

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизации обработки информации (АОИ)

Ю. В. Морозова

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ И ПРОГРАММИРОВАНИЕ

**Методические указания
по выполнению лабораторных работ
и организации самостоятельной работы для студентов
заочной формы обучения с применением ДОТ**

Томск 2018

Корректор: А. Н. Миронова

Морозова Ю. В.

Объектно-ориентированный анализ и программирование : методические указания по выполнению лабораторных работ и организации самостоятельной работы для студентов заочной формы обучения с применением ДОТ / Ю. В. Морозова. – Томск : ФДО, ТУСУР, 2018. – 54 с.

© Морозова Ю. В., 2018

© Оформление.

ФДО, ТУСУР, 2018

ОГЛАВЛЕНИЕ

1 Введение.....	4
2 Методические указания к проведению лабораторных работ	5
2.1 Знакомство с объектно-ориентированным языком Java и IDE Eclipse	5
2.2 Лабораторная работа № 1 «Классы».....	19
2.3 Лабораторная работа № 2 «Абстрактные классы и интерфейсы».....	31
2.4 Лабораторная работа № 3 «Потоки».....	40
3 Требования к содержанию и оформлению отчета.....	48
4 Методические указания для организации самостоятельной работы	49
4.1 Общие положения.....	49
4.2 Проработка лекционного материала и подготовка к контрольным работам	49
4.3 Подготовка к лабораторным работам.....	51
5 Рекомендуемая литература.....	52
Приложение А Пример оформления титульного листа	54

1 ВВЕДЕНИЕ

Целью дисциплины «Объектно-ориентированный анализ и программирование» является формирование у студентов объектно-ориентированного мышления и объектно-ориентированного 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.

Общие требования к выполнению лабораторных работ

Выбор варианта лабораторной работы осуществляется по общим правилам с использованием следующей формулы:

$$V = (N \times K) \operatorname{div} 100,$$

где V – искомый номер варианта,

N – общее количество вариантов (15),

div – целочисленное деление,

при $V = 0$ выбирается максимальный вариант,

K – код варианта.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Знакомство с объектно-ориентированным языком Java и IDE Eclipse

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 про-

странства. 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 – Запуск Eclipse

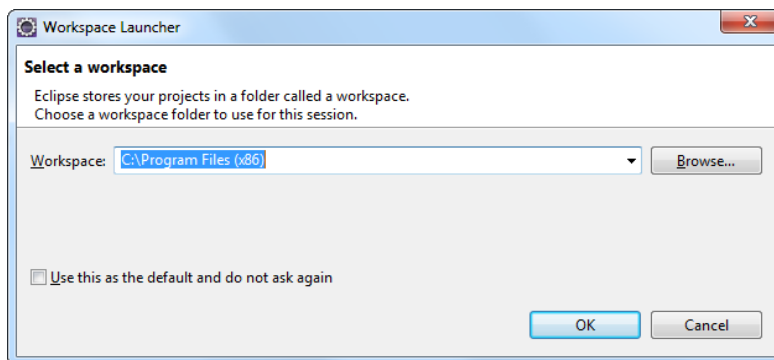


Рис. 2.2 – *Workspace*

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).

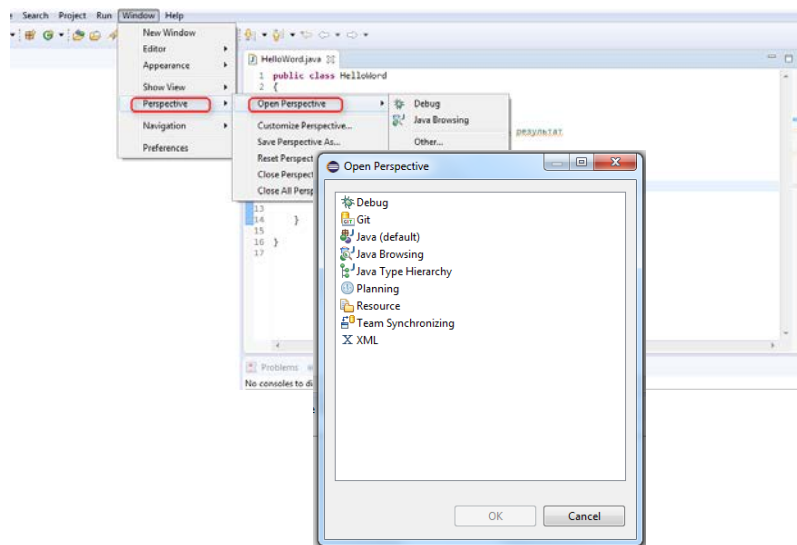


Рис. 2.3 – *Perspective*

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).

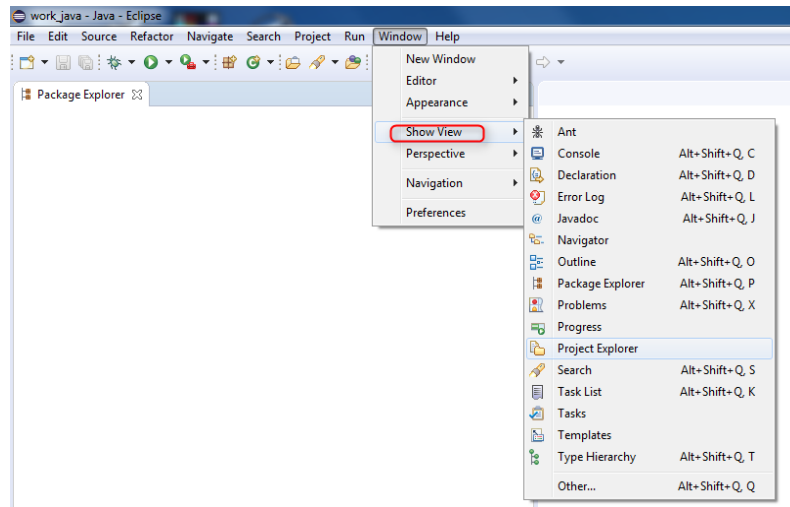


Рис. 2.4 – Views

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.

Проект *Java development tools (JDT)* с помощью JDT-плагины 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 *ОК* 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, содержат 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», рабочая область программиста.

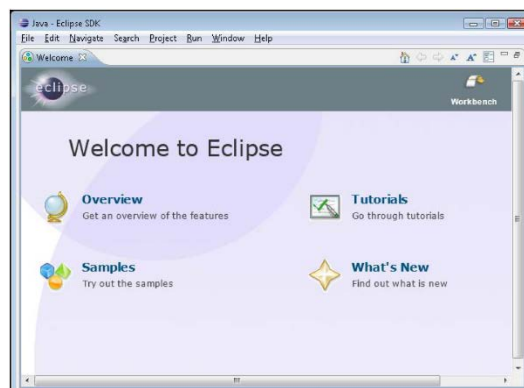


Рис. 2.5 – Окно приветствия Eclipse

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 – Характеристика представлений

Представления	Описание
<i>Package Explorer</i>	Отображает Java-проект с его структурой, определяемой сборкой проекта, в виде узлов папок и библиотек, пакетов, файлов с их внутренней структурой
<i>Outline</i>	Отображает компилируемую структуру редактируемого в данный момент Java-файла

Представления	Описание
Окончание табл. 2.1 <i>Problems</i>	Отображает ошибки и предупреждения сборщика проекта
<i>Javadoc</i>	Отображает документацию выбранного в данный момент Java-элемента
<i>Declaration</i>	Отображает исходный код выбранного в данный момент Java-элемента
<i>Console</i>	Отображает системный вывод выполнения Java-кода
<i>Debug</i>	Обеспечивает управление процессом отладки и запуска Java-кода
<i>Tasks</i>	Отображает список маркеров задач проекта
<i>Breakpoints</i>	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).

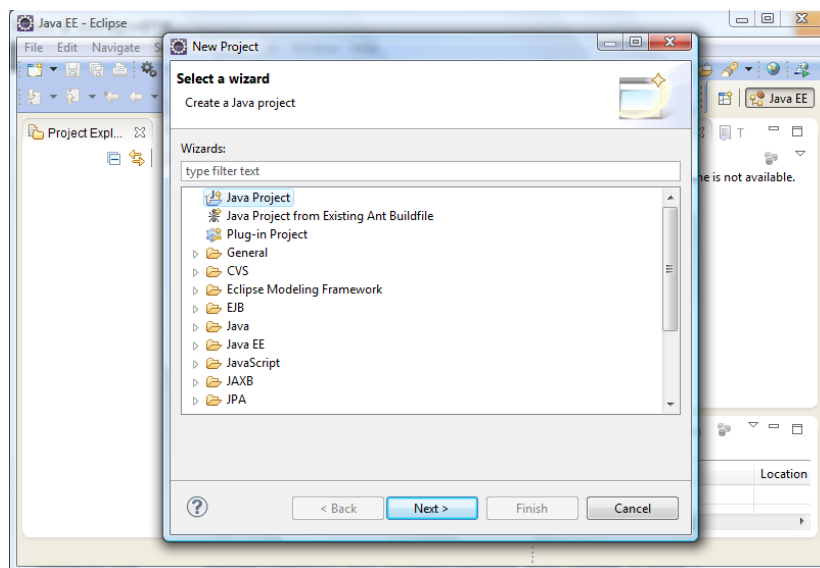


Рис. 2.6 – Создание нового проекта. Шаг 1

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

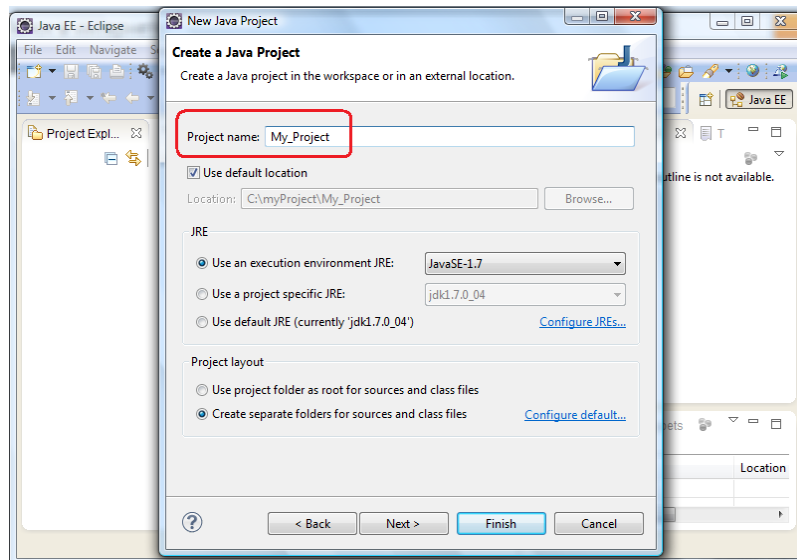
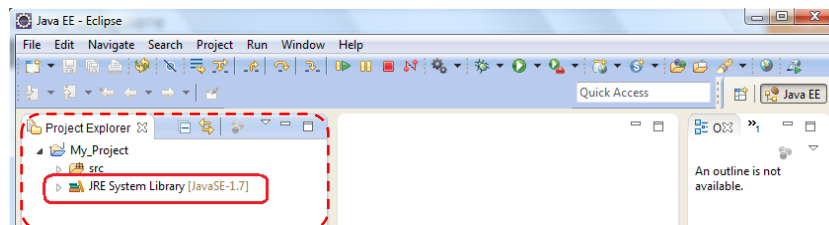


Рис. 2.7 – Создание нового проекта. Шаг 2

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).

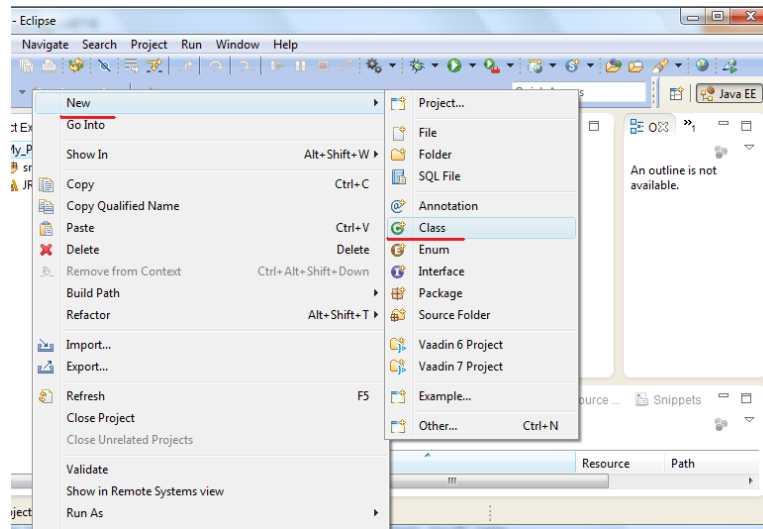


Рис. 2.9 – Создание нового класса. Шаг 1

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 HelloWorld. Откроем созданный класс и завершим нашу программу. Добавим в метод `main()` следующий код (рис. 2.11) `System.out.println("Hello world");`

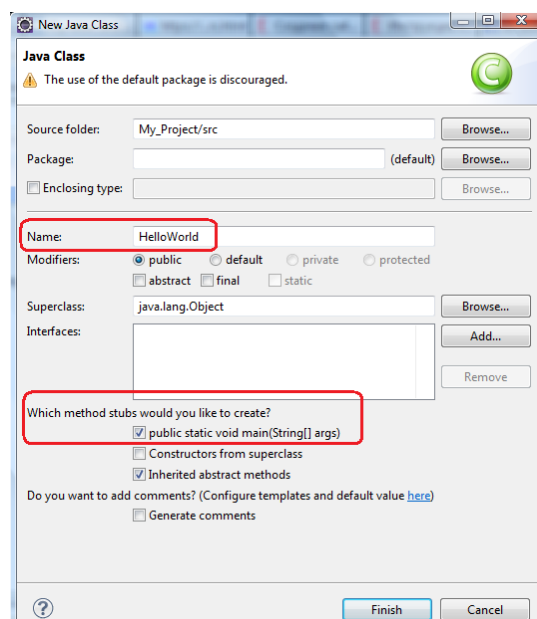


Рис. 2.10 – Создание нового класса. Шаг 2

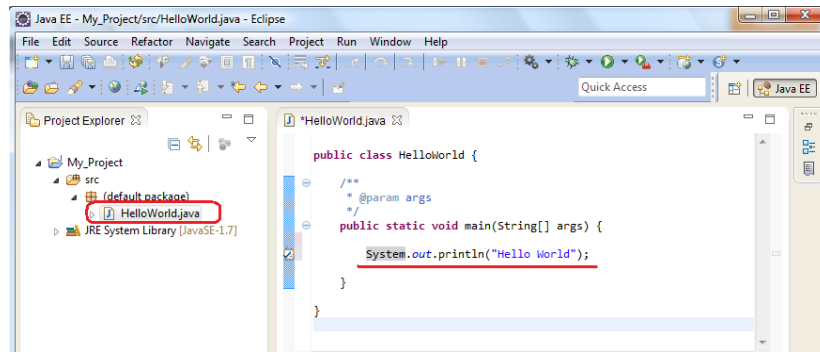


Рис. 2.11 – Создание нового класса. Шаг 3

Сохраним изменения с помощью 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 подэлемент 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*.

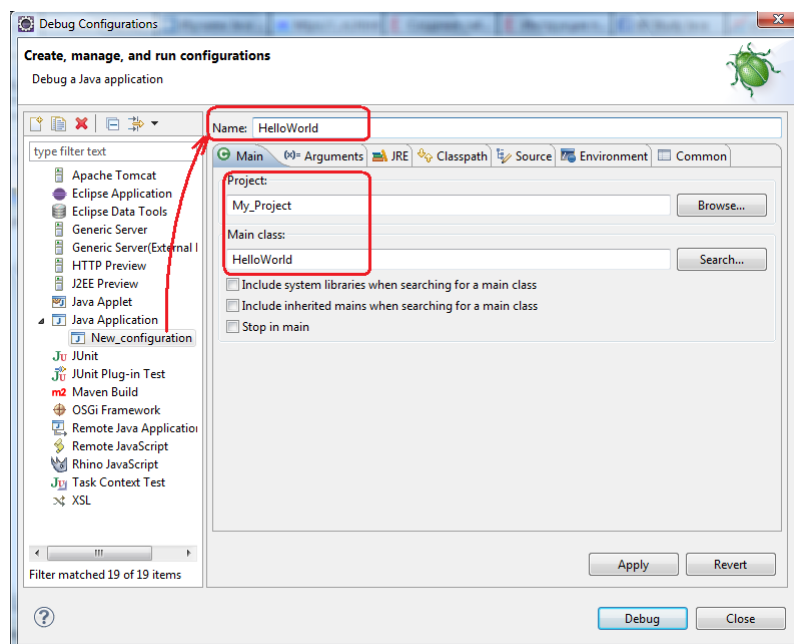


Рис. 2.12 – Запуск проекта

В результате, в консоли будет напечатано: *HelloWorld* (рис. 2.13).

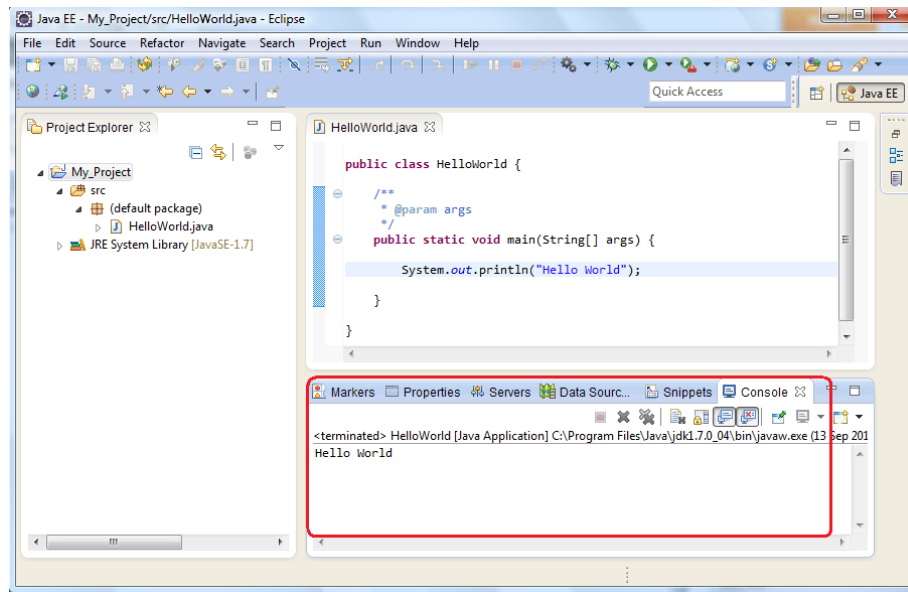


Рис. 2.13 – Console

Для запуска программы в дальнейшем достаточно нажимать специальный значок на панели инструментов, выбрав *HelloWorld* (рис. 2.14).

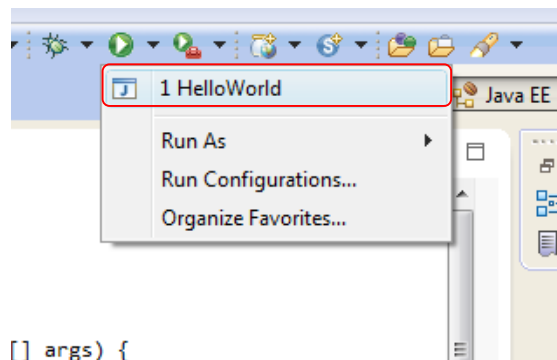


Рис. 2.14 – Запуск проекта через панель инструментов

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.

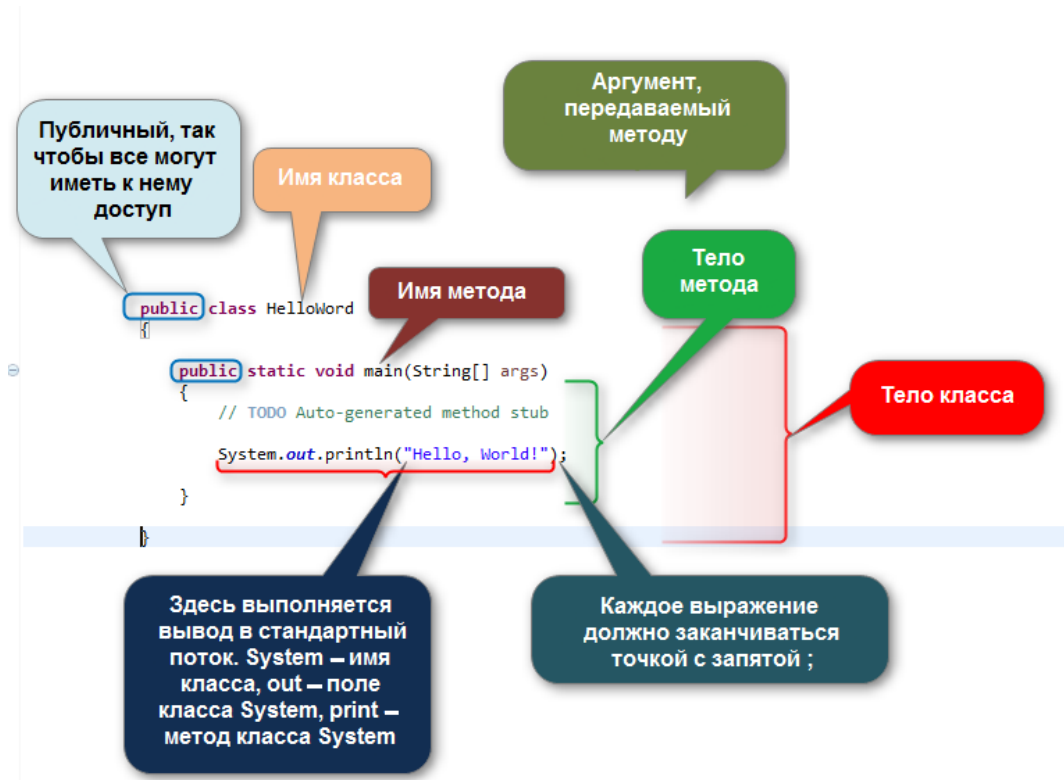


Рис. 2.15 – Программа, написанная на языке Java

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 в 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 – строковые 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`. Однако между Java-программистами заключено соглашение, называемое *Code Conventions for the Java Programming Language* (<http://www.oracle.com/technetwork/java/codeconv-138413.html>), согласно которому имя файла должно в точности совпадать с именем класса, содержащего метод `main()`. Данное правило 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!
```

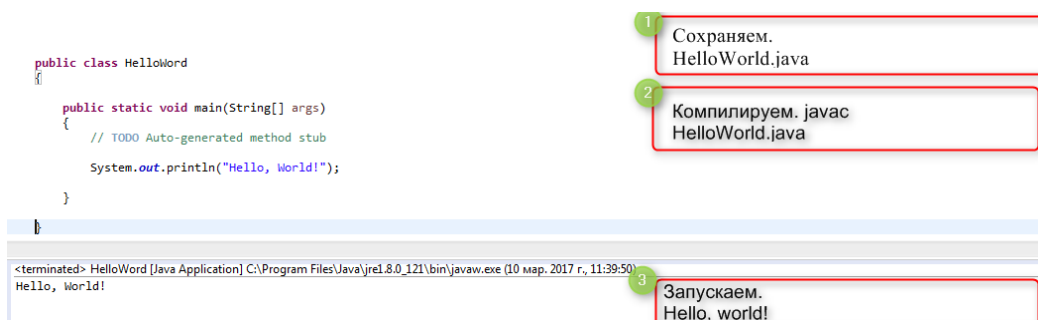


Рис. 2.16 – Запуск программы, написанной на языке Java

2.2 Лабораторная работа № 1 «Классы»

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

Краткая теория

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 определяют 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, определяющими 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).

```

Class_modifiers class Class_name {
    //attributes
    //methods
    //constructors
}

```



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 вообще, что означает уровень доступа по умолчанию.

```
public class Cat {
    private int weight; // вес кота
    private String name; // имя кота
    private String color; //окрас кота
}
```

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».

Научим нашего кота *есть* и *разговаривать*. Опишем это поведение с помощью методов.

```
//кот ест
public void eat(){
    System.out.print("Ам-ам...");
}
```

И немного иначе опишем метод – *кот говорит*:

```
//кот говорит
public String voice(String words){
    String phrase = "Котик говорит"+ words;
    return phrase;
}
```

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 класса 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». Она служит для включения дополнительной информации, которую можно прочесть 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 - наслед-
ник 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
родительского класса*/
        }
        public static void main(String[] args) {
            Kitten cat1= new Kitten();
            cat1.voice("Мася");//дочерний метод
            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 – 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) //конструктор класса Cat
    {
        name = n;
        age = a;
    }
    //кот говорит
    public String voice(String words){
        String phrase = name+"говорит"+words;
        return phrase;
    }
}
```

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

```
Cat = new Cat("Василий", 2);
```

В результате переменная `cat` будет указывать на конкретного кота по кличке Василий, которому 2 года.

Переменную `cat` теперь можно использовать для *вызова методов класса Cat*, например:

```
System.out.print(cat.voice("мяу"));
```

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.

Варианты задания для лабораторной работы № 1

Вариант 1

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

Вариант 2

Определить класс `Matrix` размерности $n \times n$. Класс должен содержать несколько конструкторов. Реализовать методы вывода матрицы на экран, методы определения и получения размерности матрицы и самой матрицы.

Вариант 3

Определить класс «Прямоугольный треугольник, заданный длинами сторон». Предусмотреть возможность операции присваивания, определения

площади и периметра, а также логический метод, определяющий, существует ли такой треугольник. Конструктор должен позволять создавать объекты с начальной инициализацией и без нее.

Вариант 4

Создать класс Triangle (треугольник), задав в нем длину двух сторон, угол между ними, методы вычисления площади и периметра.

Вариант 5

Определить класс Color. 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 $[0 \dots 255]$. Описать свойства для получения состояния объекта и метод изменения цвета. Определить конструкторы и функцию печати.

Вариант 6

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

Вариант 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.

Вариант 8

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

Вариант 9

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.

Вариант 10

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

Вариант 11

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

Вариант 12

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

Вариант 13

Определить класс «Равнобедренный треугольник, заданный длинами сторон». Предусмотреть возможность операции присваивания, определения

площади и периметра, а также логический метод, определяющий, существует ли такой треугольник. Конструктор должен позволять создавать объекты с начальной инициализацией и без нее.

Вариант 14

Определить класс *Matrix* размерности $n \times m$. Класс должен содержать несколько конструкторов. Реализовать методы вывода матрицы на экран, методы определения и получения размерности матрицы и самой матрицы.

Вариант 15

Определить класс *Date*. Реализовать методы: задать дату, вывести на консоль день недели по заданной дате. День недели представить в виде перечисления. Рассчитать количество дней в заданном временном промежутке.

Контрольные вопросы для самоподготовки

1. Дайте определение понятию «класс».
2. Что такое поле/атрибут класса?
3. Как правильно организовать доступ к полям класса?
4. Дайте определение понятию «конструктор».
5. Чем отличаются конструктор по умолчанию, конструктор 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.

2.3 Лабораторная работа № 2 «Абстрактные классы и интерфейсы»

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

Краткая теория

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 неизвестна 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 супер-классами 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 = new Cat();
        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 некоторые функции, не имеющие конкретной реализации, которые 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 интер-
фейсы:
```

```

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
}
}

```

Варианты заданий для лабораторной работы № 2

Вариант 1

Создать абстрактный базовый класс Triangle для представления треугольника с абстрактными методами 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.

Вариант 2

Создайте интерфейс FastFood(), его наследников Sandwich и Hamburger. 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.

Вариант 3

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

Вариант 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.

Вариант 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.

Вариант 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.

Вариант 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 позволяет

снять клиенту в один день. Реализовать метод инициализации банкомата, метод загрузки купюр в банкомат и метод снятия определенной суммы денег.

Вариант 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 изделия» и «Колбасные изделия», а также организовать поиск просроченного товара (на момент текущей даты).

Вариант 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 методы equals() и toString().

Вариант 10

Создать интерфейс Plant (растение), содержащий методы: расчет количества воды для полива (принимаемые параметры: время 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.

Вариант 11

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Машина <- abstract class Грузовая Машина <- class Камаз. Реализовать методы определения грузоподъемности, вместимости топливного бака, мощности двигателя,

а также метод, позволяющий выводить информацию о машине, где грузоподъемность выше введенной с клавиатуры.

Вариант 12

Создать интерфейс Production (продукция), содержащий следующие методы: расчет процента бракованной продукции (принимаемые параметры: общий объем произведенной продукции и количество бракованных изделий), вывод информации о продукции. Создать классы «Техника» и «Канцелярия», реализующих этот интерфейс. Создать группу объектов этих классов. Создать третий класс Info, позволяющий выводить информацию о продукции, где процент брака выше введенного с клавиатуры.

Вариант 13

Создать абстрактный класс Number с методами, реализующими арифметические операции. На его основе реализовать классы Integer и Real. Создать класс List (список), содержащий массив объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 14

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. Создать класс Purse (кошелек), содержащий массив объектов этих классов. Предусмотреть возможность вывода общей суммы, переведенной в рубли, и суммы по каждой из валют.

Вариант 15

Создать абстрактный класс `Transport`. На его основе реализовать классы `Car` (автомобиль), `Bicycle` (велосипед) и `Lorry` (грузовик). Классы должны иметь возможность задавать и получать параметры средств передвижения (цена, максимальная скорость, год выпуска и т. д.). Наряду с общими полями и методами каждый класс должен содержать и специфичные для него поля. Создать класс `Garage`, содержащий массив объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка.

Контрольные вопросы для самоподготовки

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`?

2.4 Лабораторная работа № 3 «Потоки»

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

Краткая теория

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

Поток данных (stream) представляет собой 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 символьных – 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 использовать 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 целого
числа: ");
        if(in.hasNextInt()) { /*возвращает истину
если с потока ввода можно считать целое число*/
            i = in.nextInt(); /* считывает целое число
с потока ввода и сохраняем в переменную*/
            System.out.println(i);
        } else {
            System.out.println("Вы ввели не целое
число");
        }
    }
}
```

Варианты заданий для лабораторной работы № 3

Вариант 1

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

Вариант 2

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

Вариант 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.

Вариант 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.

Вариант 5

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

Вариант 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.

Вариант 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.

Вариант 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`.

Вариант 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».

Вариант 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».

Вариант 11

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».

Вариант 12

Дан файл, содержащий различные даты. Каждая дата – это число, месяц и год. Проверить допустимость значений и найти самую раннюю дату s . В случае недопустимых значений полей выбрасываются исключения. Обработать исключения: ошибки чтения и записи файла и «файл не существует».

Вариант 13

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. Обработать исключения: ошибки чтения и записи файла и «файл не существует».

Вариант 14

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 сначала шли положительные, потом отрицательные числа.

Вариант 15

Требуется ввести последовательность строк из текстового потока и выполнить указанные действия: подсчитать количество букв и знаков препинания. Эти данные записать в новый текстовый файл.

Контрольные вопросы для самоподготовки

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`?

3 ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ОТЧЕТА

Лабораторные занятия выполняются студентами самостоятельно.

После выполнения работы студент должен представить отчет о проделанной работе с описанием полученных результатов и выводов. В отчете должны содержаться:

- 1) тема 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 в виде скриншота консоли;
- 6) выводы.

Оформление отчета по работе должно соответствовать требованиям образовательного стандарта вуза ОС ТУСУР 01–2013. Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления. Режим доступа: <https://regulations.tusur.ru/documents/70>.

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 12 пт. На материалы, взятые из литературы и других источников должны быть даны ссылки с указанием номера источника по списку использованной литературы. В приложениях размещаются листинг с комментариями, схемы программы, скриншоты интерфейса. Приложения нумеруются русскими буквами в порядке появления ссылок на них в основном тексте документа.

4 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

4.1 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 самостоятельное изучение тем теоретической части курса, подготовку к лабораторным работам и подготовку к контрольным работам, оформление отчетов по лабораторным работам.

4.2 Проработка лекционного материала и подготовка к контрольным работам

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

Проработка лекционного материала включает:

- чтение студентами рекомендованной литературы и усвоение теоретического материала курса;
- знакомство с интернет-источниками;
- подготовку к контрольной работе;
- 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.

При изучении дисциплины необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений, терминов, сведений, требующих запоминания и являющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знаний 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 итоговой аттестации.

Вопросы для самоконтроля

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, конструктор 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?

4.3 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;
- подведение итогов лабораторной работы и формулирование основных выводов;
- оформление отчета лабораторной работы.

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

5 РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Ашарина И. В. Объектно-ориентированное программирование в С++ : лекции и упражнения : учеб. пособие [Электронный ресурс] / И. В. Ашарина. – 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 С# : учеб. пособие [Электронный ресурс] / П. Б. Хорев. – 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] / И. А. Васюткина. – Новосибирск : 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 : учебник и практикум для академического бакалавриата [Электронный ресурс] / С. В. Зыков. – М. : 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 С++ : практический курс : учеб. пособие для СПО [Электронный ресурс] / М. В. Огнева, Е. В. Кудрина. – М. : 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. С/С++. Процедурное и объектно-ориентированное программирование : учебник для вузов [Электронный ресурс] / Т. А. Павловская. – СПб. : Питер, 2015. – 496 с. – Режим доступа: <https://ibooks.ru/reading.php?productid=341427> (дата обращения: 4.05.2018).

7. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Гради Буч. – 3-е изд. – М. : ООО «ИД «Вильямс», 2008. – 720 с.

8. Гослинг Дж. Язык программирования Java SE 8. Подробное описание / Дж. Гослинг, Б. Джой, Г. Стил, Г. Брача, А. Бакли. – 5-е изд. – М. : ООО «ИД «Вильямс», 2015. – 672 с.

9. Мейер Б. Почувствуй класс / Б. Мейер ; пер. с англ. под ред. В. А. Биллига. – М. : НОУ «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011. – 775 с.

10. Шилдт Г. Java 8: руководство для начинающих : пер. с англ. / Г. Шилдт. – 6-е изд. – М. : ООО «ИД «Вильямс», 2015. – 720 с.

11. Блинов И. Н. Java 2: практ. руководство / И. Н. Блинов, В. С. Романчик. – Минск : УниверсалПресс, 2005. – 400 с.

12. Вайсфельд М. Объектно-ориентированное мышление / М. Вайсфельд. – СПб. : Питер, 2014. – 304 с.

13. Эккель Б. Философия Java. Библиотека программиста / Б. Эккель. – 4-е изд. – СПб. : Питер, 2009. – 640 с.

ПРИЛОЖЕНИЕ А
ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА

Министерство науки и высшего образования 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
«Объектно-ориентированный анализ и программирование»

Студент гр. ____
____ И. О. Фамилия
« ____ » _____ 201_ г.

Руководитель
доцент каф. АОИ,
канд. техн. наук
____ Ю. В. Морозова
« ____ » _____ 201_ г.

Томск 201_