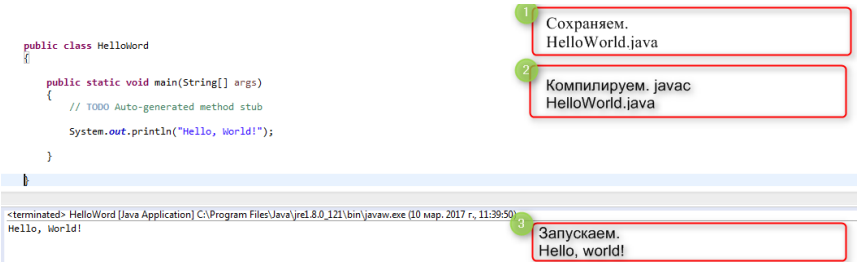

Рис. 2.16 – Запуск программы, написанной на языке Java

Порядок выполнения работы

1. Зайти среду Eclipse.
2. Набрать и запустить программу «HelloWorld!».
3. Изменить различные части этой программы и ознакомьтесь с полученными сообщениями об ошибках.
4. Выполнить работу согласно варианту.
5. Написать отчет по лабораторной работе.
6. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Определите первую и последнюю цифры двузначного натурального числа N. Число вводит пользователь.
2. Дано двузначное натуральное число, в котором все цифры различны. а) Определить его максимальную и минимальную цифры. б) Найти сумму его максимальной и минимальной цифр. Число задавать генератором случайных чисел.
3. Вывести на экран все целые числа от 100 до 200, которые содержат в своем составе цифру 3.
4. Найдите первую, вторую и третью цифры трехзначного натурального числа N. Число задавать генератором случайных чисел.
5. Найдите сумму из первой, второй и третьей цифр трехзначного натурального числа N. Число задавать генератором случайных чисел.
6. Определите четность числа n (true для четного числа и false для нечетного числа). Число вводит пользователь.

7. Две точки на плоскости заданы своими координатами. Выяснить, образуют ли эти точки вместе с центром координат прямоугольный треугольник. Координаты задавать генератором случайных числе.

8. Вывести на экран все целые числа от 100 до 200, кратные трем.

9. Найти все двузначные числа, которые делятся на n или содержат цифру n. n вводит пользователь.

10. Известно количество секунд, прошедших с начала дня. Найти три целых переменных (часы, минуты и секунды). Количество секунд задает пользователь.

Контрольные вопросы

1. Какие элементы языка Java имеют имена? Какие из них должны быть объявлены?
2. Из каких символов может состоять имя переменной (корректный идентификатор)?
3. Что такое IDE?
4. Перечислите основные представления, доступные в перспективе Java.
5. Перечислите основные представления, доступные в перспективе Debug.
6. Что такое компоновка (perspective)? Назовите три основные перспективы.
7. Что такое представление (views)?

2.3 Лабораторная работа «Массивы и строки»

Цель работы: получить практические навыки работе с массивами и строками.

Теоретические основы

Массив в Java представляет собой объект, где имя массива является *объектной ссылкой*.

Массив (Array) – это объект, хранящий в себе фиксированное количество значений одного типа.

Другими словами, массив – это нумерованный набор переменных. Переменная в массиве называется *элементом массива*, а ее позиция в массиве задается *индексом*.

Нумерация элементов массива начинается с 0, а длина массива устанавливается в момент его создания и фиксируется. Для того чтобы создать массив нужно его объявить, зарезервировать для него память и инициализировать.

Первый этап – *объявление* (declaration). На этом этапе определяется только переменная типа *ссылка* (reference) на массив, содержащая тип массива.

Одномерный массив – это ряд значений одного и того же типа, хранящихся в соседних ячейках памяти.

Синтаксис объявления одномерного массива выглядит так:

Тип_массива имя_массива [] = new тип_массива [размер];

Подобным образом можно объявить массив любого типа:

```
double[] a, b;
```

Здесь определены две переменные – ссылки a и b на массивы типа double. При объявлении происходит лишь создание ссылки на массив.

Второй этап – *определение* (instantation). На этом этапе указывается количество элементов массива, называемое его длиной, выделяется место для массива в оперативной памяти, переменная-ссылка получает адрес массива с помощью ключевого слова new или прямой инициализации. Значения элементов неинициализированного массива, для которого выделена память, устанавливаются в значения по умолчанию для массива базового типа или null для массива объектных ссылок.

Например,

```
a = new double[5];
b = new double[100];
ar = new int[50];
```

При этом все элементы массива получают *нулевые значения*. Можно эти два этапа записать в одно действие.

В Java осуществляется автоматическая проверка границ массива, и проверку невозможно отключить. Длина массива устанавливается в момент создания массива, и после этого не может быть изменена (ее можно изменить только создав новый массив). JVM проверяет выход за границы массива, и в случае необходимости генерирует исключение:

ArrayIndexOutOfBoundsException

Многомерных массивов в Java не существует, но можно объявлять массив массивов.

Многомерные массивы – это массивы, элементами которых являются массивы меньшей размерности. *Размерность* – это количество индексов элемента многомерного массива.

Для объявления многомерных массивов в Java используется следующая форма:

```
int D[][] = new int[4][];
D[0] = new int[2];
D[1] = new int[5];
D[2] = new int[3];
D[3] = new int[4];
```

Количество квадратных скобок указывает на размерность.

Примеры создания массивов фиксированной длины:

```
int[][] a = new int[5][5];// двумерный массив
int[][][] b = new int[3][4][5];// трехмерный массив
int[][][][] c = new int[2][4][5][5];/*четырехмерный массив*/
// и т.д.
```

Однако, не обязательно изначально указывать размер на всех уровнях, можно указать размер только на первом уровне.

```
int[][] a1 = new int[5][];
```

В данном случае, пока неизвестно сколько будет элементов в каждой строке, это можно определить позже, причем, массив может содержать в каждой строке разное количество элементов, то есть быть несимметричным. Определим количество элементов в каждой строке для массива a1

```
a1[0] = new int [1];
a1[1] = new int [2];
a1[2] = new int [3];
a1[3] = new int [4];
a1[4] = new int [5];
```

В математических вычислениях часто используются матрицы – двумерные массивы. В следующем примере создается матрица размером 4 на 4 с элементами типа double, причем ее диагональные элементы (те, для которых x==y) заполняются единицами, а все остальные элементы остаются равными нулю.

```
double m[][]; m = new double[4][4];
m[0][0]=1;
m[1][1] = 1;
m[2][2] = 1;
m[3][3] = 1;
System.out.println(m[0][0] +" "+ m[0][1] +" "+ m[0][2] +" "+ m[0][3]);
```

```
    System.out.println(m[1][0] +" "+ m[1][1] +" "+ m[1][2]
+" "+ m[1][3]);
    System.out.println(m[2][0] +" "+ m[2][1] +" "+ m[2][2]
+" "+ m[2][3]);
    System.out.println(m[3][0] +" "+ m[3][1] +" "+ m[3][2]
+" "+ m[3][3]);
```

Запустив эту программу, получим следующий результат:

1.0 0.0 0.0 0.0

0.0 1.0 0.0 0.0

0.0 0.0 1.0 0.0

0.0 0.0 0.0 1.0

Для работы с массивами в библиотеке классов Java в пакете java.util определен специальный класс Arrays. С его помощью можно производить ряд операций над массивами.

Класс java.util.Arrays содержит методы для работы с целыми массивами, например:

- copyOf() − предназначен для копирования массива;
- copyOfRange() − копирует часть массива;
- toString() − позволяет получить все элементы в виде одной строки;
- sort() – с ортирует массив методом quick sort;
- binarySearch() − ищет элемент методом бинарного поиска;
- fill() − заполняет массив переданным значением;
- equals() − проверяет на идентичность массивы;
- deepEquals() − проверяет на идентичность массивы массивов;
- asList() − возвращает массив как коллекцию.

Массивы часто используют в циклах. Допустим, 5 котов поймали разное количество мышек. Как узнать среднее арифметическое значение:

```
int[] mice = {4, 8, 10, 12, 16 };
int result = 0;
for (int i = 0; i < mice.length; i++) {
        result = result + mice[i];
}
result = result / mice.length; /* общий результат
делим на число элементов в массиве*/
```

```
System.out.println("Среднее арифметическое: " + result);
```

Строка – это упорядоченная последовательность символов.

Стандартная библиотека Java предоставляет два основных класса для работы со строками – String и StringBuffer. Класс String представляет строки, содержимое которых не может изменяться после инициализации. Класс StringBuffer позволяет менять содержимое строки и реализует функциональность для динамически изменяющихся строк. Принципиальная разница между строками, реализованными в виде объектов классов String и StringBuffer, состоит в том, что в первом случае созданная строка изменена быть не может, во втором – может. Под изменением строки в данном случае подразумевается изменение объекта, через который реализована строка. При реализации строки объектом класса String для изменения строки создается новый объект, и ссылка на него присваивается соответствующей объектной переменной. Поскольку при реализации строк объектами класса StringBuffer предусмотрено автоматическое выделение дополнительного буфера для записи текста, в объекты этого класса можно вносить изменения.

Тип String не является примитивным типом данных, однако это один из наиболее используемых типов в Java.

Простейший способ создать строку – присвоить строковой переменной литерал – набор символов, заключенный в двойные кавычки:

```
String s = "the string is created as literal";
```

В этом случае не нужно использовать оператор new, хотя с аналогичным успехом для создания строки можно воспользоваться стандартным способом.

Самый правильный способ создать объект с точки зрения ООП – это вызвать его конструктор в операции

```
String s = new String("the string is created by constructor");
```

Экземпляры только этого класса можно создавать без использования ключевого слова new. Каждый строковый литерал порождает экземпляр String, и это единственный литерал (кроме null), имеющий объектный тип. Обе строки, независимо от способа создания являются объектами — экземплярами класса String.

В Java строка не является ни символом, ни массивом символов в том понимании, как это принято в большинстве языков программирования, это просто еще один тип данных (хотя, конечно, тесно связанный с символами). Поэтому на запись:

```
char ch = 'a';
String s;
s = ch; //1
```

компилятор выдаст сообщение о несоответствии типов в последней строке.

В классе String существует масса полезных методов, которые можно применять к строкам:

- int length() – возвращает длину строки (количество символов в ней);
- boolean isEmpty() – проверяет, пустая ли строка;
- String replace(a, b) – возвращает строку, где символ a (литерал или переменная типа char) заменён на символ b;
- String toLowerCase() – возвращает строку, где все символы исходной строки преобразованы к строчным;
- String toUpperCase() – возвращает строку, где все символы исходной строки преобразованы к прописным;
- boolean equals(s) – возвращает истину, если строка к которой применён метод, совпадает со строкой s указанной в аргументе метода (с помощью оператора == строки сравнивать нельзя, как и любые другие объекты);
- int indexOf(ch) – возвращает индекс символа ch в строке (индекс – это порядковый номер символа, но нумероваться символы начинают с нуля). Если символ совсем не будет найден, то возвратит -1. Если символ встречается в строке несколько раз, то возвратит индекс его первого вхождения.
- int lastIndexOf(ch) – аналогичен предыдущему методу, но возвращает индекс последнего вхождения, если символ встретился в строке несколько раз.
- int indexOf(ch,n) – возвращает индекс символа ch в строке, но начинает проверку с индекса n (индекс это порядковый номер символа, но нумероваться символы начинают с нуля).
- char charAt(n) – возвращает код символа, находящегося в строке под индексом n.

```
String str ="Последний символ в этой строке - W";
int last = str.length()-1;/*длина строки - 1, так как отсчет начинается с 0*/
```

```
char ch = str.charAt(last);
System.out.println(ch);
```

- public int compareTo(String s) – сравнивает две строки лексикографически. Возвращает значение меньше нуля, если строка, вызывающая метод, меньше аргумента метода, ноль – если строки равны, больше нуля, если текущая строка больше аргумента. Сравнение основано на значениях Unicode для символов.

- public boolean equals(Object anObject) – сравнивает текущую строку с объектом anObject. Результатом будет true только в том случае, если аргумент не равен null и является строкой String, представляющую ту же последовательность символов, что и текущая строка.

```
String str1 = "www.tusur.ru";
String str2 = "WWW.tusur.tu";
System.out.println(str1.equals(str2)); //false
```

Причина, по которой для сравнения строк нельзя использовать операторы сравнения «равно» (==) и «не равно» (!=), думается очевидна, однако на ней стоит все же остановиться. Вообще говоря, эти операторы использовать можно, но результат может быть несколько неожиданным.

```
String str1=new String("Алексей");
String str2="Алексей";
System.out.println(str1.equals(str2)); //true
System.out.println(str1==str2); //false
```

Если для сравнения использовать команду str1==str2 или str1!=str2, то в соответствии с этими командами сравниваются значения объектных переменных str1 и str2, а не текстовое содержание объектов, на которые эти переменные ссылаются. Если же переменные ссылаются на разные объекты, значение выражения равно false. При этом разные объекты могут иметь одинаковые текстовые значения.

Если сравнивать переменные str1 и str2 с помощью оператора сравнения «равно» (командой strA==strB), получаем значение false, поскольку переменные ссылаются на разные объекты.

- public byte[ ] getBytes() – преобразует строку в массив байтов используя платформенно-независимую таблицу кодов.

- public int indexOf(String s) – возвращает индекс первого вхождения подстроки s в текущую строку, если подстроки s в текущей строке нет, то возвращается значение –1. Метод перегружен для работы с символами и дополнительными параметрами.

- public integer length() – возвращает длину строки. Заметьте, что в отличие от массивов, здесь длина строки определяется посредством *метода*, а не члена класса.

```
String str = "Строка из букв, цифр 123 и специальных символов %*;№?";
int length = str.length();
System.out.println("Длина строки = " + length);
```

- public String substring(int beginIndex, int endIndex) – возвращает новую строку - подстроку текущей строки, начинающуюся с символа в позиции beginIndex текущей строки и заканчивающуюся символом в позиции endIndex-1.

```
String s = "www.tusur.ru";
String name = s.substring(4, s.length()-3);
System.out.println(name);    /*на    консоль    выведет "tusur"*/
String domain = s.substring(4);
System.out.println(domain);    /*на    консоль    выведет "tusur.ru"*/
```

- public static String valueOf(переменная примитивного типа) – возвращает строковое представление параметра. Метод перегружен для различных типов параметра.

- String[] split(String regex) – разбиения строк на части используется метод String[] split(String regex), который разбивает строку на основании заданного регулярного выражения.

- char[] toCharArray() – преобразует строку в новый массив символов.

- boolean contains(CharSequence s) – проверяет, содержит ли строка заданную последовательность символов и возвращает true или false.

```
String s = "www.tusur.ru";
boolean isContain1 = s.contains("tusur");
System.out.println(isContain1);//нашел - выведет true
boolean isContain2 = s.contains("WWW");
System.out.println(isContain2);/*не  нашел  -  выведет false*/
```

- toLowerCase() – преобразовать строку в нижний регистр;
- toUpperCase() – преобразовать строку в верхний регистр;
- trim() – отсечь на концах строки пустые символы;

```
String str = " ТУСУР – чемпион! ";
//уберем символы пробела в начале и конце строки
str = str.trim();
System.out.println(str.toLowerCase());
System.out.println(str.toUpperCase());
```

- String replace(char oldChar, char newChar), replace(CharSequence target, CharSequence replacement) – замена в строке одного символа или подстроки на другой символ или подстроку.

```
String str = "1 000 000 000";
String newStr = str.replace(" ", "."); /*заменим пробел на точку*/
System.out.println(newStr);
```

Для конкатенации (соединения) строк есть метод concat(String s), но удобнее пользоваться оператором «+». Объекты строковых классов являются единственными непримитивными данными, к которым применим этот оператор:

```
String s1 = "Vasya";
String s2 = "Ivanov";
String s = s1+s2;
```

Порядок выполнения работы

1. Выполнить работу согласно варианту.
2. Написать отчет.
3. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Дано предложение (вводит пользователь). Определить, есть ли буква *а* в нем. В случае положительного ответа найти также порядковый номер первой и последней из них.
2. Дан массив целых чисел (сгенерировать случайным), найти среди элементов массива числа, которые делятся на 3. Сформировать из этих чисел новый массив и вывести его на консоль.
3. Дан двумерный массив, содержащий отрицательные и положительные числа (сгенерировать случайным). Выведете на экран номера тех ячеек массива, которые содержат отрицательные числа.
4. Даны два слова (вводит пользователь). Напечатать только те буквы слов, которые встречаются в обоих словах только один раз.

5. Дано предложение (вводит пользователь). Найти длину его самого короткого слова.

6. Дан массив целых чисел. а) Все элементы, кратные числу 10, заменить нулем. б) Все нечетные элементы удвоить, а четные уменьшить вдвое. в) Нечетные элементы уменьшить на m, а элементы с нечетными номерами увеличить на n.

7. Из элементов массива m сформировать массив n того же размера по правилу: неотрицательные элементы массива m уменьшить в три раза, остальные — возвести в квадрат. Для заполнения массива m использовать генератор случайных чисел.

8. Дана строка из нескольких слов. Слова отделяются друг от друга пробелами или запятыми. Вывести все слова, длина которых меньше заданного числа. Число вводить пользователь с консоли.

9. Дан массив действительных чисел, размерность которого N. Подсчитать, сколько в нем отрицательных, положительных и нулевых элементов. N задает пользователь, значения массива генерируются датчиком случайных чисел.

10. Дана целочисленная квадратная nxn матрица (сгенерировать случайным). Найти в каждой строке наибольший элемент и поменять его местами с элементом главной диагонали. n задает пользователь, значения массива генерируются датчиком случайных чисел.

Контрольные вопросы

1. Дайте определение понятию «конкатенация строк».
2. Как сравнить значение двух строк?
3. Как перевернуть строку?
4. Как работает сравнение двух строк?
5. Как обрезать пробелы в конце строки?
6. Как заменить символ в строке?
7. Как получить часть строки?
8. Какой метод позволяет выделить подстроку в строке?
9. Какой метод вызывается для преобразования переменной в строку?
10. Что такое массив?
11. Какие виды массивов вы знаете?
12. Как определить размер массива?

2.4 Лабораторная работа «Классы»

Цель работы: изучить структуру программ на основе использования объектно-ориентированного программирования, основные элементы структуры ООП: классы, объекты.

Теоретические основы

В объектно-ориентированном программировании базовыми единицами программ и данных являются объекты. (Можно сказать, что в чисто объектно-ориентированной системе ничего, кроме объектов нет).

Объект – это осязаемая сущность, которая четко проявляет свое поведение.

Термин «объект» в программном обеспечении впервые был введен в языке Simula и применялся для моделирования реальности.

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

С точки зрения восприятия человеком объектом может быть:

- осязаемый и (или) видимый предмет;
- нечто, воспринимаемое мышлением;
- нечто, на что направлена мысль или действие.

Процесс представления предметной области в виде совокупности объектов, обменивающихся сообщениями, называется *объектной декомпозицией*.

Объект обладает *состоянием, поведением* и *идентичностью*; структура и поведение схожих объектов определяет общий для них класс; термины «экземпляр класса» и «объект» взаимозаменяемы.

Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств.

Поведение – это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

Всякая программа, написанная на языке Java, представляет собой один или несколько классов, в этом простейшем примере только один *класс* (class).

Класс представляет набор объектов, которые обладают общей структурой и одинаковым поведением. Формально *класс* – шаблон поведения объектов определенного типа с определенными параметрами, определя-

ющими состояние. Все экземпляры одного класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств и общее поведение, одинаково реагируют на одинаковые сообщения.

Класс изображается в виде прямоугольника, состоящего из трех частей (рис. 2.17).

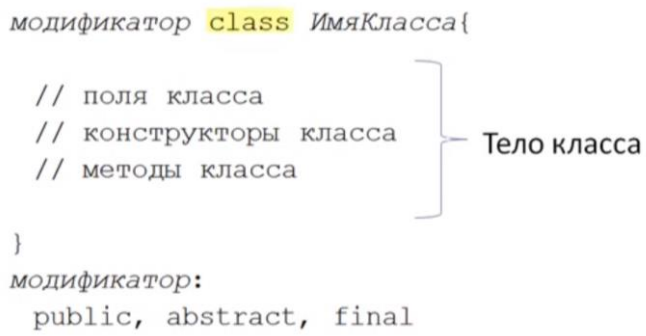

Рис. 2.17 – Структура класса

В Java имя класса также является именем нового *ссылочного типа*.

Начало класса отмечается служебным словом class, за которым следует имя класса, выбираемое произвольно, в данном случае это имя HelloWorld. Все, что содержится в классе, записывается в фигурных скобках и составляет тело класса (*class body*).

Поле (атрибут) класса – это характеристика объекта.

Объявление полей начинается с перечисления модификаторов. Возможно применение любого из трех модификаторов доступа, либо никакого вовсе, что означает уровень доступа по умолчанию.

Все действия в программе производятся с помощью *методов* обработки информации, коротко говорят просто метод (*method*). Методы используются в объектно-ориентированных языках вместо функций, применяемых в процедурных языках.

Все, что содержит метод, тело метода (*method body*), записывается в фигурных скобках. При обращении к полям и методам объекта в Java используется только оператор «точка».

Имя файла должно в точности совпадать с именем класса, содержащего метод main(). Данное правило очень желательно выполнять. При этом система исполнения Java будет быстро находить метод *main*() для начала работы, просто отыскивая класс, совпадающий с именем файла. Расширение имени файла должно быть java.

Перегрузка методов

Часто одно и то же слово имеет несколько разных значений – оно *перегружено*. Также и с методами. Можно создавать методы с одинаковыми именами, но с разным набором аргументов (*сигнатурой*).

Сигнатурой метода называются его имя и набор параметров. Java позволяет создавать несколько методов с одинаковыми именами, но разными сигнатурами.

Создание метода с тем же именем, но с другим набором параметров называется *перегрузкой*.

Перегрузка методов реализует такое важное свойство в программировании, как полиморфизм.

Например, создадим класс Cat, в котором опишем перегрузку метода eat():

```
class Cat {
    void eat() {
        // параметры отсутствуют
    }
    void eat(int count) {
        // используется один параметр типа int
    }
    void eat(int count, int pause) {
        // используются два параметра типа int
    }
    long eat(long time) {
        // используется один параметр типа double
        return time;
    }  }
```

Вы можете вызвать любой метод из класса:

```
Cat cat = new Cat(); // cat – объект класс Cat
cat.eat();
cat.eat(3);
cat.eat(3, 2);
cat.eat(4500.25);
```

Если присмотреться, то можно догадаться, какая именно версия метода вызывается в каждом конкретном случае.

Аналогично, перегрузка используется и для конструкторов.

Переопределение методов

Кроме перегрузки существует также *замещение*, или *переопределение методов* (*overriding*).

Замещение происходит, когда класс потомок (подкласс) определяет некоторый метод, который уже есть в родительском классе(суперклассе), таким образом новый метод заменяет метод суперкласса. У нового метода подкласса должны быть те же параметры или сигнатура, тип возвращаемого результата, что и у метода родительского класса.

Начинается метод с загадочной конструкции @Override. Называется эта конструкция «аннотация». Служит для включения дополнительной информации, которую можно прочитать и использовать. Аннотации появились только в версии Java 1.5 и более ранние версии их не поддерживают.

```
public class Cat // родительский класс
{
   public String name;
   public int age;
   public void voice(String name) {
          this.name=name;
        System.out.println(name+"! Мяу");
    }
}
public class Kitten extends Cat /*Kitten – наследник
Cat*/
{
   public String name;
//переопределение родительского метода voice()
@Override
    public void voice(String name) {
          this.name="My kitty"+name;
        System.out.println(name+"! Мяв-мяв");
        super.voice(name);   /* вызов версии метода ро-
дительского класса*/
    }
   public static void main(String[] args) {
          Kitten cat1= new Kitten();
```

```
            cat1.voice("Мася");//дочерний метод
            Cat cat2= new Cat();
            cat2.voice("Василий");//родительский метод
    }
}
```

Вывод на консоль:

```
Мася! Мяв-мяв
Мася! Мяу
Василий! Мяу
```

Переопределение метода происходит только тогда, когда имена и сигнатуры типов этих двух методов идентичны. Если это не так, то оба метода просто перегружены.

Поля нельзя переопределить, их можно только скрыть. Если пометить метод модификатором *final*, то метод не может быть переопределен. Иногда требуется, чтобы методы были не определены, а только объявлены (т.е. не была бы представлена реализация метода). Такие методы могут быть реализованы в дочерних классах.

Конструкторы

Каждый класс может также иметь специальные методы, которые автоматически вызываются при создании и уничтожении объектов этого класса:

- конструктор (constructor) – выполняется при создании объектов;
- деструктор (destructor) – выполняется при уничтожении объектов.

Как и в C++, в классах Java имеются конструкторы. Их назначение полностью совпадает с назначением аналогичных методов C++. В отличие от C++ в языке Java предусмотрен единственный способ распределения памяти - оператором new. В отношении выделения блоков памяти во многом действуют те же правила, что и в C++.

Конструктор – это особенный метод класса, который вызывается автоматически в момент создания объектов этого класса. Имя конструктора совпадает с именем класса. Конструкторы добавляются в класс, если в момент создания объекта нужно выполнить какие-то действия (начальную настройку) с его данными (полями), т.е. *конструктор* предназначен для *инициализации объекта*.

Аналогично, в одном классе допускается любое количество конструкторов, если у них различные *сигнатуры*.

Конструкторы добавляются в класс, если в момент создания объекта нужно выполнить какие-то действия (начальную настройку) с его данными (полями).

Класс обязательно должен иметь конструктор, иначе невозможно порождать объекты ни от него, ни от его наследников. Поэтому если в классе не объявлен ни один конструктор, компилятор добавляет один по умолчанию. Это public-конструктор без параметров и с телом, описанным парой пустых фигурных скобок. Автоматический вызов *конструктора* позволяет избежать ошибок, связанных с использованием неинициализированных переменных. Этот конструктор не имеет параметров, все что он делает – это вызывает конструктор без параметров класса-предка.

Например, в классе Cat может быть конструктор с двумя параметрами, который при создании новой кошки позволяет сразу задать ее кличку и возраст.

```
class Cat {
int age; // возраст
String name;// кличка
   public Cat (String n,int a)
   {
   name = n;
   age = a;
   }
public void eat ()
{
       for (int i = 1; i <= age; i++) {
              System.out.println("ам-ам");
} }  }
```

Конструктор вызывается после ключевого слова **new** в момент создания объекта. Теперь, когда у нас есть такой конструктор, мы можем им воспользоваться:

```
Cat cat = new Cat("Мася", 2);
```

В результате переменная *cat* будет указывать на «кошку» по кличке Мася, которой 2 года.

Если в классе не определен конструктор без аргументов (но определен хотя бы один конструктор), рассчитывать на конструктор по умолчанию (конструктор без аргументов) нельзя – необходимо передавать аргументы в соответствии с описанным вариантом конструктора.

Порядок выполнения работы

1. Выполнить работу согласно варианту.

2. Написать отчет по лабораторной работе.

3. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Определить класс Квадратное уравнение. Класс должен содержать несколько конструкторов. Реализовать методы вывода уравнения на экран, методы определения и получения коэффициентов уравнения.

2. Определить класс Matrix размерности (nxn). Класс должен содержать несколько конструкторов. Реализовать методы вывода матрицы на экран, методы определения и получения размерности матрицы и самой матрицы.

3. Определить класс «прямоугольный треугольник» заданный длинами сторон». Предусмотреть возможность операции присваивания, определения площади и периметра, а также логический метод, определяющий существует или такой треугольник. Конструктор должен позволить создавать объекты без и с начальной инициализацией.

4. Создать класс «комната», имеющая площадь. Определить конструктор и метод доступа. Создать класс однокомнатных квартира, содержащий комнату и кухню (ее площадь), этаж (комната содержится в классе однокомнатная квартира). Определить конструкторы, методы доступа.

5. Определить класс «цветная точка». Для точки задаются координаты и цвет. Цвет описывается с помощью трех составляющих (красный, зеленый, синий). Предусмотреть различные методы инициализации объекта с проверкой допустимости значений. Допустимым диапазоном для каждой составляющей является [0… 255]. Описать свойства для получения состояния объекта и метод изменения цвета. Определить конструкторы и функцию печати.

6. Определить класс Polynom степени n. Написать несколько конструкторов. Реализовать методы вывода полинома на экран, методы определения и получения коэффициентов полинома и самого полинома.

7. Определить класс Vector размерности n. Класс должен содержать несколько конструкторов. Реализовать методы вывода вектора на экран, методы определения и получения размерности вектора и самого вектора.

8. Определить класс Stack для хранения целых чисел. Класс должен содержать несколько конструкторов. Реализовать методы для добавления, удаления и вывода элементов на экран.

9. Определить класс Булев вектор. Класс должен содержать несколько конструкторов. Реализовать методы вывода вектора на экран, методы определения и получения размерности вектора и самого вектора.

10. Определить класс Complex. Класс должен содержать несколько конструкторов. Реализовать методы для определения и получения коэффициентов комплексного числа, вывода комплексного числа на экран.

Контрольные вопросы

1. Дайте определение понятию «класс».
2. то такое поле/атрибут класса?
3. Как правильно организовать доступ к полям класса?
4. Дайте определение понятию «конструктор».
5. Чем отличаются конструкторы по умолчанию, копирования и конструктор с параметрами?
6. Дайте определение понятию «метод».
7. Что такое сигнатура метода?
8. Какие методы называются перегруженными?
9. Могут ли нестатические методы перегрузить статические?
10. Расскажите про переопределение методов.

2.5 Лабораторная работа «Внутренние и внешние классы»

Цель работы: научиться применять на практике принцип объектно-ориентированного программирования инкапсуляции с использованием специальных механизмов языка Java.

Теоретические основы

Синтаксис языка Java позволяет объявлять классы внутри другого класса, такие классы называются *внутренними* или *вложенными*. *Вложенный класс* не может существовать независимо от класса, в который он вложен. Следовательно, область действия вложенного класса ограничена его *внешним классом*, однако вложенные классы имеют доступ к методам и переменным внешнего.

Вложенные классы делятся на два вида: статические и не статические.

Вложенные классы, объявленные, как статические, называются *вложенными статическими* (static nested classes). Вложенные классы, объ-

явленные без ключевого слова static, называются *внутренними* классами (inner classes).

```
class OuterClass {
// Простой вложенный класс
static class StaticNestedClass {
public void show() {
System.out.println("Метод вложенного класса");
        }
    }
// Простой внутренний класс
class InnerClass {
public void show() {
System.out.println("Метод внутреннего класса");
        }
    }
public static void main(String[] args) {
OuterClass.InnerClass inner = new Outer-
Class().new InnerClass();
OuterClass.StaticNestedClass nestedObject = new
OuterClass.StaticNestedClass();
inner.show();
nestedObject.show();
    }
 }
```

Статический внутренний класс (вложенный) видит только статические переменные внешнего класса. Другие члены внешнего класса доступны ему посредством ссылки на объект.

Как и другие поля класса, вложенные классы могут быть объявлены как private, public, protected, или package private. Внешний класс (OuterClass) может быть только public или package-private!

Доступ к вложенному классу осуществляется с помощью следующей конструкции:

```
OuterClass.StaticNestedClass nestedObject = new
OuterClass.StaticNestedClass();
```

Внутренние (нестатические) классы, как переменные и методы связаны с *объектом* внешнего класса. Внутренние классы так же имеют прямой доступ к полям внешнего класса. Такие классы не могут содержать в себе статические методы и поля. Внутренние классы не могут существовать без экземпляра внешнего. Для создания объекта:

```
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

Если необходимо получить ссылку на объект внешнего класса, то следует использовать наименование внешнего класса с ключевым словом this, разделенных точкой, например, OuterClass.this.

Внутренние классы бывают трех типов в зависимости от того, где и как вы определяете их:

- внутренние;
- локальные;
- анонимный.

В Java, мы можем написать класс в методе, и это будет локальный тип. Как локальные переменные, область внутреннего класса ограничивается в рамках метода.

В следующем примере, внутренний класс расположен в методе outerMethod():

```
public class Outer {
void outerMethod() {
System.out.println("Метод внешнего класса");
/*Внутренний класс является локальным для метода outerMethod()*/
class Inner {
public void innerMethod() {
System.out.println("Метод внутреннего класса");
            }
        }
Inner inner = new Inner();
inner.innerMethod();
    }
    public static void main(String[] args) {
        Outer outer = new Outer();
        outer.outerMethod();
    } }
```

Анонимные внутренние классы (anonymous Inner classes) объявляются без указания имени класса. Они могут быть созданы двумя путями:

1. Как наследник определённого класса.

```
public class Outer {
//Анонимный класс наследуется от класса Demo
static Demo demo = new Demo() {
@Override
 public void show() {
     super.show();
System.out.println("Метод анонимного класса");
        }
    };
public static void main(String[] args) {
 demo.show();
   }
}
class Demo {
public void show() {
   System.out.println("Метод суперкласса");
    }
}
```

Вывод на консоль:

```
Метод суперкласса
Метод внутреннего анонимного класса
```

В коде выше, класс Demo является суперклассом, от которого наследуется анонимный класс, и оба они имеют метод show(). В анонимном классе метод show() будет переопределён.

2. Как реализация определённого интерфейса.

```
public class Outer {
// Анонимный класс, который реализует интерфейс Hello
 static Hello h = new Hello() {
  public void show() {
   System.out.println("Метод анонимного класса");
        }
    };
 public static void main(String[] args) {
```

```
        h.show();
    }
}
interface Hello {
 void show();
}
```

Вывод на консоль:

Метод внутреннего анонимного класса

Любой анонимный внутренний класс может за один раз реализовать только один интерфейс. Так же, за один раз можно либо расширить класс, либо реализовать интерфейс, но не одновременно.

Порядок выполнения работы

1. Выполнить работу согласно варианту.
2. Написать отчет по лабораторной работе.
3. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Создать класс Диагноз с внутренним классом, с помощью объектов которого можно хранить информацию о принятой схеме лечения: лекарственные средства, их дозировка, длительность, лечебные процедуры.
2. Создать класс Студент с внутренним классом Зачетная книжка, с помощью объектов которого можно хранить информацию о сдаче студентом сессии.
3. Создать класс Город с внутренним классом, с помощью объектов которого можно хранить информацию о проспектах, улицах, переулках, площадях.
4. Создать класс Покупка с внутренним классом, с помощью объектов которого можно сформировать покупку из нескольких товаров.
5. Создать класс Расписание для учебной группы с внутренним классом, с помощью объектов которого можно хранить информацию о занятиях – номер аудитории, время, название занятия, преподаватель.
6. Создать класс Мобильный телефон с внутренним классом, с помощью объектов которого можно хранить информацию о вызовах (входящий/исходящий, номер, дата, время, время вызова, пропущен/отклонен).

7. Создать класс Счет с внутренним классом, с помощью объектов которого можно хранить информацию обо всех операциях со счетом (снятие, платежи, поступления).

8. Создать класс Научный сотрудник с внутренним классом, с помощью объектов которого можно хранить информацию обо всех научных темах, разработках конкретного сотрудника.

9. Создать класс Каталог с внутренним классом, с помощью объектов которого можно хранить историю выдачи книги читателям.

10. Создать класс Университет с внутренним классом, с помощью объектов которого можно хранить информацию обо всех факультетах (названия кафедр, деканы, зам. декана, количество обучающихся студентов по кафедрам).

Контрольные вопросы

1. Что означает один из принципов ООП «инкапсуляция»?
2. Какие модификации уровня доступа вы знаете, расскажите про каждый из них.
3. Что такое статический класс, какие особенности его использования?
4. Какие особенности создания вложенных классов: простых и статических.
5. Что вы знаете о вложенных классах, зачем они используются? Классификация, варианты использования, о нарушении инкапсуляции.
6. В чем разница вложенных и внутренних классов?
7. Какие классы называются анонимными?
8. Каким образом из вложенного класса получить доступ к полю внешнего класса?
9. Какие классы называются локальными?
10. О чем говорят ключевые слова this, super, где и как их можно использовать?

2.6 Лабораторная работа «Абстрактные классы и интерфейсы»

Цель работы: научиться применять на практике принципы ООП наследование и полиморфизм с использованием специальных механизмов языка Java. Реализовать абстрактные классы или интерфейсы.

Теоретические основы

По мере изучения особенностей наследования объектов в языке Java может понадобиться создать класс, характеризующий объект обобщенно, на базе которого можно создать впоследствии подклассы.

Самый общий класс в данном случае будет абстрактным; он формирует «внешнюю оболочку» для подклассов, и только подклассы наполняют эту оболочку конкретным содержанием – кодом, реализующим задачи программы.

Абстрактный метод – это метод, реализация которого неизвестная на данный момент. На данный момент известно только то, что этот метод должен быть у всех наследников.

Перед таким абстрактным методом указывается идентификатор *abstract*, а заканчивается описание сигнатуры метода в классе традиционно – точкой с запятой:

```
abstract void method();
```

Абстрактные классы реализуют на практике один из принципов ООП – *полиморфизм*.

Абстрактный класс в силу очевидных причин не может использоваться для создания объектов. Поэтому абстрактные классы являются суперклассами для подклассов. При этом в подклассе абстрактные методы абстрактного суперкласса должны быть определены в явном виде (иначе подкласс тоже будет абстрактным).

Описание абстрактного класса начинается с ключевого слова abstract. Конечно, класс не может быть одновременно abstract и final. Это же верно и для методов. Кроме того, абстрактный метод не может быть private, native, static.

```
abstract class Pet {
    String name;
    int age;
    abstract void voice();
}
class Cat extends Pet {
    void voice() {
   System.out.println("Мяу-мяу");    }   }
class Dog extends Pet {
    void voice() {
   System.out.println("Гав-гав");  }  }
class Fish extends Pet {
void voice() {
   System.out.println("Рыбы не мяукают"); } }
public class Animal {
    public static void main(String[] args) {
```

```
        Cat cat = new Cat();
        Dog dog = new Dog();
        Fish nemo = new Fish();
         cat.voice();
         dog.voice();
         nemo.voice();
}  }
```

Механизм наследования очень удобен, но он имеет свои ограничения. В частности, возможно наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование. Java множественное наследование не поддерживает. В языке Java подобную проблему позволяют решить *интерфейсы*. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

Интерфейсы в Java являются ссылочными типами, как классы, но они могут содержать в себе только константы, сигнатуры методов.

Пример интерфейса:

```
class Cat  {
    void voice() {
        System.out.println("Мяу-мяу");
    }
    void eat() {
        System.out.println("ам-ам");
    }
}
interface Color {
String black="Черный";
String smoky="Дымчатый";
String white="Белый";
}
//Расширение интерфейса
interface Color_cat extends Color {
String getColor();
String getDescription();
}
//Подкласс наследует суперкласс и реализует интерфейсы:
```

```
class Kitten extends Cat implements Color_cat   {
String color;
public String getColor(int number) {
switch(number) {
case 0: this.color=this.black;
break;
case 1: this.color=this.smoky;
break;
case 2: this.color=this.white;
default:
this.color="Черно-белый в полоску"; }
return color; };
public String getDescription() {
return ("Умный котик");
}  }
public class Catt {
    public static void main(String[] args) {
   Kitten cat = new Kitten(); /* создается объект класса Cat, на него ссылается переменная cat*/
       cat.voice();
       System.out.println(cat.getDescription());
       System.out.println(cat.getColor(1));
     }  }
```

Начиная с JDK 8 в интерфейсах доступны статические методы - они аналогичны методам класса. С методом по умолчанию, все иначе: интерфейс может отметить метод ключевым словом default и обеспечить реализацию для него. Например:

```
public interface InterfaceWithDefaultMethods {
 void performAction();
 default void DefaulMethod() {
     // Implementation here
 }
}
```

Порядок выполнения работы

1. Выполнить работу согласно варианту.
2. Написать отчет по лабораторной работе.

3. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Создать абстрактный базовый класс Triangle для представления треугольника с абстрактными методы вычисления площади и периметра. Поля данных должны включать две стороны и угол между ними. Определить производные классы: прямоугольный треугольник, равнобедренный треугольник, равносторонний треугольник с собственными функциями вычисления площади и периметра.

2. Создайте интерфейс FastFood (), его наследник Sandwich и Гамбургер. Реализуйте методы приготовления и методы, позволяющим вывести на экран информацию о товаре (состав фастфуда), а также определить, соответствует ли она сроку годности на текущую дату.

3. Создать интерфейс Worker (работник), содержащий следующие методы: расчет заработной платы работника (принимаемые параметры стаж и должность), вывод информации о работнике. Создать два класса Ingineer (инженер) и Boss (руководитель), реализующих этот интерфейс. Создать группу объектов классов Ingineer и Boss. Создать третий класс SearchInfo, позволяющий выводить информацию о работниках, чья зарплата выше введенной с клавиатуры.

4. Создать интерфейс Animal (животное), содержащий методы: расчет количества корма (принимаемые параметры вес и возраст животного), вывод информации о животном. Создать два класса Dog (собака) и Cat (кошка), реализующих этот интерфейс. Создать группу объектов классов Dog и Cat. Создать третий класс SearchInfo, позволяющий выводить информацию о животных, употребивших наибольшее количество корма, значение которого введено с клавиатуры.

5. Создать абстрактный класс Currency (валюта) для работы с денежными суммами. Определить абстрактный метод конвертации валют. Реализовать производные классы Dollar (доллар) и Euro (евро) с собственными методами перевода и вывода на экран.

6. Создать интерфейс Figure (геометрическая фигура), содержащий следующие методы: расчет площади геометрической фигуры (принимаемые параметры высота и сторона), вывод информации о фигуре. Создать два класса Triangle (треугольник) и Square (квадрат), реализующих этот интерфейс. Создать группу объектов классов Triangle и Square. Создать третий класс SearchInfo, позволяющий выводить информацию о геометрических фигурах, чья площадь больше (меньше) числа введенного с клавиатуры.

7. Создать абстрактный класс Bankomat, моделирующий работу банкомата. В классе должны содержаться поля для хранения идентификационного номера банкомата, информации о текущей сумме денег, оставшейся в банкомате, минимальной и максимальной суммах, которые позволяется снять клиенту в один день. Реализовать метод инициализации банкомата, метод загрузки купюр в банкомат и метод снятия определенной суммы денег.

8. Создать абстрактный класс Production (товар) с методами, позволяющим вывести на экран информацию о товаре, а также определить, соответствует ли она сроку годности на текущую дату. Создать классы хлебобулочные и колбасные изделия, а также организовать поиск просроченного товара (на момент текущей даты).

9. Создать абстрактный базовый класс Body (тело) с абстрактными методами вычисления площади поверхности и объема. Создать производные классы: параллелепипед и конус с собственными функциями площади поверхности и объема. Переопределить метод equals() и toString().

10. Создать интерфейс Plant (растение), содержащий методы: расчет количества воды для полива (принимаемые параметры пора года и возраст растения), вывод информации о растении. Создать два класса Tree (дерево) и Flower (цветок), реализующих этот интерфейс. Создать группу объектов классов Tree и Flower. Создать третий класс SearchInfo, позволяющий выводить информацию о растениях, требующих наибольшее количество воды для полива в зависимости от введенной поры года.

Контрольные вопросы

1. Каков порядок вызова конструкторов и блоков инициализации двух классов: потомка и его предка?

2. Где и для чего используется модификатор abstract?

3. Можно ли объявить метод абстрактным и статическим одновременно?

4. Дайте определение понятия «интерфейс».

5. Какие модификаторы по умолчанию имеют поля и методы интерфейсов?

6. Почему нельзя объявить метод интерфейса с модификатором final или static?

2.7 Лабораторная работа «Коллекции»

Цель работы: изучить способы использования коллекций при разработке java-приложений.

Теоретические основы

В пакете java.util содержится библиотека коллекций (collection), которая предоставляет большие возможности для работы с множествами, хэш-таблицами, векторами, различными списками и т.д.

Коллекция – это хранилища, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним.

Простейшей коллекцией является массив. Но массив имеет ряд недостатков. Один из самых существенных – размер массива фиксируется до начала его использования. То есть мы должны заранее знать или подсчитать, сколько нам потребуется элементов коллекции до начала работы с ней. Зачастую это неудобно, а в некоторых случаях – невозможно. Поэтому все современные базовые библиотеки различных языков программирования имеют тот или иной вариант поддержки коллекций объектов. Коллекции обладают одним важным свойством – их размер не ограничен. Выделение необходимых для коллекции ресурсов спрятано внутри соответствующего класса.

В библиотеке коллекций Java существует два базовых интерфейса, реализации которых и представляют совокупность всех классов коллекций:

1. **Collection** – коллекция содержит набор объектов (элементов). Здесь определены основные методы для манипуляции с данными, такие как вставка (*add, addAll*), удаление (*remove, removeAll, clear*), поиск (*contains*). Интерфейс **Collection** расширяют интерфейсы **List, Set** и **Queue**:

- **List п**редставляет собой упорядоченную коллекцию, в которой допустимы дублирующие значения. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу.

- **Set** описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set).

- **Queue –** очередь. Это коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки. В дополнение к базовым операциям интерфейса Collection, очередь предоставляет дополнительные операции вставки, получения и контроля.

2. **Map** описывает коллекцию, состоящую из пар «ключ – значение». У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения (map). Такую коллекцию часто называют еще словарем (dictionary) или ассоциативным массивом (associative array). Словарь может содержать произвольное число элементов.

Map не наследует интерфейс *Collection*, так как они несовместимы. В интерфейсе *Collection* описан метод *add(Object o)*. Словари не могут содержать этот метод, потому что работают с парами ключ/значение. Также, словари имеют представления *keySet, valueSet,* которых нет в коллекциях. В связи с этими различиями, интерфейс *Map* не может наследовать интерфейс *Collection*, и представляет собой отдельную ветвь иерархии.

Интерфейсы Collection и Map являются базовыми, но они не есть единственными. Их расширяют другие интерфейсы, добавляющие дополнительный функционал.

В таблице 2.2 представлены классы наборов данных.

Таблица 2.2 – Коллекции

Классы	**Описание**
ArrayList	Индексируемая последовательность, размер которой может увеличиваться и уменьшаться.
LinkedList	Упорядоченная последовательность, обеспечивающая эффективное выполнение операций включения или удаления элемента в любой позиции.
HashSet	Неупорядоченный набор, не допускающий дублирования элементов.
TreeSet	Сортированное множество элементов.
EnumSet	Набор значений нумерованногог типа.
LinkedHashSet	Множество, которое помнит порядок, в котором элементы были включены в него.
PriorityQueue	Набор, обеспечивающий эффективное удаление наименьшего элемента.
HashMap	Карта, которая хранит связи ключ/значение.

TreeMap	Карта, в которой ключи отсортированы.
EnumMap	Карта, в которой ключи принадлежат нумерованному типу.
LinkedHashMap	Карта, которая помнит порядок включения элементов в него.
WeakHashMap	Карта, не используемые значения которой могут быть обработаны системой сборки мусора.
IdentityHashMap	Карта, для сравнения ключей которой может быть использована операция ==.

Так как большинство коллекций параметризированны (начиная с Java 5), то принято при описании интерфейсов и классов указывать T, что означает любой класс, которым параметризируете коллекцию. Использование <E> – это указание типа объекта, который коллекция может содержать. Это помогает сократить ошибки времени выполнения с помощью проверки типов объектов во время компиляции.

Интерфейс **Collection** расширяет интерфейс **Iterable**, у которого есть только один метод iterator(). Это значит, что любая коллекция, которая есть наследником **Iterable** должна возвращать итератор.

Объект типа **Iterator** может использоваться для последовательного перебора элементов коллекции.

Интерфейс Iterator имеет следующее определение:

```
public interface Iterator <E>{

    E next();
    boolean hasNext();
    void remove();
}
```

Реализация интерфейса предполагает, что с помощью вызова метода next() можно получить следующий элемент. С помощью метода hasNext() можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции. И если элементы еще имеются, то hasNext() вернет значение true. Метод hasNext() следует вызывать перед методом next(), так как при достижении конца коллекции метод next() выбрасывает исключение NoSuchElementException. И метод remove() удаляет текущий элемент, который был получен последним вызовом next().

Используем итератор для перебора коллекции ArrayList:

```
import java.util.*;
 public class CollectionApp {
     public static void main(String[] args) {
ArrayList<String> cats = new ArrayList<String>();
        states.add("Мася");
        states.add("Василий");
        states.add("Мурзик");
        states.add("Барсик");

        Iterator<String> iter = cats.iterator();
        while(iter.hasNext()){
             System.out.println(iter.next());
        }
    }
}
```

В Java8 у интерфейса Iterable появился метод forEach, принимающий лямбда-выражение и применяющий это выражение на каждый элемент коллекции:

```
Map<String, Integer> cats = new HashMap<>();
cats.put("Васька", 10);
cats.put("Мася", 15);
cats.put("Пушистик", 6);
cats.put("Барсик", 2);
cats.put("Муся", 5);
cats.put("Мышка", 7);
cats.forEach((key, value) -> {
        System.out.println(key + " == " + value);
});
```

Интерфейс Iterator предоставляет ограниченный функционал. Гораздо больший набор методов предоставляет другой итератор – интерфейс ListIterator. Данный итератор используется классами, реализующими интерфейс List, то есть классами LinkedList, ArrayList и др.

Помимо итераторов можно использовать следующие конструкции для перебора элементов:

- цикл с итератором;
- цикл for;
- расширенный цикл for;
- цикл while.

```
List<String> cats = Arrays.asList("Васька", "Муся", "Барсик", "Рыжик", "Пушок");
```

Рассмотрим операцию перебора элементов коллекции в старых версиях Java:

```
for (int i=0; i<cats.length;i++) {
System.out.println(cats.get(i))
}
```

Java также предлагает более современный подход:

```
for (String cat_name :cats) {
System.out.println(cat_name);
}
```

Главные преимущества коллекций:

- Уменьшаются затраты времени на написание кода.
- Улучшается производительность, благодаря использованию высокоэффективных алгоритмов и структур данных.
- Коллекции являются универсальным способом хранения и передачи данных, что упрощает взаимодействие разных частей кода.
- Простота в изучении, потому что необходимо выучить только самые верхние интерфейсы и поддерживаемые операции.
- Реализуется поддержка многопоточного доступа.

Порядок выполнения работы

1. Выполнить работу согласно варианту.
2. Написать отчет по лабораторной работе.
3. Защитить лабораторную работу, ответив на вопросы по теме лабораторной работы и выполнив дополнительные задания.

Варианты задания

1. Дан набор из 10 чисел. Создать две очереди: первая должна содержать числа из исходного набора с нечетными номерами (1, 3, …, 9), а вторая — с четными (2, 4, …, 10); порядок чисел в каждой очереди должен совпадать с порядком чисел в исходном наборе. Вывести указатели на начало и конец первой, а затем второй очереди.

2. Создать очередь, информационными полями которой являются: длины катетов прямоугольного треугольника (два вещественных числа). Добавить в очередь сведения о новом треугольнике. Организовать просмотр данных очереди. Определить периметр треугольника в начале очереди.

3. Выпуклый многоугольник задан на плоскости перечислением координат вершин, сохраненных в списке, в порядке обхода его границы. Определить площадь многоугольника. Предусмотреть хранение, сортировку и обработку информации с использованием возможностей коллекций.

4. Для каждого из множества заданных координатами вершин треугольников определить его тип: разносторонний, равнобедренный, равносторонний или прямоугольный. Предусмотреть хранение, сортировку и обработку информации с использованием возможностей коллекций.

5. Создать стек, информационными полями которого являются: фамилия работника и его оклад. Добавить в стек сведения о новом работника. Организовать просмотр данных стека и вычислить средний оклад.

6. Создать очередь, информационными полями которого являются: книга и её цена. Добавить в очередь сведения о новой книге. Организовать просмотр данных очереди и вычислить среднюю цену книг.

7. Создать стек, информационными полями которого являются: название книги и количество страниц. Добавить в стек сведения о новой книге. Организовать просмотр данных стека и определить количество книг в стеке.

8. Создать стек, информационными полями которого являются: название горы и высота. Добавить в стек сведения о новой горе. Организовать просмотр данных стека и определить среднюю высоту гор.

9. Заданы два множества точек на плоскости. Определить пересечение и разность этих множеств. Предусмотреть хранение, сортировку и обработку информации с использованием возможностей коллекций.

10. Создать очередь из сведений о клиентах банка: фамилии и суммы на счету. Определить количество клиентов банка, у которых сумма на счету больше 10000 руб. Организовать просмотр данных очереди.

Контрольные вопросы

1. Дайте определение понятию «коллекция».
2. Назовите преимущества использования коллекций.
3. Какие данные могут хранить коллекции?
4. Какова иерархия коллекций?
5. Что вы знаете о коллекциях типа List?
6. Что вы знаете о коллекциях типа Set?
7. Что вы знаете о коллекциях типа Queue?
8. Что вы знаете о коллекциях типа Map, в чем их принципиальное отличие?
9. Дайте определение понятию «итератор».
10. Как реализован цикл foreach?
11. Почему нет метода iterator.add() чтобы добавить элементы в коллекцию?
12. Почему в классе iterator нет метода для получения следующего элемента без передвижения курсора?
13. В чем разница между Iterator и ListIterator?

2.8 Лабораторная работа «Потоки»

Цель работы: получение практических навыков по использованию механизмов ввода-вывода.

Теоретические основы

Ввод и вывод данных в Java реализуется через *потоки ввода-вывода* (stream).

Поток данных (stream) представляет из себя абстрактный объект, предназначенный для получения или передачи данных единым способом, независимо от связанного с потоком источника или приемника данных.

Потоки реализуются с помощью классов, входящих в пакет java.io. Потоки делятся на две больших группы – *потоки ввода*, и *потоки вывода*. Потоки ввода связаны с источниками данных, потоки вывода – с приемниками данных. Кроме того, потоки делятся на *байтовые* и *символьные*. Единицей обмена для байтовых потоков является байт, для сим-

вольных – символ Unicode. Чтобы эти классы стали доступными в программе, необходимо подключить (импортировать) пакет java.io.

На вершине иерархии байтовых потоков находятся два абстрактных класса: *InputStream* и *OutputStream*. В этих классах определены методы *read*() и *write*(), предназначенные для чтения данных из потока и записи данных в поток соответственно.

Иерархия классов для символьных потоков ввода-вывода начинается с абстрактных классов *Reader* и *Writer*. В этих классах определены методы *read*() для считывания символьных данных из потока и *write*() для записи символьных данных в поток.

Для преобразования между байтовыми и символьными потоками используются классы-«мосты»: *InputStreamReader* – для преобразования от байтового потока к символьному для чтения данных и *OutputStreamWrite* – от символьного потока к байтовому для записи данных.

Операции ввода/вывода по сравнению с операциями в оперативной памяти выполняются очень медленно. Для компенсации в оперативной памяти выделяется некоторая промежуточная область – буфер, в которой постепенно накапливается информация. Буфер представляет собой контейнер для данных простых типов, таких как *byte, int, float* и др. кроме *boolean*.

Классы *BufferedInputStream, BufferedOutputStream, BufferedReader* и *BufferedWriter* предназначены для буферизации ввода-вывода. Они позволяют читать и записывать данные большими блоками. При этом обмен данными со стороны приложения ведется с буфером, а по мере необходимости в буфер из источника данных подгружается новая порция данных, либо из буфера данные переписываются в приемник данных.

Класс *BufferedReader* имеет дополнительный метод *readLine*() для чтения строки символов, ограниченной разделителем строк. Класс *BufferedWriter* имеет дополнительный метод *newLine*() для вывода разделителя строк.

В отличие от большинства классов ввода/вывода, класс *File* работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов. Методы класса File:

- getAbsolutePath() – абсолютный путь файла, начиная с корня системы. В Android корневым элементом является символ «/»;
- canRead() – доступно для чтения;

- canWrite() – доступно для записи;
- exists() – файл существует или нет;
- getName() – возвращает имя файла;
- getParent() – возвращает имя родительского каталога;
- getPath() – путь;
- lastModified() – дата последнего изменения;
- isFile() – объект является файлом, а не каталогом;
- isDirectory – объект является каталогом;
- isAbsolute() – возвращает *true*, если файл имеет абсолютный путь;
- renameTo(File newPath) – переименовывает файл. В параметре указывается имя нового имени файла. Если переименование прошло неудачно, то возвращается *false;*
- delete() – удаляет файл. Также можно удалить **пустой** каталог/

Часть возможностей ввода-вывода может быть реализована посредством класса *System*. Класс System не предоставляет каких-либо публичных конструкторов, поэтому нельзя создать экземпляр этого класса.

Класс System содержит три переменных потока: in, out и err. Эти поля имеют атрибуты public и static:

- Поле System.out – поток стандартного вывода. По умолчанию он связан с консолью. Поле System.out является объектом класса PrintStream.
- Поле System.in – это поток стандартного ввода. По умолчанию он связан с клавиатурой. Поле является объектом класса InputStream.
- Поле System.err – это стандартный поток ошибок. По умолчанию поток связан с консолью. Поле является объектом класса PrintStream.

Для вывода на консоль ранее использовался метод println() класса PrintStream, никогда не определяя экземпляры этого класса. Можно использовали статическое поле out класса System, которое является объектом класса PrintStream. Исполняющая система Java связывает это поле с консолью.

Для ввода данных используется класс *Scanner* из библиотеки пакетов Java. Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Java представлен объектом – System.in.

```
   import java.util.Scanner;
public class Cat {
```

```
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in); /*создаём
объект класса Scanner*/
        int i;
        System.out.print("Введите возраст кота в виде це-
лого числа: ");
        if(in.hasNextInt()) { /*возвращает истину если с
потока ввода можно считать целое число*/
          i = in.nextInt(); /* считывает целое число с по-
тока ввода и сохраняем в переменную*/
          System.out.println(i);
        } else {
          System.out.println("Вы ввели не целое число");
        }
    }
}
```

Порядок выполнения работы

1. Изучить лекционный материал.

2. Выполнить лабораторную работу согласно варианту.

3. Представить отчет по лабораторной работе для защиты.

Варианты задания

1. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: Записать в другой файл в обратном порядке символы каждой строки.

2. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: В каждом слове длиннее двух символов все строчные символы заменить прописными.

3. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: Удалить из текста все «лишние» пробелы и табуляции, оставив только необходимые для разделения операторов.

4. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: найти номера строк, совпадающих с заданной строкой. Имя файла и строка для поиска – аргументы командной строки. Вывести строки файла и номера строк, совпадающих с заданной.

5. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: выбрать и сохранить m последних слов в каждой из последних n строк.

6. Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: Выделить отдельные слова, разделяемые пробелами. Написать метод поиска слова по образцу-шаблону. Вывести найденное слово в другой файл.

7. Программой по введенной пользователем фамилии производится поиск сотрудника в импровизированной базе данных, представленной в виде текстового файла. Текстовый файл personal.txt, используемый в качестве базы поиска, содержит записи о сотрудниках.

8. Даны два символьных файла f1 и f2. f1 содержит произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Файл f2 содержит не более 40 слов, которые разделены запятыми. Эти слова образуют пары: каждое первое слово считается заменяемым, каждое второе слово – заменяющим. Найти в файле f1 все заменяемые слова и заменить их на соответствующие заменяющие. Результат поместить в файл f3.

9. Написать программу, которая считывает текст из входного файла, подсчитывает, сколько раз встретился каждый символ русского алфавита, и выводит результат в выходной файл, например, в виде строк «символ – число».

10. Написать программу, которая считывает текст из входного файла, формирует множество слов и выводит результат в выходной файл. Одинаковые слова, встретившиеся в тексте, нужно вывести в третий файл в виде строк «слово – число».

Контрольные вопросы

1. Какие существуют виды потоков ввода/вывода?

2. Что общего и чем отличаются следующие потоки: InputStream, OutputStream, Reader, Writer?

3. Как известно, консольные операции ввода-вывода осуществляются в текстовом виде. Почему же в Java для этой цели используются байтовые потоки?

4. Как открыть файл для чтения байтов?

5. Как открыть файл для чтения символов?

6. Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?

7. Экземпляром какого класса является поле System.in?

2.9 Лабораторная работа «Исключительные ситуации»

Цель работы: получение практических навыков по использованию механизмов обработки исключительных ситуаций.

Теоретические основы

Исключительные ситуации (exceptions) могут возникнуть во время *выполнения* (runtime) программы, прервав ее обычный ход. К ним относится деление на нуль, отсутствие загружаемого файла, отрицательный или вышедший за верхний предел индекс массива, переполнение выделенной памяти и др., которые могут случиться в самый неподходящий момент. Когда исключительная ситуация возникает, создается объект, представляющий исключение, который возбуждается в методе, вызвавшем ошибку. Этот метод может либо обработать исключение самостоятельно, либо пропустить его. В обоих случаях, в некоторой точке исключение перехватывается и обрабатывается.

Конечно, можно предусмотреть такие ситуации и использовать условный оператор:

```
if (){  // Предпринимаем аварийные действия
      }else{ // Обычный ход действий
   }
```

Но при этом много времени уходит на проверки, и программа превращается в набор этих проверок. Что не есть хорошо.

Обработка исключительных ситуаций (exception handling) – механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма.

Обработка исключений – это встроенная возможность языка Java. Концепция обработки исключений позволяет сделать код более надежным и позволяет лучше читать и сопровождать его. Исключения делятся на несколько классов, но все они имеют общего предка – класс *Throwable*. Его потомками являются подклассы *Exception* и *Error*.

Ошибки (Errors) представляют собой более серьёзные проблемы, которые, согласно спецификации Java, не следует пытаться обрабатывать в собственной программе, поскольку они связаны с проблемами уровня JVM. Например, исключения такого рода возникают, если закончилась память, доступная виртуальной машине. Программа дополнительную память всё равно не сможет обеспечить для JVM.

В Java все исключения делятся на три типа: *контролируемые* исключения (checked) и *неконтролируемые* (unchecked), к которым относятся ошибки (Errors) и исключения времени выполнения (RuntimeExceptions, потомок класса Exception).

1. *Checked* исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода.

2. Unchecked могут не обрабатываться и не быть описаны.

Unchecked исключения в Java – наследованные от RuntimeException, checked – от Exception (не включая unchecked).

Контролируемые исключения представляют собой ошибки, которые можно и нужно обрабатывать в программе, к этому типу относятся все потомки класса Exception (но не RuntimeException).

В Java есть пять ключевых слов для работы с исключениями:

1. try – данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке.

2. catch – ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.

3. finally – ключевое слово для отметки начала блока кода, которое является дополнительным. Этот блок помещается после последнего блока 'catch'. Управление обычно передаётся в блок finally в любом случае.

4. throw – служит для генерации исключений.

5. throws – ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом.

Откомпилируем и запустим такую программу:

```
class Main {
     public static void main(String[] args) {
          int a = 4;
          System.out.println(a/0);
     }  }
```

В момент запуска на консоль будет выведено следующее сообщение:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
     at Main.main(Main.java:4)
```

Из сообщения виден класс случившегося исключения – ArithmeticException. Это исключение можно обработать:

```
class Main {
     public static void main(String[] args) {
          int a = 4;
```

```
        try {
             System.out.println(a/0);
        } catch (ArithmeticException e) {
System.out.println("Произошла недопустимая ариф-
метическая операция");
        }
     }
}
```

Если реализация метода предусматривает выброс исключения, это должно быть описано в объявлении метода.

Для этого в Java можно поступить следующим образом:

```
int calculate(int a, int b) {
    if (b != 0) {
        return a / b;
    } else
        throw new RuntimeException();
    }
}
```

В этот момент создастся новый объект типа RuntimeException. Бросая его (throw), функция прерывается (никаких значений она при этом вообще не возвращает), прерывается и та функция, которая ее вызвала и т.д.

Иногда исключения нецелесообразно обрабатывать в том методе, в котором они возникают. В таком случае их следует указывать с помощью ключевого слова throws. Ниже приведена общая форма объявления метода, в котором присутствует ключевое слово throws. Ключевое throws указывает на то, что метод может сгенерировать (выбросить) одно из перечисленных в списке исключений.

Пример

```
import java.util.*;
import java.io.*;
public class Main {
public static void main(String[] args)
{
FileInputStream f = new FileInputStream("input.txt");
        }
}
```

В момент запуска на консоль будет выведено следующее сообщение:

Ошибка компиляции time: 0.04 memory: 711168 signal:-1

Main.java:15: error: unreported exception FileNotFoundException; must be caught or declared to be thrown FileInputStream f = new FileInputStream(“input.txt");

1 error

Встроенные в Java исключения позволяют обрабатывать большинство распространенных ошибок. Тем не менее в прикладных программах возможны особые ситуации, требующие наличия и обработки соответствующих исключений.

Для того чтобы создать класс собственного исключения, достаточно определить его как производный от класса Exception, который, в свою очередь, является производным от класса Throwable. В подклассах собственных исключений совсем не обязательно реализовать что-нибудь. Их присутствия в системе типов уже достаточно, чтобы пользоваться ими как исключениями.

```
    /* тестовый метод создания, обработки и пробрасывания исключения*/
    import java.io.*;
     class ArithmeticExp extends Exception{
        private int number;
        public int getNumber(){return number;}
        public ArithmeticExp(String message, int num){
            super(message);
            number=num;
        }   }
    public class Main {
            public static int get_d(int x, int y) throws ArithmeticExp{
            int result=1;
    if(y==0) throw new ArithmeticExp("Знаменатель не может равен 0", y);
            result=x/y;
            return result;
        }
           public static void main(String[] args) throws IOException {
    int a=5;
    int b=0;
```

```
try{
    int result = get_d(a,b);
    System.out.println(result);    }
catch(ArithmeticExp ex){
    System.err.println(ex.getMessage());
    System.err.println(ex.getNumber());
}  }   }
```

В момент запуска на консоль будет выведено следующее сообщение:

Знаменатель не может равен 0

0

Здесь для определения ошибки, связанной с делением на ноль, определен класс ArithmeticExp, который наследуется от Exception и который содержит всю информацию о вычислении. В конструкторе ArithmeticExp в конструктор базового класса Exception передается сообщение об ошибке: super(message)**.** Кроме того, отдельное поле предназначено для хранения знаменателя.

Для генерации исключения в методе вычисления дроби выбрасывается исключение с помощью оператора throw**:** throw new ArithmeticExp ("Знаменатель не может равен 0", y). Кроме того, так как это исключение не обрабатывается с помощью try...catch, то мы передаем обработку вызывающему методу, используя оператор throws: public static int get_d(int x, int y) throws ArithmeticExp.

Порядок выполнения работы

1. Изучить лекционный материал.

2. Выполнить лабораторную работу согласно варианту.

3. Представить отчет по лабораторной работе для защиты.

Варианты заданий

1. Даны числа a, b (0 < a < b) и набор из десяти элементов. Найти минимальный и максимальный из элементов, содержащихся в интервале (a, b). Для обработки исключения необходимо организовать класс исключения: ошибочный ввод числа, выходящего за пределы определенного диапазона и если требуемые элементы отсутствуют.

2. Дан файл f, компоненты которого являются целыми числами (положительные и отрицательные, но не равны 0). Файл f содержит столько же отрицательных чисел, сколько и положительных. Используя вспомогательный файл h, переписать компоненты файла f в файл g так, чтобы в файле g не было двух соседних чисел с одним знаком. Обрабо-

тать исключения: ошибки чтения и записи файла и файл не существует. Для обработки исключения необходимо организовать соответствующий класс.

3. Дан файл, содержащий различные даты. Каждая дата - это число, месяц и год. Проверкой допустимость значений и найти самую раннюю дату с. В случае недопустимых значений полей выбрасываются исключения. Обработать исключения: ошибки чтения и записи файла и файл не существует. Для обработки исключения необходимо организовать соответствующий класс.

4. Составить описание класса для представления времени. Предусмотреть возможности установки времени и изменения его отдельных полей (час, минута, секунда) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения. Создать методы изменения времени на заданное количество часов, минут и секунд.

5. Ввести массив, состоящий из 20 элементов целого типа меньше 0. Создать два новых массива: в первый записать элементы исходного массива, которые больше по модулю 5, а во второй – остальные. Определить, в каком массиве меньше сумма элементов. Обработать исключения: ошибочный ввод положительного или нулевого значения вместо отрицательного, а также выход за границы массива. Для обработки исключения необходимо организовать соответствующий класс.

6. Дан файл f, компоненты которого являются целыми числами (положительные и отрицательные). Файл f содержит столько же отрицательных чисел, сколько и положительных и такое же количество 0. Используя вспомогательный файл h, переписать компоненты файла f в файл g так, чтобы в файле g сначала шли 0, потом положительные и далее отрицательные числа. Обработать исключения: ошибки чтения и записи файла и файл не существует. Для обработки исключения необходимо организовать соответствующий класс.

7. Описать класс, представляющий треугольник. Предусмотреть методы для создания объектов, перемещения на плоскости, изменения размеров и вращения на заданный угол. Описать свойства для получения состояния объекта. При невозможности построения треугольника выбрасывается исключение.

8. Реализовать класс Вектор. Размер вектора ограничен значением MAX_SIZE=30. Реализовать методы доступа по индексу ([int i]), добавление элемента, удаление элемента из начала вектора. Предусмотреть генерацию исключительных ситуаций.

9. Построить описание класса, содержащего информацию о почтовом адресе организации (индекс, область, город, район, улица, дом). Предусмотреть возможность раздельного изменения составных частей адреса и проверки допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения

10. Составить описание класса для представления даты. Предусмотреть возможности установки даты и изменения ее отдельных полей (год, месяц, день) с проверкой допустимости вводимых значений. В случае недопустимых значений полей выбрасываются исключения. Создать методы изменения даты на заданное количество дней, месяцев и лет.

Контрольные вопросы

1. Дайте определение понятию «исключение».
2. Какова иерархия исключений?
3. Можно/нужно ли обрабатывать ошибки jvm?
4. Какие существуют способы обработки исключений?
5. О чем говорит ключевое слово throws?
6. В чем особенность блока finally? Всегда ли он исполняется?
7. Может ли не быть ни одного блока catch при отлавливании исключений?
8. Может ли один блок catch отлавливать несколько исключений (с одной и разных веток наследований)?
9. Что вы знаете об обрабатываемых и не обрабатываемых (catched/uncatched) исключениях?
10. В чем особенность RuntimeException?

2.10 Лабораторная работа «Графика»

Цель работы: познакомиться с основными классами и интерфейсами пакетов java.awt или Swing, научиться создавать пользовательский интерфейс, используя стандартные компоненты.

Теоретические основы

Графический интерфейс пользователя (Graphical user interface, GUI) – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т.п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

В Java для создания графического интерфейса обычно используются библиотеки AWT и Swing.

В библиотеке AWT универсальность программного кода обеспечивается за счет использования разных инструментальных средств с целью реализации программного кода для разных операционных систем. Удобство такого подхода состоит в том, что на разных системах программы работают одинаково и имеют единый программный интерфейс. Однако такой подход имеет и недостаток: он применим только при написании относительно небольших программ. Кроме того, при тестировании программного кода нередко оказывается, что один и тот же код генерирует разные ошибки для разных операционных систем. Решить эти фундаментальные проблемы призвана библиотека Swing. При этом библиотека Swing не заменяет библиотеку AWT, а дополняет ее. Механизм обработки событий в библиотеке Swing тот же, что и в AWT.

AWT (Abstract Window Toolkit, java.awt) – набор классов-оберток компонентов GUI операционной системы, на которой выполняется Java-приложение:

- присутствует во всех реализациях Java;
- адекватная для многих приложений;
- трудно построить понятный интерфейс.

Фрейм (Frame) – окно, которое является независимым от апплета и от браузера. Он может использоваться как контейнер или как компонент. Он может быть создан, используя конструкторы следующим образом:

- Frame() – создаёт фрейм, который является невидимым
- Frame(String title) – создаёт невидимый фрейм с данным заголовком, как создавать фрейм.

При создании окна необходимо предусмотреть возможность для этого окна реагировать хоть на какие-то события, в частности на попытку это окно закрыть. Для этого создается специальный обработчик события закрытия окна. С помощью метода addWindowListener() ссылка на этот обработчик добавляется в класс, реализующий окно.

```
import java.awt.*;
import java.awt.event.*;
class GUI extends Frame {
public GUI(String title) {
super(title);   }
public static void main (String args[]) {
GUI f=new GUI("I have been Framed!!!") ;
f.setSize(300,300);
```

```
f.setVisible(true);
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent ev){
System.exit(0); } });
} }
```

Класс *Panel* является суперклассом для новых контейнеров с особой работой с вложенными компонентами. Впрочем, поскольку Panel класс не абстрактный, его можно использовать для иерархической организации сложного пользовательского интерфейса, группируя компоненты в такие простейшие контейнеры. Самый простой способ создания Панели – через ее конструктор *Panel()*.

Пример ниже показывает, как создать панель.

```
import java.awt.*;
import java.awt.event.*;
class GUI extends Panel {
public static void main(String args []) {
GUI p= new GUI();
Frame f=new Frame("Testing a Panel!");
f.add(p);
f.setSize (400,400);
f.setVisible(true);
f.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent ev){
System.exit(0); } });
} }
```

Перед использованием управляющих компонентов (например, кнопок) их надо расположить на форме в нужном порядке. Вместо ручного расположения применяются *менеджеры размещения*, определяющие способ, который панель использует для задания порядка размещения управляющего элемента на форме. Менеджеры размещения контролируют, как выполняется позиционирование компонентов, добавляемых в окна, а также их упорядочение. Если пользователь изменяет размер окна, менеджер размещения переупорядочивает компоненты в новой области так, чтобы они оставались видимыми и в то же время сохранили свои позиции относительно друг друга.

Менеджер размещения представляет собой один из классов FlowLayout, BorderLayout, GridLayout, CardLayout, BoxLayout, реализующих интерфейс LayoutManager, устанавливающий размещение.

Процедура для упорядочивания размещения должна быть следующая:

1. Создайте элементы GUI.

2. Установите их индивидуально или установите желательную схему размещения.

Всеми размещениями осуществляет интерфейс LayoutManager. Менеджер размещения автоматически устанавливает компоненты в пределах контейнера. Рассмотрим различные размещения, доступные в Java:

- FlowLayout – менеджер, используемый по умолчанию, размещает компоненты последовательно в строку. Его использование оправдано в том случае, когда Вы знаете точные размеры компонент.

- BorderLayout – создает так называемое полярное расположение: разбивает панель на 5 зон (South, North, Center, West, East). Он учитывает разницу в размерах отдельных компонентов и пытается максимально использовать пространство контейнеров.

- GridLayout – создает решетку, состоящую из прямоугольников одинакового размера, в каждом из которых располагается один компонент.

- CardLayout – предназначен для последовательной визуализации различных панелей на одной основный.

- GridBagLayout – данный менеджер является наиболее сложным. Он позволят реализовывать сложный интерфейс, в котором контейнер содержит много компонентов различных размеров, которые должны находиться в одном и том же заданном положении относительно других. В приведенном ниже примере рассматриваются все описанные выше компоненты.

Внешний вид компонентов AWT определяется не средствами Java, а платформой. Поскольку в компонентах AWT используются ресурсы в виде машинно-зависимого кода, их называют *тяжеловесными* (heavyweight). За небольшим исключением все компоненты Swing являются легковесными!

Легковесный компонент (lightweight)) – означает, что они написаны полностью на Java и не зависят от конкретной платформы, поскольку не опираются на платформенно-зависимые равноправные компоненты.

Библиотека *Swing* предоставляет коллекцию классов и интерфейсов, поддерживающих богатый набор визуальных компонентов, таких как кнопки, поля для ввода текста, полосы прокрутки, флажки, деревья узлов и таблицы.

Программы очень часто выводят результат своей работы в виде графических образов – графиков, рисунков или чего–то другого. Естественно, во всех языках высокого уровня есть встроенные средства для построения графических образов. В Java в библиотеке AWT используются классы на основе абстрактного класса *Graphics*.

Классы Graphics и Graphics2D нужны для работы с графическим контекстом. Поскольку графический контекст сильно зависит от конкретной графической платформы, эти классы сделаны абстрактными. Поэтому нельзя непосредственно создать экземпляры класса Graphics или Graphics2D. Однако каждая виртуальная машина Java реализует методы этих классов, создает их экземпляры для компонента и предоставляет объект класса Graphics методом getGraphics() класса Component или передает его как аргумент методов paint() и update().

Класс Graphics имеет очень большой набор методов для работы с графикой, рассмотрим наиболее часто используемые методы:

- drawLine(int x1, int y1, int x2, int y2); рисует линию из точки с координатами (x1,y1) до точки с координатами (x2,y2).
- drawOval(int x1, int y1, int width, int height); рисует овал (эллипс) с координатами левого верхнего угла (x1,y1), шириной width и высотой height.
- drawRect(int x1, int y1, int width, int height); рисует прямоугольник с координатами левого верхнего угла (x1,y1), шириной width и высотой height.
- drawArc(int x1, int y1, int width, int height, int startAngle, int arcAngle); строит окружность с координатами верхнего левого угла (x1,y1) затем отсекает и рисует арку сектора с углом arcAngle. Начальный угол, от которого начинается отсечение startAngle.
- drawPolygon(int[] xPoints, int[] yPoints, int points); рисует многоугольник по точкам, которые содержат массивы xPoints иуPoints, с количеством переломов points.
- fillOval(int x1, int y1, int width, int height); fillRect(int x1, int y1, int width, int height); fillPolygon(int[] xPoints, int[] yPoints, int points);
- fillArc(int x1, int y1, int width, int height, int startAngle, int arcAngle);

Для установки текущего цвета чернил нужно в качестве аргумента метода setColor() указать объект типа Color. Можно предварительно создать такой объект конструкцией Color clr = **new** Color();

Метод setFont(Font newFont) класса Graphics устанавливает текущий шрифт для вывода текста. Метод getFont() возвращает текущий шрифт. Как и все в языке Java, шрифт – это объект, а именно объект класса Font.

Объекты класса Font хранят начертания (glyphs) символов, образующие шрифт. Их можно создать двумя конструкторами:

- Font(Map attributes) – задает шрифт с указанным аргументом attributes атрибутами. Ключи атрибутов и некоторые их значения задаются константами класса TextAttribute из пакета java.awt.font. Этот конструктор характерен для Java 2D и будет рассмотрен далее в настоящей главе;
- Font(String name, int style, int size) – задает шрифт по имени name, со стилем style и размером size типографских пунктов.

```
import javax.swing.*;
import java.awt.*;
public class GUI extends JFrame {
GUI(){
super();
setLayout(null);
setSize(300,500);
setVisible(true);
this.setDefaultCloseOperation(EXIT_ON_CLOSE);    }
public void paint(Graphics my_picture){
my_picture.setColor(Color.WHITE);
my_picture.fillRect(0, 0, 300, 800);
my_picture.setColor(Color.BLACK);
my_picture.drawOval(90, 50, 100, 100);
my_picture.drawLine(140, 150, 140, 300);
my_picture.drawLine(140, 300, 100, 400);
my_picture.drawLine(140, 300, 180, 400);
my_picture.drawLine(140, 200, 75, 250);
my_picture.drawLine(140, 200, 205, 250);    }
public static void main(String[] args) {
```

```
new GUI();
}    }
```

Порядок выполнения работы

1. Изучить лекционный материал.

2. Выполнить лабораторную работу согласно варианту.

3. Представить отчет по лабораторной работе для защиты.

Варианты заданий

1. Создать фрейм со строкой, движущейся горизонтально, отражаясь от границ и меняя при этом свой цвет, на цвет, выбранный из выпадающего списка.

2. Промоделировать вращение спутника вокруг планеты по эллиптической орбите. Когда скрывается за планетой – спутник не виден.

3. Промоделировать аналоговые часы (со стрелками) с кнопками для увеличения/уменьшения времени на час/минуту.

4. Создать класс Треугольник и класс Точка. Объявить массив из n объектов класса Точка, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.

5. Создать класс Треугольник. Отобразить вращение треугольника вокруг своего центра тяжести.

6. Создать программу вывода текста и рисования в форме графиков функций y(x)=sin(x)-sin(2x), x_0=-3pi/2, x_n=3pi/2.

7. Разработать модель светофора, которая является визуальным представлением светофора.

8. Разработать модель гусеницы, которая является визуальным представлением жизненного цикла гусеницы.

9. Создать фрейм с областью для рисования «пером». Создать меню для выбора цвета и толщины линии.

10. Изобразить во фрейме приближающийся издали шар, удаляющийся шар. Шар должен двигаться с постоянной скоростью.

Контрольные вопросы

1. Что такое графическая библиотека классов?
2. Что называется графическим компонентом?
3. Назовите известные вам графические компоненты.
4. Что такое контейнер в графическом интерфейсе?
5. Будет ли основное окно приложения контейнером?

6. Можно ли использовать библиотеку Swing без библиотеки AWT?

7. Какая разница между компонентами AWT и компонентами Swing?

8. Можно ли совсем отказаться от компонентов библиотеки AWT?

9. Назовите несколько методов класса Graphics Вычерчивающих примитивные фигуры, залитые текущим цветом?

3 Методические указания для организации самостоятельной работы

3.1 Общие положения

Целью самостоятельной работы является систематизация, расширение и закрепление теоретических знаний, использование материала, собранного и полученного в ходе самостоятельной подготовки к лабораторным работам.

Самостоятельная работа включает в себя подготовку к лабораторным работам, проработку лекционного материала и подготовку к контрольным работам, проработку тем дисциплины, вынесенных на самостоятельное изучение.

3.2 Проработка лекционного материала и подготовка к контрольным работам

Изучение теоретической части дисциплин призвано не только углубить и закрепить знания, полученные на аудиторных занятиях, но и способствовать развитию у студентов творческих навыков, инициативы и организовать свое время.

Проработка лекционного материала включает:

- чтение студентами рекомендованной литературы и усвоение теоретического материала дисциплины;
- знакомство с Интернет-источниками;
- подготовку к различным формам контроля (контрольные работы);
- подготовку ответов на вопросы по различным темам дисциплины в той последовательности, в какой они представлены.

Планирование времени, необходимого на изучение дисциплин, студентам лучше всего осуществлять весь семестр, предусматривая при этом регулярное повторение материала.

Материал, законспектированный на лекциях, необходимо регулярно прорабатывать и дополнять сведениями из других источников литературы, представленных не только в программе дисциплины, но и в периодических изданиях.

При изучении дисциплины сначала необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений, терминов, сведений, требующих запоминания и яв-

ляющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знания по дисциплине рекомендуется использовать Интернет-ресурсы; проводить поиски в различных системах и использовать материалы сайтов, рекомендованных преподавателем.

Задачи, стоящие перед студентом при подготовке и написании контрольной работы:

- закрепление полученных ранее теоретических знаний;
- выработка навыков самостоятельной работы;
- выяснение подготовленности студентов к зачету.

Контрольные выполняются студентами в аудитории, под наблюдением преподавателя.

Темы контрольных работ:

1. Принципы ООП.

2. Библиотеки классов.

Вопросы, выносимые на контрольную работу «Принципы ООП».

1. Назовите принципы ООП и расскажите о каждом.
2. Дайте определение понятию «класс».
3. Что такое поле/атрибут класса?
4. Дайте определение понятию «конструктор».
5. Чем отличаются конструкторы по умолчанию, копирования и конструктор с параметрами?
6. Какие модификации уровня доступа вы знаете, расскажите про каждый из них.
7. О чем говорят ключевые слова «this», «super», где и как их можно использовать?
8. Дайте определение понятию «метод».
9. Что такое сигнатура метода?
10. Какие методы называются перегруженными?
11. Расскажите про переопределение методов.
12. Чем отличается переопределение от перегрузки?
13. Где и для чего используется модификатор abstract?
14. Можно ли объявить метод абстрактным и статическим одновременно?
15. Что означает ключевое поле static?
16. К каким конструкциям Java применим модификатор static?

17. О чем говорит ключевое слово final?

18. Какое ключевое слово используется, чтобы указать, что класс реализует интерфейс?

19. Почему нельзя объявить метод интерфейса с модификатором final или static?

20. В чем разница вложенных и внутренних классов?

Вопросы, выносимые на контрольную работу «Библиотеки классов».

1. Как связан любой пользовательский класс с классом Object?

2. Какой пакет импортируется по умолчанию?

3. Расскажите про каждый из методов класса Object.

4. Что за метод equals()? Чем он отличается от операции ==.

5. Что общего и чем отличаются следующие потоки: InputStream, OutputStream, Reader, Writer?

6. Ввод/вывод в Java: фильтрованные потоки.

7. Ввод/вывод в Java: буферизированные потоки.

8. Что вы знаете об обрабатываемых и не обрабатываемых (catched/uncatched) исключениях?

9. В чем особенность RuntimeException?

10. Как написать собственное (пользовательское) исключение? Какими мотивами вы будете руководствоваться при выборе типа исключения: checked/unchecked?

11. Дайте определение понятию «коллекция».

12. Назовите преимущества использования коллекций.

13. Какие данные могут хранить коллекции?

14. Какова иерархия коллекций?

15. Что вы знаете о коллекциях типа List?

16. Что вы знаете о коллекциях типа Set?

17. Что вы знаете о коллекциях типа Queue?

18. Что вы знаете о коллекциях типа Map, в чем их принципиальное отличие?

19. Назовите основные реализации List, Set, Map.

20. Какие коллекции синхронизированы?

3.3 Подготовка к лабораторным работам

Проведение лабораторных работ включает в себя следующие этапы:

- постановку темы занятий и определение задач лабораторной работы;
- определение порядка лабораторной работы или отдельных ее этапов;
- непосредственное выполнение лабораторной работы студентами и контроль за ходом занятий;
- подведение итогов лабораторной работы и формулирование основных выводов;
- оформление отчета и защиты лабораторной работы (демонстрация работы и ответы на вопросы по теме лабораторной работы).

При подготовке к лабораторным занятиям необходимо заранее изучить методические рекомендации по его проведению. Обратить внимание на цель занятия, на основные вопросы для подготовки к занятию, на содержание темы занятия.

Если в процессе лабораторной работы или над изучением теоретического материала у студента возникают вопросы, разрешить которые самостоятельно не удается, необходимо обратиться к преподавателю для получения у него разъяснений или указаний.

3.4 Самостоятельное изучение тем теоретической части курса

Темы, отводимые на самостоятельное изучение:

1. Паттерны.
2. Лямбда-выражения.
3. Перечисления.
4. Обобщения.

Рекомендуемая литература:

1. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. Паттерны проектирования. – СПб. : Питер, 2011. – 656.

2. М. Йенер, А. Фидом. Java EE. Паттерны проектирования для профессионалов. – СПб. : Питер, 2016. – 240 с.

3. Г. Шилдт. Java 8: руководство для начинающих, 6-е изд. : Пер. с англ. – М. : Вильямс, 2015. – 720 с.

4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.

3.4.1 Паттерны

Паттерны относятся к более высокому уровню, чем библиотеки. Они определяют способы структурирования классов и объектов для решения некоторых задач, главная задача – адаптировать их для своих конкретных приложений.

Паттерны проектирования – это «описания обменивающихся информацией объектов и классов, которые адаптированы для решения общей проблемы проектирования в конкретных условиях».

Паттерны – не что иное, как применение принципов ООП. Паттерны проектирования предлагают решения распространенных проблем проектирования приложений. В объектно-ориентированном программировании паттерны проектирования обычно нацелены на решение задач, связанных с созданием и взаимодействием объектов, а не крупномасштабных задач, стоящих перед общей архитектурой программного обеспечения. Они обеспечивают обобщенные решения в виде шаблонов, которые могут быть применены к практическим задачам.

Паттерны проектирования были изначально разделены на три группы.

1. Порождающие паттерны управляют созданием и инициализацией объекта, а также выбором класса. Примерами паттернов из этой группы могут быть «Одиночка» и «Фабрика».

2. Паттерны поведения управляют связью, обменом сообщениями и взаимодействием между объектами. Примером паттерна из этой группы может быть «Наблюдатель».

3. Структурные паттерны упорядочивают отношения между классами и объектами, обеспечивая критерии соединения и совместного использования связанных объектов для достижения желаемого поведения. Хороший пример паттерна из этой группы – «Декоратор».

Перечень вопросов, подлежащих изучению

1. Что такое шаблоны проектирование?
2. Для каких задач нужны шаблоны проектирования.
3. Виды и применение структурных шаблонов.
4. Виды и применение порождающих шаблонов.
5. Виды и применение поведенческих шаблонов.

3.4.2 Лямбда-выражения

Выпуск JDK 8 дополнил Java новым средством – лямбда-выражениями, значительно усилившим выразительные возможности языка. Лямбда-выражения не только вводят в язык новый синтаксис, но и упрощают реализацию некоторых часто используемых конструкций. Подобно тому как введение обобщенных типов несколько лет тому назад оказало значительное влияние на дальнейшее развитие Java, лямбда-выражения формируют сегодняшний облик Java. Их роль в развитии языка Java действительно весьма существенна.

Лямбда-выражение – это, по сути, анонимный (т.е. неименованный) метод. Однако сам этот метод никогда не выполняется. Он лишь позволяет назначить реализацию кода метода, определяемого функциональным интерфейсом. Таким образом, лямбда-выражение представляет собой некую форму анонимного класса. Другой часто употребляемый эквивалентный термин в отношении лямбда-выражений - замыкание.

Лямбда-выражения вводят в язык Java новый синтаксис. В них используется новый лямбда-оператор -> (другое название - оператор стрелка). Этот оператор разделяет лямбда-выражение на две части. В левой части указываются параметры, если этого требует лямбда-выражение, а в правой – тело лямбда-выражения, которое описывает действия, выполняемые лямбда-выражением. В Java поддерживаются две разновидности тел лямбда-выражений. Тело одиночного лямбда-выражения состоит из одного выражения, тело блочного - из блока кода.

Перечень вопросов, подлежащих изучению

1. Что такое лямбда-оператор?
2. Что такое функциональный интерфейс?
3. Какая связь существует между функциональными интерфейсами и лямбда-выражениями?
4. Назовите два общих типа лямбда-выражений.
5. Составьте лямбда-выражение, которое возвращает значение true, если число принадлежит к диапазону чисел 10-20, включая граничные значения.

3.4.3 Перечисления

В своей простейшей форме *перечисление* – это список именованных констант, определяющих новый тип данных. В объектах перечислимого типа могут храниться лишь значения, содержащиеся в этом списке. Таким образом, перечисления позволяют определять новый тип данных, характеризующийся строго определенным рядом допустимых значений.

Перечисления создаются с использованием ключевого слова enum.

Несмотря на то, что предыдущие примеры позволили продемонстрировать создание и использование перечислений, они не дают полного представления обо всех возможностях этого типа данных. В Java, в отличие от других языков программирования, перечисления реализованы как типы классов. И хотя для создания экземпляров класса enurn не требуется использовать оператор new, во всех остальных отношениях они ничем не отличаются от классов. Реализация перечислений Java в виде классов позволила значительно расширить их возможности. В частности, допускается определение конструкторов перечислений, добавление в них объектных переменных и методов и даже создание перечислений, реализующих интерфейсы.

Перечень вопросов, подлежащих изучению

1. Константы перечислимого типа иногда называют самотипизированными. Что это означает?
2. Какой класс автоматически наследуют перечисления?

3.4.4 Обобщения

Термин «обобщение», по сути, означает параметризированный тип. Специфика параметризированных типов состоит в том, что они позволяют создавать классы, интерфейсы и методы, в которых тип данных указывается в виде параметра. Используя обобщения, можно создать единственный класс, который будет автоматически работать с различными типами данных. Классы, интерфейсы и методы, оперирующие параметризированными типами, называются обобщенными, как, например, обобщенный класс или обобщенный метод.

Главное преимущество обобщенного кода состоит в том, что он будет автоматически работать с типом данных, переданным ему в качестве параметра. Многие алгоритмы выполняются одинаково, независимо от того, к данным какого типа они будут применяться. Например, быстрая сортировка не зависит от типа данных, будь то Integer, String, Object или Thread. Используя обобщения, можно реализовать алгоритм один раз, а затем применять его без дополнительных усилий к любому типу данных.

Ниже приведена общая форма объявления обобщенного класса.

class имя_класса<список_параметров_типа> { // ...

А вот как выглядит синтаксис объявления ссылки на обобщенный класс.

имя_класса<список_аргументов_типа> имя_переменной new имя_класса<список_аргументов_типа> (список_ аргументов_ конструктора);

Перечень вопросов, подлежащих изучению

1. Обобщения очень важны, поскольку они позволяют создавать код, который: а) обеспечивает типовую безопасность; б) пригоден для повторного использования; в) отличается высокой надежностью; г) обладает всеми перечисленными выше свойствами.
2. Можно ли указывать простой тип в качестве аргумента типа?
3. Что обозначает знак ? в обобщениях?
4. Существуют ли параметры типа на стадии выполнения программы?
5. Что означает пара угловых скобок (< >)?

4 Рекомендуемые источники

1. Ашарина И.В. Объектно-ориентированное программирование в C++: лекции и упражнения [Электронный ресурс] : учеб. пособие. – М. : Горячая линия-Телеком, 2012. — 320 с. — Режим доступа: https://e.lanbook.com/book/5115 (дата обращения: 4.05.2018).

2. Хорев П.Б. Объектно-ориентированное программирование с примерами на C# [Электронный ресурс] : учебн. пособие / Хорев П.Б. – М.: Форум, НИЦ ИНФРА-М, 2016. – 200 с. — Режим доступа: http://znanium.com/bookread2.php?book=529350 (дата обращения: 4.05.2018).

3. Васюткина И.А. Технология разработки объектно-ориентированных программ на JAVA [Электронный ресурс]. – Новосиб. : НГТУ, 2012. – 152 с. – Режим доступа: http://znanium.com/bookread2.php?book=557111 (дата обращения: 4.05.2018).

4. Зыков С. В. Программирование. Объектно-ориентированный подход [Электронный ресурс] : учебник и практикум для академического бакалавриата / С. В. Зыков. – М. : Издательство Юрайт, 2018. – 155 с. – Режим доступа : www.biblio-online.ru/book/E006A65E-B936-4856-B49E-1BA48CF1A52F (дата обращения: 4.05.2018).

5. Огнева, М. В. Программирование на языке C++ [Электронный ресурс] : практический курс : учебное пособие для СПО / М. В. Огнева, Е. В. Кудрина. – М. : Издательство Юрайт, 2018. – 335 с. — Режим доступа : www.biblio-online.ru/book/B76AB4A4-7623-4842-9136-B6ADC57B90BC (дата обращения: 4.05.2018).

6. Павловская Т. А. C/C++. Процедурное и объектно-ориентированное программирование [Электронный ресурс] : учебник для вузов / Т. А. Павловская. – СПб. : Питер, 2015. – 496 с. – Режим доступа : https://ibooks.ru/reading.php?productid=341427 (дата обращения: 4.05.2018).

7. Морозова Ю.В. Информатика и программирование для студентов направлений «Программной инженерии» и «Бизнес-информатика» [Электронный ресурс]. – Режим доступа: https://sdo.tusur.ru/course/view.php?id=39 (дата обращения: 4.05.2018).

Приложение А

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

НАЗВАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Отчет по лабораторной работе по дисциплине
«Информатика и программирование»

Студент гр. ___
_____ И. О. Фамилия
«___»_____201_ г.

Руководитель
доцент каф. АОИ,
канд. техн. наук
_____ Ю. В. Морозова
«___»_____201_ г.

Томск 201_