|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практические РАБОТы

по дисциплине «Программирование на языка Джава»

Выполнил студент группы ИМБО-02-22 Ким К. С.

Принял старший преподаватель *Рачков А.В.*

Практические работы работа выполнены «\_\_\_»\_\_\_\_\_\_\_2023г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2023г.

Москва 2023

Оглавление

[Практическая работа №1 3](#_Toc150359842)

[Практическая работа №2 10](#_Toc150359843)

[Практическая работа №3 21](#_Toc150359844)

[Практическая работа №4 27](#_Toc150359845)

[Практическая работа №5 35](#_Toc150359846)

[Практическая работа №6 48](#_Toc150359847)

[Практическая работа №7 56](#_Toc150359848)

[Практическая работа №8 60](#_Toc150359849)

[Практическая работа №9 67](#_Toc150359850)

[Практическая работа №10 73](#_Toc150359851)

[Практическая работа №11 78](#_Toc150359852)

[Практическая работа №12 82](#_Toc150359853)

[Практическая работа №13 88](#_Toc150359854)

[Практическая работа №14 92](#_Toc150359855)

[Практическая работа №15 99](#_Toc150359856)

[Практическая работа №16 103](#_Toc150359857)

[Практическая работа №17 111](#_Toc150359858)

[Практическая работа №18 114](#_Toc150359859)

[Практическая работа №19 117](#_Toc150359860)

[Практическая работа №20 119](#_Toc150359861)

[Практическая работа №21 122](#_Toc150359862)

[Практическая работа №22 128](#_Toc150359863)

[Практические работы №23 и №24 134](#_Toc150359864)

[Используемая литература 135](#_Toc150359865)

# **Практическая работа №1**

**Цель работы:**

Освоить на практике работу с классами языка программирования Java.

**Теоретическое введение:**

В Java, класс является определением объектов одного и того же вида. Другими словами, класс — это тип данных, создаваемый программистом для решения задач. Он представляет из себя шаблон, или прототип, который определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида.

Экземпляр класса - реализация конкретного объекта типа класс. Другими словами, экземпляр экземпляра класса. Все экземпляры класса имеют аналогичные свойства, как задано в определении класса. Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент": " Петр", " Павел" и " Полина ". Термин "Объект " обычно относится к экземпляру класса. Но он часто используется свободно, которые могут относиться к классу или экземпляру.

**Выполнения практической работы**

*Задание №1:*

Реализуйте простейший класс «Собака».

*Решение*:

Данное решение представлено на Листинге – 1.1 и Листинге – 1.2.

*Листинг 1.1 – Dog.java*

|  |
| --- |
| *package dog;*  *import java.lang.\*;*  *public class Dog {*  *private String name;*  *private int age;*  *public Dog(String n, int a) {*  *name = n;*  *age = a;*  *}*  *public Dog(String n) {*  *name = n;*  *age = 0;*  *}*  *public Dog() {*  *name = "Pup";*  *age = 0;*  *}*  *public void setAge(int age) {*  *this.age = age;*  *}*  *public void setName(String name) {*  *this.name = name;*  *}*  *public String getName(String name) {*  *return name;*  *}*  *public int getAge() {*  *return age;*  *}*  *public String toString() {*  *return this.name + ", age " + this.age;*  *}* |

*Продолжение Листинга 1.1*

|  |
| --- |
| *public void intoHumanAge() {*  *System.out.println(name + "'s age in human years is " + age \* 7 + " years");*  *}*  *}* |

*Листинг 1.2 – TestDog.java*

|  |
| --- |
| package dog;  import java.lang.\*;  public class TestDog {  public static void main(String[] args) {  Dog d1 = new Dog("Mike", 2);  Dog d2 = new Dog("Helen", 7);  Dog d3 = new Dog("Bob");  d3.setAge(1);  System.out.println(d1);  d1.intoHumanAge();  d2.intoHumanAge();  d3.intoHumanAge();  }  } |

*Результат выполнения программ:*

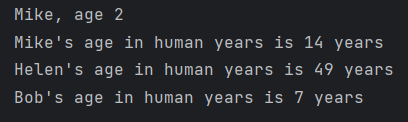


Рисунок 1.1 – Результат задачи №1

*Задание №2:*

Реализуйте простейший класс «Мяч».

*Решение*:

Данное решение представлено на Листинге – 1.3 и Листинге – 1.4.

*Листинг 1.3 – Ball.java*

|  |
| --- |
| package ball;  import java.lang.\*;  public class Ball {  private String name;  private String material;  public Ball(String n, String m) {  name = n;  material = m;  }  public Ball(String n) {  name = n;  }  public Ball() {  name = "Untitled";  material = "Untitled";  }  public void setName(String name) {  this.name = name;  }  public void setmaterial(String material) {  this.material = material;  }  public String getName(String name) {  return name;  }  public String getmaterial(String material) {  return material;  }  @Override  public String toString() {  return this.name + " мяч, Материал: "+ this.material;  }  } |

*Листинг 4 – TestBall.java*

|  |
| --- |
| package ball;  public class TestBall {  public static void main(String[] args) {  Ball b1 = new Ball("Футбольный", "латекс");  System.out.println(b1);  b1.setName("Баскетбольный");  b1.setmaterial("кожа");  System.out.println(b1);  }  } |

*Результат выполнения программ:*

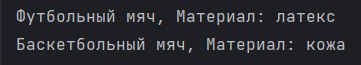


Рисунок 1.2 – Результат задачи №2

*Задание №3:*

Реализуйте простейший класс «Книга».

*Решение*:

Данное решение представлено на Листинге – 1.5 и Листинге – 1.6.

*Листинг 1.5 – Book.java*

|  |
| --- |
| package book;  import java.lang.\*;  public class Book {  private String name;  private String author;  private int year;  public Book(String n, String a, int y) {  name = n;  author = a;  year = y;  }  public Book(String n, String a) {  name = n;  author = a;  year = 0;  }  public Book() {  name = "Untitled";  author = "Untitled";  year = 0;  }  public void setYear(int year) {  this.year = year;  }  public void setName(String name) {  this.name = name;  }  public void setAuthor(String author) {  this.author = author;  }  public String getName(String name) {  return name;  }  public String getAuthor(String author) {  return author;  }  public int getYear() {  return year;  }  @Override  public String toString() {  return "Название книги: " + this.name + ", Автор книги: " + this.author + ", Год выпуска: " + this.year;  }  } |

*Листинг 1.6 – TestBook.java*

|  |
| --- |
| package book;  public class TestBook {  public static void main(String[] args) {  Book b1 = new Book("Преступление и наказание", "Ф.М.Достоевский", 1866);  Book b2 = new Book("1984", "Джордж Оруэлл", 1949);  Book b3 = new Book("Игра престолов", "Джордж Р.Р. Мартин");  b3.setYear(1996);  System.out.println(b1);  System.out.println(b2);  System.out.println(b3);  }  } |

*Результат выполнения программ:*

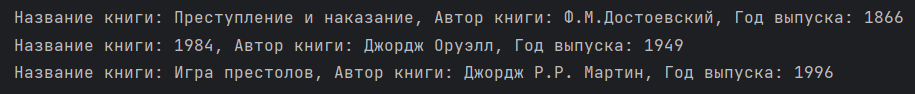


Рисунок 1.3 – Результат задачи №3

*Выводы по работе:*

В результате выполнения работы научились понимать классы, чем отличается модификатор контроля доступа public и private. Создавать методы классов, конструкторы на языке Java.

# **Практическая работа №2**

**Цель работы:**

Целями данной работы являются получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

**Теоретическое введение:**

Язык Джава — это объектно-ориентированный язык программирования, с со строгой инкапсуляцией и типизацией. Программы, написанные на языке, Джава могут выполняться под управлением различных операционных системах при наличии необходимого ПО – Java Runtime Environment. Для того чтобы создать и запускать программы на языке Джава необходимо следующее ПО:

* Java Development Kit (JDK);
* Java Runtime Environment (JRE);
* Среда разработки. Например, IDE IntelliJ IDEA или NetBeans

**Выполнения практической работы**

*Задание №1:*

Написать программу, в результате которой массив чисел создается с помощью инициализация (как в Си) вводится и считается в цикле сумма элементов целочисленного массива, а также среднее арифметическое его элементов результат выводится на экран. Использовать цикл for.

*Решение*:

Данное решение представлено на Листинге – 2.1.

*Листинг 2.1 – array\_1.java*

|  |
| --- |
| package ru.mirea.lab1;  import java.util.Scanner;  public class array\_1 {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int num = in.nextInt();  double a = 0;  int[] arr = new int[num];  for (int i = 0; i < num; i++) {  arr[i] = in.nextInt();  a += arr[i];  }  System.out.println(a);  double ans = a / num;  System.out.println(ans);  }  } |

*Результат выполнения программ:*

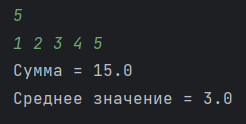
**

Рисунок 2.1 – Результат задачи №1

*Задание №2:*

Написать программу, в результате которой массив чисел вводится пользователем с клавиатуры считается сумма элементов целочисленного массива с помощью циклов do while, while, также необходимо найти максимальный и минимальный элемент в массиве, результат выводится на экран.

*Решение*:

Данное решение представлено на Листинге – 2.2.

*Листинг 2.2 – while\_2.java*

|  |
| --- |
| package ru.mirea.lab1;  import java.util.Scanner;  import java.util.ArrayList;  import java.util.Collections;  public class while\_2 {  public static void main(String[] args) {  System.out.println("Решаем через while");  System.out.println("Введи число");  Scanner in = new Scanner(System.in);  int num = in.nextInt();  if (num > 0) {  var arr = new ArrayList<Integer>(num);  boolean logic = true;  while (logic) {  int i = in.nextInt();  arr.add(i); |

*Продолжение Листинга 2*

|  |
| --- |
| if (arr.size() == num) {  break;  }  }  int min = Collections.min(arr);  int max = Collections.max(arr);  System.out.println(min);  System.out.println(max);  }  else {  System.out.println("Ввел не то число");  }  System.out.println("Решаем через do-while");  System.out.println("Введи число");  int num1 = in.nextInt();  if (num1 > 0) {  var arr1 = new ArrayList<Integer>(num1);  do {  int i = in.nextInt();;  arr1.add(i);  num1--;  } while (num1 != 0);  int min1 = Collections.min(arr1); |

*Продолжение Листинга 2*

|  |
| --- |
| int max1 = Collections.max(arr1);  System.out.println(min1);  System.out.println(max1);  }  else {  System.out.println("Ввел не то число");  }  }  } |

*Результат выполнения программ:*

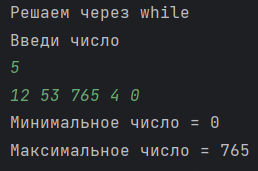
**

Рисунок 2.2 – Результат задачи №2

*Задание №3:*

Написать программу, в результате которой выводятся на экран аргументы командной строки в цикле for.

*Решение*:

Данное решение представлено на Листинге – 2.3.

*Листинг 2.3 – arguments\_3.java*

|  |
| --- |
| package ru.mirea.lab1;  public class arguments\_3 {  public static void main(String[] args) {  System.out.println("Hello, World!");  for (int i = 1; i <= args.length; i++) {  System.out.println("Аргумент №" + i + " - " + args[i]);  }  }  } |

*Результат выполнения программ:*

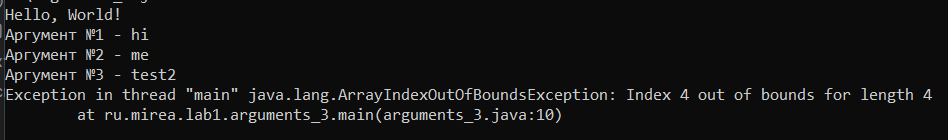


Рисунок 2.3 – Результат задачи №3

*Задание №4:*

Написать программу, в результате работы которой выводятся на экран первые 10 чисел гармонического ряда (форматировать вывод).

*Решение*:

Данное решение представлено на Листинге – 2.4.

*Листинг 2.4 – harmonic\_view\_4.java*

|  |
| --- |
| package ru.mirea.lab1;  import java.util.Scanner;  public class harmonic\_view\_4 {  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  int num = 10;  double n = 0;  for (int i = 1; i <= num; i++) {  n += (double) 1 / i;  System.out.println(i + " знаменатель = " + n);  }  System.out.println(n);  }  } |

*Результат выполнения программ*:

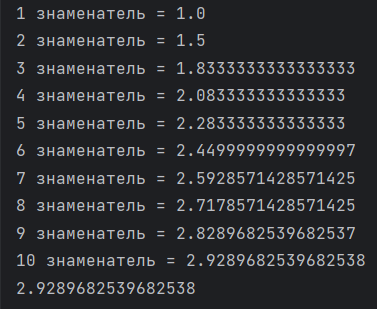


Рисунок 2.4 – Результат задачи №4

*Задание №5:*

Написать программу, которая с помощью метода класса, вычисляет факториал числа (использовать управляющую конструкцию цикла), проверить работу метода.

*Решение*:

Данное решение представлено на Листинге – 2.5.

*Листинг 2.5 – factorial\_5.java*

|  |
| --- |
| package ru.mirea.lab1;  import java.util.Scanner;  public class factorial\_5 {  public static void main(String[] args) {  while (true) {  System.out.println("Введи число больше 0 и меньше 20!");  Scanner in = new Scanner(System.in);  long num = in.nextInt();  if (num < 0) {  System.out.println("Введи число больше 0!");  }  else if (num > 20) {  System.out.println("Введи другое меньше 20!");  }  else {  long prod = 1;  for (int i = 1; i <= num; i++) {  prod \*= i;  }  System.out.println(prod);  break;  }  }  }  } |

*Результат выполнения программ*:

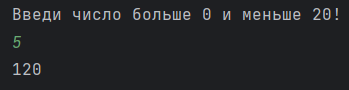
****

Рисунок 2.5 – Результат задачи №5

*Выводы по работе:*

В результате выполнения работы научились создавать проекты в IDE IntelliJ IDEA и создавать свой собственный Git репозиторий. Узнали, как пользовать переменными, массивами, условными/цикловыми конструкциями и методами в языке Java.

# **Практическая работа №3**

**Цель работы:**

Работа с UML-диаграммами классов

**Теоретическое введение:**

Язык моделирования Unified Modeling Language (UML) является стандартом де-факто с 1998 года для проектирования и документирования объектно-ориентированных программ. Средствами UML в виде диаграмм можно графически изобразить класс и экземпляр класса.

**Выполнения практической работы**

*Задание №1:*

По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 3.0.

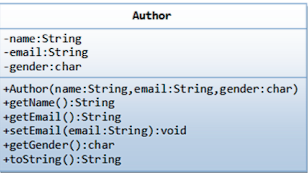
**

Рисунок 3.0 – Диаграмма класса Author

*Решение*:

Данное решение представлено на Листинге – 3.1 и Листинге – 3.2.

*Листинг 3.1 – Author.java*

|  |
| --- |
| package mirea.lab2\_1;  import java.lang.\*;  public class Author {  private String name;  private String email;  private char gender;  public Author (String name, String email, char gender) {  this.name = name;  this.email = email;  this.gender = gender;  }  public String getName() {  return name;  }  public String getEmail() {  return email;  } |

*Продолжение Листинга 3.1*

|  |
| --- |
| public void setEmail(String email) {  this.email = email;  }  public char getGender() {  return gender;  }  @Override  public String toString() {  return "Name " + name + ", Email " + email + ", Gender " + gender;  }  } |

*Листинг 3.2 – TestAuthor.java*

|  |
| --- |
| package mirea.lab2\_1;  import java.lang.\*;  public class TestAuthor {  public static void main(String[] args) {  Author test = new Author("Kirill", "noobmaster6969@ya.ru", 'm');  test.setEmail("sss@wer.wer");  System.out.println(test.toString());  }  } |

*Результат выполнения программ*:

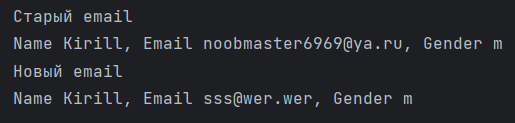
****

Рисунок 3.1 – Результат задачи №1

*Задание №2:*

По UML диаграмме класса, представленной на рисунке 3.0.1 написать

программу, которая состоит из двух классов. Один из них Ball должен

реализовывать сущность мяч, а другой с названием TestBall тестировать

работу созданного класса. Класс Ball должен содержать реализацию методов,

представленных на UML. Диаграмма на рисунке описывает сущность Мяч

написать программу.

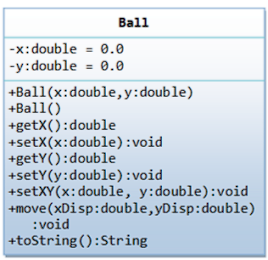
**

Рисунок 3.0.1 – Диаграмма класса Ball

*Решение*:

Данное решение представлено на Листинге – 3.3 и Листинге – 3.4.

*Листинг 3.3 – Ball.java*

|  |
| --- |
| package mirea.lab2\_2;  public class Ball {  private double x = 0.0;  private double y = 0.0;  public Ball (double x, double y) {  setXY(x, y);  }  private double x = 0.0;  private double y = 0.0;  public Ball (double x, double y) {  setXY(x, y); } |

*Продолжение Листинга 3.3*

|  |
| --- |
| public Ball (double x, double y) {  setXY(x, y);  }  public double getX() {  return x;  }  public void setX(double x) {  this.x = x;  }  public double getY() {  return y;  }  public void setY(double y) {  this.y = y;  }  public void setXY(double x, double y) {  this.x = x;  this.y = y;  }  public void move(double xDisp, double yDisp) {  setXY(x + xDisp, y + yDisp);  }  @Override  public String toString() {  return "Ball: x = " + x + ", y = " + y;  }  } |

*Листинг 3.4 – TestBall.java*

|  |
| --- |
| package mirea.lab2\_2;  import java.lang.\*;  public class TestBall {  public static void main(String[] args) {  Ball ball1 = new Ball(10, 10);  System.out.println(ball1.toString());  ball1.move(10, 5);  System.out.println(ball1.toString());  Ball ball2 = new Ball(5, 3);  System.out.println(ball2.toString());  } |

*Результат выполнения программ*:

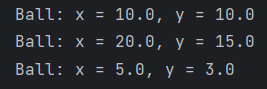
****

Рисунок 3.2 – Результат задачи №2

*Выводы по работе:*

В результате выполнения научились работать с UML-диаграммами классов. Это удобно представлять графически представлять класс в виде прямоугольника.

# **Практическая работа №4**

**Цель работы:**

Изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

**Теоретическое введение:**

Язык Java - объектно-ориентированный язык программирования. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы. Сокрытие данных называется инкапсуляцией.

Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования.

Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм подтипов — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Класс в ООП — это в чистом виде абстрактный тип данных, создаваемый программистом. С этой точки зрения объекты являются значениями данного абстрактного типа, а определение класса задаёт внутреннюю структуру значений и набор операций, которые над этими значениями могут быть выполнены. Желательность иерархии классов (а значит, наследования) вытекает из требований к повторному использованию кода — если несколько классов имеют сходное поведение, нет смысла дублировать их описание, лучше выделить общую часть в общий родительский класс, а в описании самих этих классов оставить только различающиеся элементы.

Необходимость совместного использования объектов разных классов, способных обрабатывать однотипные сообщения, требует поддержки полиморфизма — возможности записывать разные объекты в переменные одного и того же типа. В таких условиях объект, отправляя сообщение, может не знать в точности, к какому классу относится адресат, и одни и те же сообщения, отправленные переменным одного типа, содержащим объекты разных классов, вызовут различную реакцию.

**Выполнения практической работы**

*Задание №1:*

Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения, изменения этих свойств. Протестировать работу класса в классе CircleTest, содержащим метод статический main(String[] args).

*Решение*:

Данное решение представлено на Листинге – 4.1 и Листинге – 4.2.

*Листинг 4.1 – Circle.java*

|  |
| --- |
| package circle;  public class Circle {  private double radius;  public Circle(double radius) {  this.radius=radius;  }  public double getRadius() {  return radius;  }  public void setRadius(double radius) {  this.radius=radius;  }  public double getArea() {  return Math.PI \* radius \* radius;  }  public double getPerimeter() {  return Math.PI \* radius \* 2;  } |

*Продолжение Листинга 4.1*

|  |
| --- |
| @Override  public String toString() {  return "Circle{" +  "radius=" + radius +  '}';  }  } |

*Листинг 4.2 – CircleTest.java*

|  |
| --- |
| package circle;  public class CircleTest {  public static void main(String[] args) {  Circle c = new Circle(5);  System.out.println("S = " + c.getArea());  System.out.println("P = " + c.getPerimeter());  System.out.println(c.toString());  c.setRadius(10);  System.out.println("Новый радиус: " + c.getRadius());  System.out.println("S = " + c.getArea());  System.out.println("P = " + c.getPerimeter());  }  } |

*Результат выполнения программ*:

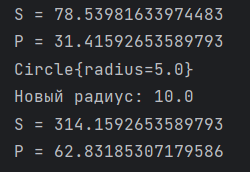
****

Рисунок 4.1 – Результат задачи №1

*Задание №2:*

Создать класс, описывающий тело человека (Human). Для описания каждой части тела создать отдельные классы (Head, Leg, Hand). Описать необходимые свойства и методы для каждого класса. Протестировать работу класса Human

*Решение*:

Данное решение представлено на Листинге – 4.3 и Листинге – 4.4.

*Листинг 4.3 – Human.java*

|  |
| --- |
| package human;  public class Human {  private String name;  private int age;  public Human(String name, int age) {  this.name = name;  this.age = age;  }  public String get\_name() {  return this.name;  }  public void set\_name(String name) {  this.name = name;  }  public int get\_age() {  return this.age;  }  public void set\_age(int age) {  if (0 < age && age < 100) {  this.age = age;  }  }  @Override  public String toString() {  return "Имя человека: " + this.name + ", Возраст: " + this.age;  }  public class Head {  private boolean hurt;  public boolean get\_hurt() {  return this.hurt;  } |

*Продолжение Листинга 4.3*

|  |
| --- |
| public void set\_hurt(boolean hurt) {  this.hurt = hurt;  }  public void h(boolean hurt) {  if (hurt) {  System.out.println("Голова болит");  }  else {  System.out.println("Голова не болит");  }  }  }  public class Hand {  private boolean clap;  public boolean get\_clap() {  return this.clap;  }  public void set\_clap(boolean clap) {  this.clap = clap;  }  public void c(boolean clap) {  if (clap) {  System.out.println("Стоим на руках!");  }  else {  System.out.println("Руки ничего не делают");  }  }  }  public class Leg {  private boolean kick;  public boolean get\_kick() {  return this.kick;  }  public void set\_kick(boolean kick) {  this.kick = kick;  }  public void k(boolean kick) {  if (kick) {  System.out.println("Ногой стучим ритм!");  } |

*Продолжение Листинга 4.3*

|  |
| --- |
| else {  System.out.println("Ноги ничего не делают");  }  }  }  } |

*Листинг 4.4 – HumanTest.java*

|  |
| --- |
| package human;  public class HumanTest {  public static void main(String[] args){  Human human = new Human("Кирилл", 18);  human.set\_age(19);  System.out.println(human);  Human.Head human\_head = human.new Head();  human\_head.h(true);  Human.Hand human\_hand = human.new Hand();  human\_hand.c(false);  Human.Leg human\_leg = human.new Leg();  human\_leg.k(false);  }  } |

*Результат выполнения программ*:

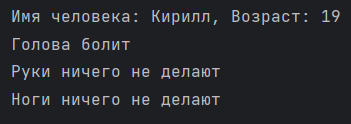
****

Рисунок 4.2 – Результат задачи №2

*Задание №3:*

Создать класс, описывающий книгу (Book). В классе должны быть описаны нужные свойства книги (автор, название, год написания и т. д.)и методы для получения, изменения этих свойств. Протестировать работу класса в классе BookTest, содержащим метод статический main(String[] args).

*Решение*:

Данное решение представлено на Листинге – 1 и Листинге – 2.

*Листинг 1 – Book.java*

|  |
| --- |
| package book;  public class Book {  private String author;  private String name;  private int year;  public Book(String a, String n, int y) {  author = a;  name = n;  year = y;  }  public Book(String a, String n) {  author = a;  name = n;  year = 0;  }  public Book() {  author = "Untitled";  name = "Untitled";  year = 0;  }  public void setYear(int year) {  this.year = year;  }  public void setName(String name) {  this.name = name;  }  public void setAuthor(String author) {  this.author = author;  }  public String getName(String name) { |

*Продолжение Листинга 1*

|  |
| --- |
| return name;  }  public String getAuthor(String author) {  return author;  }  public int getYear() {  return year;  }  @Override  public String toString() {  return "Автор книги: " + this.author + ", Название книги: " + this.name + ", Год выпуска: " + this.year;  }  } |

*Листинг 2 – BookTest.java*

|  |
| --- |
| package book;  public class BookTest {  public static void main(String[] args) {  Book b1 = new Book("Преступление и наказание", "Ф.М.Достоевский", 1865);  System.out.println(b1);  b1.setYear(1866);  System.out.println(b1);  }  } |

*Результат выполнения программ*:

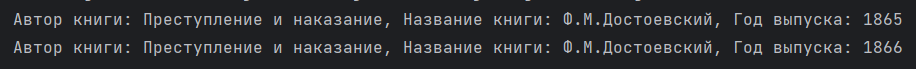
****

Рисунок 4.3 – Результат задачи №3

*Выводы по работе:*

В результате выполнения изучили основные концепции объектно-ориентированного программирования, изучили понятие класса и научились создавать классы.

# **Практическая работа №5**

**Цель работы:**

Освоить на практике работу с абстрактными классами и наследованием на Java.

**Теоретическое введение:**

Класс, содержащий абстрактные методы, называется абстрактным

классом. Такие классы при определении помечаются ключевым словом

abstract.

Абстрактный метод внутри абстрактного класса не имеет тела, только

прототип. Он состоит только из объявления и не имеет тела:

abstract void yourMethod();

По сути, мы создаём шаблон метода. Например, можно создать

абстрактный метод для вычисления площади фигуры в абстрактном классе

Фигура. А все другие производные классы от главного класса могут уже

реализовать свой код для готового метода. Ведь площадь у прямоугольника и

треугольника вычисляется по разным алгоритмам и универсального метода

не существует.

Если вы объявляете класс, производный от абстрактного класса, но

хотите иметь возможность создания объектов нового типа, вам придётся

предоставить определения для всех абстрактных методов базового класса.

Если этого не сделать, производный класс тоже останется абстрактным, и

компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также

абстрактные конструкторы и абстрактные статические методы. Любой

подкласс абстрактного класса должен либо реализовать все абстрактные

методы суперкласса, либо сам быть объявлен абстрактным.

**Выполнения практической работы**

*Задание №1:*

Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).

Перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square рисунка 5.0.

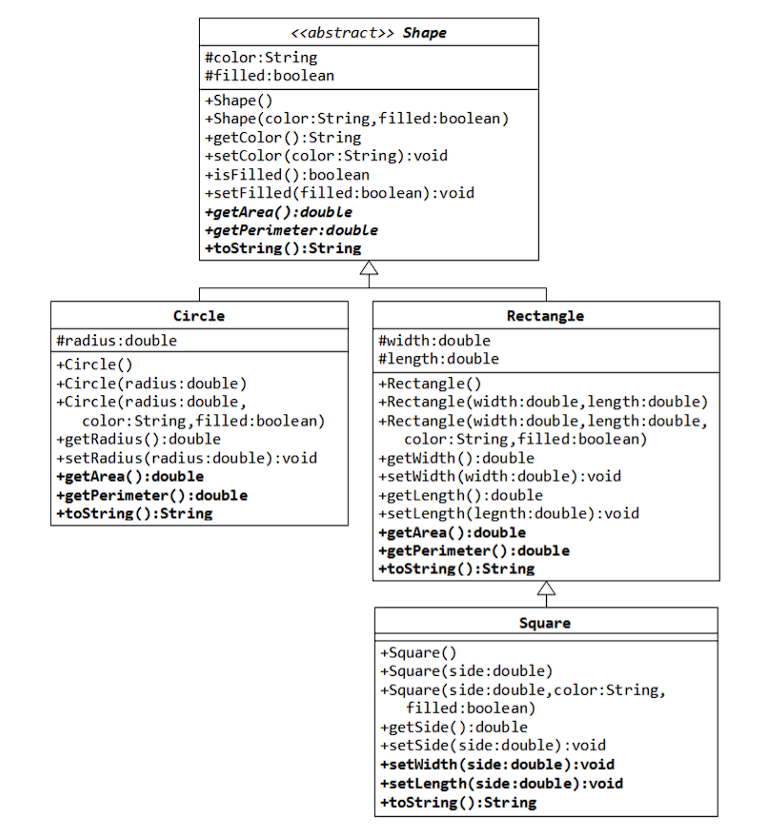


Рисунок 5.0 – Результат задачи №1

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3, Листинге 4 и Листинге – 5.

*Листинг 1 – Shape.java*

|  |
| --- |
| package mirea.shape;  public abstract class Shape {  protected String color;  protected boolean filled;  public Shape() {  color = "Green";  filled = false;  }  public Shape(String color, boolean filled) {  this.color = color;  this.filled = filled;  }  public String getColor() {  return color;  }  public void setColor(String color) {  this.color = color;  }  public boolean isFilled() {  return filled;  }  public void setFilled(boolean filled) {  this.filled = filled;  }  public abstract double getArea();  public abstract double getPerimeter();  @Override  public String toString() {  return "Shape{" +  "colour='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

*Листинг 2 – Circle.java*

|  |
| --- |
| package mirea.shape;  public class Circle extends Shape {  protected double radius;  public Circle() {  super();  radius = 1;  }  public Circle(double radius) {  super();  this.radius = radius;  }  public Circle(double radius, String colour, boolean filled) {  super(colour, filled);  this.radius = radius;  }  public double getRadius() {  return radius;  }  public void setRadius(double radius) {  this.radius = radius;  }  @Override  public double getArea() {  return Math.PI \* radius \* radius;  }  @Override  public double getPerimeter() {  return Math.PI \* radius \* 2;  }  @Override  public String toString() {  return "Circle{" +  "radius=" + radius +  ", colour='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

*Листинг 3 – Rectangle.java*

|  |
| --- |
| package mirea.shape;  public class Rectangle extends Shape {  protected double width, length;  public Rectangle() {  super();  length = 1;  width = 1;  }  public Rectangle(double width, double length) {  super();  this.width = width;  this.length = length;  }  public Rectangle(double width, double length, String colour, boolean filled) {  super(colour, filled);  this.width = width;  this.length = length;  }  public double getWidth() {  return width;  }  public void setWidth(double width) {  this.width = width;  }  public double getLength() {  return length;  }  public void setLength(double length) {  this.length = length;  }  @Override  public double getArea() {  return width \* length;  }  @Override  public double getPerimeter() {  return 2 \* (width + length);  }  @Override  public String toString() { |

*Продолжение Листинга 3*

|  |
| --- |
| return "Rectangle{" +  "width=" + width +  ", length=" + length +  ", colour='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

*Листинг 4 – Square.java*

|  |
| --- |
| package mirea.shape;  public class Square extends Rectangle {  public Square() {  super();  }  public Square(double side) {  super(side, side);  }  public Square(double side, String colour, boolean filled) {  super(side, side, colour, filled);  }  public double getSide() {  return width;  }  public void setSide(double side){  width = side;  length = side;  }  @Override  public void setWidth(double width) {  super.setWidth(width);  super.setLength(length);  }  @Override  public void setLength(double length) {  super.setLength(length);  super.setWidth(width);  }  @Override  public String toString() { |

*Продолжение Листинга 4*

|  |
| --- |
| return "Square{" +  "width=" + width +  ", length=" + length +  ", colour='" + color + '\'' +  ", filled=" + filled +  '}';  }  } |

*Листинг 5 – TestShape.java*

|  |
| --- |
| package mirea.shape;  public class TestShape {  public static void main(String[] args) {  Shape s1 = new Circle(5.5, "RED", false); // Upcast Circle to  System.out.println(s1); // which version?  System.out.println(s1.getArea()); // which version?  System.out.println(s1.getPerimeter()); // which version?  System.out.println(s1.getColor());  System.out.println(s1.isFilled());  Shape s3 = new Rectangle(1.0, 2.0, "RED", false); // upcast  System.out.println(s3);  System.out.println(s3.getArea());  System.out.println(s3.getPerimeter());  System.out.println(s3.getColor());  Rectangle r1 = (Rectangle)s3; // downcast  System.out.println(r1);  System.out.println(r1.getArea());  System.out.println(r1.getColor());  System.out.println(r1.getLength());  Shape s4 = new Square(6.6); // Upcast  System.out.println(s4);  System.out.println(s4.getArea());  System.out.println(s4.getColor());  Rectangle r2 = (Rectangle)s4;  System.out.println(r2);  System.out.println(r2.getArea());  System.out.println(r2.getColor());  System.out.println(r2.getLength());  Square sq1 = (Square)r2;  System.out.println(sq1); |

*Результат выполнения программ*:

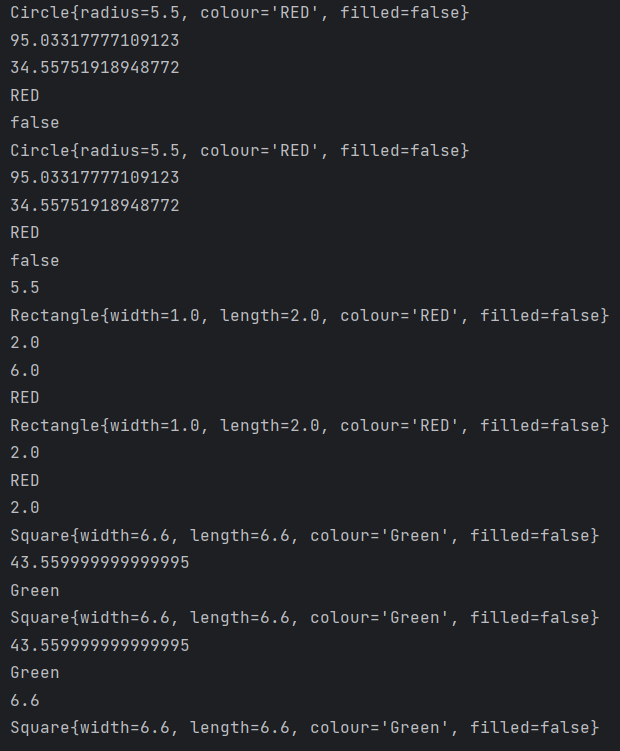
****

Рисунок 5.1 – Результат задачи №1

*Задание №2:*

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

Напишите новый класс MovableRectangle (движущийся прямоугольник). Его можно представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable. Убедитесь, что две точки имеет одну и ту же скорость (нужен метод это проверяющий).

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3, Листинге – 4 и Листинге – 5.

*Листинг 1 – Movable.java*

|  |
| --- |
| package mirea.mov;  public interface Movable {  void moveUp();  void moveDown();  void moveLeft();  void moveRight();  } |

*Листинг 2 – MovablePoint.java*

|  |
| --- |
| package mirea.mov;  public class MovablePoint implements Movable {  protected int x;  protected int y;  protected int xSpeed;  protected int ySpeed;  MovablePoint(int x, int y, int xSpeed, int ySpeed) {  this.x = x;  this.y = y;  this.xSpeed = xSpeed;  this.ySpeed = ySpeed;  }  @Override  public String toString() {  return "Координаты: (" + x + " " + y + "); Сдвиг " + xSpeed + " " + ySpeed; |

*Продолжение Листинга 2*

|  |
| --- |
| }  @Override  public void moveUp() {  this.y += this.ySpeed;  System.out.println("Вверх " + this.ySpeed);  }  @Override  public void moveDown() {  this.y -= this.ySpeed;  System.out.println("Вниз " + this.ySpeed);  }  @Override  public void moveLeft() {  this.x -= this.xSpeed;  System.out.println("Налево " + this.xSpeed);  }  @Override  public void moveRight() {  this.x += this.xSpeed;  System.out.println("Направо " + this.xSpeed);  }  } |

*Листинг 3 – MovableCircle.java*

|  |
| --- |
| package mirea.mov;  public class MovableCircle implements Movable {  private int radius;  MovablePoint center;  MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {  this.center = new MovablePoint(x, y, xSpeed, ySpeed);  this.radius = radius;  }  @Override  public String toString() {  return "Радиус = " + radius + "; центр = " + center.toString();  }  @Override  public void moveUp() {  center.moveUp();  } |

*Продолжение Листинга 3*

|  |
| --- |
| @Override  public void moveDown() {  center.moveDown();  }  @Override  public void moveLeft() {  center.moveLeft();  }  @Override  public void moveRight() {  center.moveRight(); }  } |

*Листинг 4 – MovableRectangle.java*

|  |
| --- |
| package mirea.mov;  public class MovableRectangle extends MovablePoint implements Movable {  MovablePoint topLeft, bottomRight;  boolean hasCorrectSpeed() {  return topLeft.xSpeed == bottomRight.xSpeed && topLeft.ySpeed == bottomRight.ySpeed;  }  public MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int ySpeed) {  super((x1 + x2) / 2, (y1 + y2) / 2, xSpeed, ySpeed);  topLeft = new MovablePoint(x1, y1, xSpeed, ySpeed);  bottomRight = new MovablePoint(x2, y2, xSpeed, ySpeed);  }  @Override  public void moveUp() {  if (this.hasCorrectSpeed()) {  y += ySpeed;  topLeft.y += ySpeed;  bottomRight.y += ySpeed;  }  }  @Override  public void moveDown() {  if (this.hasCorrectSpeed()) {  y -= ySpeed;  topLeft.y -= ySpeed; |

*Продолжение Листинга 4*

|  |
| --- |
| bottomRight.y -= ySpeed;  }  }  @Override  public void moveLeft() {  if (this.hasCorrectSpeed()) {  x -= xSpeed;  topLeft.x -= xSpeed;  bottomRight.x -= xSpeed;  }  }  @Override  public void moveRight() {  if (this.hasCorrectSpeed()) {  x += xSpeed;  topLeft.x += xSpeed;  bottomRight.x += xSpeed;  }  }  } |

*Листинг 5 – Main.java*

|  |
| --- |
| package mirea.mov;  public class Main {  public static void main(String[] args) {  MovablePoint movable\_point = new MovablePoint(10, 0, 0, 1);  System.out.println(movable\_point);  movable\_point.moveDown();  movable\_point.moveRight();  System.out.println(movable\_point);  System.out.println();  MovableCircle movable\_circle = new MovableCircle(10, 0, 0, 1, 1);  System.out.println(movable\_circle);  movable\_circle.moveUp();  movable\_circle.moveRight();  System.out.println(movable\_circle);  System.out.println();  MovablePoint movable\_rectangle = new MovableRectangle(0, 0, 4, 4, 2, 4);  System.out.println(movable\_rectangle); |

*Продолжение Листинга 5*

|  |
| --- |
| movable\_rectangle.moveDown();  movable\_rectangle.moveLeft();  System.out.println(movable\_rectangle);  }  } |

*Результат выполнения программ*:

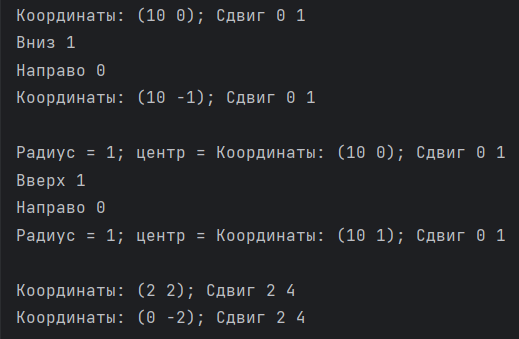
****

Рисунок 5.2 – Результат задачи №2

*Выводы по работе:*

В результате выполнения освоили абстрактные классы и наследование классов на Java.

# **Практическая работа №6**

**Цель работы:**

Цель данной работы - изучить понятие наследования, и научиться реализовывать наследование в Java.

**Теоретическое введение:**

Одним из ключевых аспектов объектно-ориентированного программирования является наследование. С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого.

Кроме обычных классов в Java есть абстрактные классы. Абстрактный класс похож на обычный класс. В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов- наследников. А производные классы уже реализуют этот функционал.

**Выполнения практической работы**

*Задание №1:*

Создать абстрактный класс, описывающий посуду (Dish). С помощью наследования реализовать различные виды посуды, имеющие свои свойства и методы. Протестировать работу классов.

*Решение*:

Данное решение представлено на Листинге – 1 и Листинге – 2.

*Листинг 1 – Dish.java*

|  |
| --- |
| package dish;  public abstract class Dish {  private String name;  private String type;  public Dish(String name, String type) {  this.name = name;  this.type = type;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getType() {  return type;  }  public void setType(String type) {  this.type = type;  }  public abstract void name();  }  class Plate extends Dish {  private String color;  public Plate(String name, String type, String color) {  super(name, type);  this.color = color;  }  public void name() {  System.out.println("Посуда: " + super.getName() + ", Материал: " + super.getType() + ", Цвет: " + color); |

*Продолжение Листинга 1*

|  |
| --- |
| }  }  class Tableware extends Dish {  private String inst;  public Tableware(String name, String type, String inst) {  super(name, type);  this.inst = inst;  }  public void name() {  System.out.println("Посуда: " + inst + ", Материал: " + super.getType());  }  } |

*Листинг 2 – Dish.java*

|  |
| --- |
| package dish;  public class TestDish {  public static void main(String[] args){  Dish dish = new Plate("Тарелка", "Керамика", "Белый");  dish.name();  Dish fork = new Tableware("Тарелка", "Металл", "Вилка");  fork.name();  }  } |

*Результат выполнения программ*:

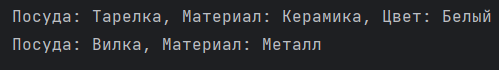
****

Рисунок 6.1 – Результат задачи №1

*Задание №2:*

Создать абстрактный класс, описывающий собак (Dog). С помощью наследования реализовать различные породы собак. Протестировать работу классов.

*Решение*:

Данное решение представлено на Листинге – 1 и Листинге – 2.

*Листинг 1 – Dog.java*

|  |
| --- |
| package dogs;  public abstract class Dog {  private String name;  private int age;  public Dog(String name, int age) {  this.name = name;  this.age = age;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "Собака: " + name + ", Лет: " + age;  }  }  class Type extends Dog {  private String type;  public Type(String name, int age, String type) {  super(name, age);  this.type = type;  }  @Override |

*Продолжение Листинга 1*

|  |
| --- |
| public String toString() {  System.out.println(super.toString());  return "Порода: " + type;  }  } |

*Листинг 2 – TestDog.java*

|  |
| --- |
| package dogs;  public class TestDog {  public static void main(String[] args){  Type dog = new Type("Ллойд", 5, "Ирландский сеттер");  System.out.println(dog.toString());  }  } |

*Результат выполнения программ*:

****

Рисунок 6.2 – Результат задачи №2

*Задание №3:*

Создать абстрактный класс, описывающий мебель. С помощью наследования реализовать различные виды мебели. Также создать класс FurnitureShop, моделирующий магазин мебели. Протестировать работу классов.

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2 и Листинге – 3.

*Листинг 1 – Furniture.java*

|  |
| --- |
| package furnitures;  public abstract class Furniture {  protected String name;  private String material;  public Furniture(String name, String material){  this.name = name;  this.material = material;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getMaterial() {  return material;  }  public void setMaterial(String material) {  this.material = material;  }  public abstract String cost();  @Override  public String toString() {  return "Мебель: " + name + ", Материал: " + material;  }  }  class Type extends Furniture {  private String type;  public Type(String name, String material, String type) { |

*Продолжение Листинга 1*

|  |
| --- |
| super(name, material);  this.type = type;  }  @Override  public String cost() {  return "Стоит на " + type;  }  @Override  public String toString() {  System.out.println(super.toString());  return name;  }  } |

*Листинг 2 – FurnitureShop.java*

|  |
| --- |
| package furnitures;  import java.util.ArrayList;  import java.util.List;  public class FurnitureShop {  private List<Furniture> furniture\_shop\_array = new ArrayList<Furniture>();  public void add\_furniture(Furniture furniture) {  this.furniture\_shop\_array.add(furniture);  System.out.println(furniture + ". Эта мебель успешно поступила в магазин");  }  public String toString() {  if (!this.furniture\_shop\_array.isEmpty()) {  return "В магазине есть: " + this.furniture\_shop\_array;  }  else {  return "Магазин пуст!";  }  }  } |

*Листинг 3 – TestFurniture.java*

|  |
| --- |
| package furnitures;  public class TestFurniture {  public static void main(String[] args){ |

*Продолжение Листинга 3*

|  |
| --- |
| Furniture f1 = new Type("Игровое кресло", "Экокожа", "колёсиках");  System.out.println(f1.toString() + " " + f1.cost());  System.out.println();  Furniture f2 = new Type("Стул", "Деревянный", "ножках");  System.out.println(f2.toString() + " " + f2.cost());  System.out.println();  FurnitureShop furniture\_shop = new FurnitureShop();  furniture\_shop.add\_furniture(f1);  furniture\_shop.add\_furniture(f2);  System.out.println(furniture\_shop);  }  } |

*Результат выполнения программ*:

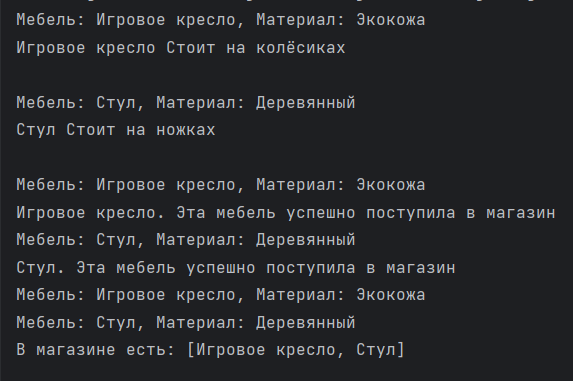
****

Рисунок 6.3 – Результат задачи №3

*Выводы по работе:*

В результате выполнения научились реализовывать наследование в Java.

# **Практическая работа №7**

**Цель работы:**

Введение в событийное программирование на языке Java.

**Теоретическое введение:**

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java c использованием следующих элементов GUI:

* Текстовые поля и области ввода текста;
* Менеджеры компоновки компонентов;
* Слушатель мыши;
* Создание меню.

**Выполнения практической работы**

*Задание №1:*

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение, у которого есть следующие компоненты GUI:

* одна кнопка JButton labeled “AC Milan”
* другая JButton подписана “Real Madrid”
* надпись JLabel содержит текст “Result: 0 X 0”
* надпись JLabel содержит текст “Last Scorer: N/A” 80
* надпись Label содержит текст “Winner: DRAW”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

*Решение*:

Данное решение представлено на Листинге – 1.

*Листинг 1 – FIFA.java*

|  |
| --- |
| package mirea.gui;  import java.awt.\*;  import javax.swing.\*;  import java.awt.event.\*;  class FIFA extends JFrame {  int milanCnt = 0;  int madridCnt = 0;  JButton milan = new JButton("AC Milan");  JButton madrid = new JButton("Real Madrid");  JLabel result = new JLabel("Result: 0 X 0");  JLabel lastScorer = new JLabel("Last Scorer: N/A");  Label winner = new Label("Winner: DRAW");  Font fnt1 = new Font("Times new roman", Font.BOLD, 25);  Font fnt2 = new Font("Serif", Font.BOLD, 25);  public FIFA() { |

*Продолжение Листинга 1*

|  |
| --- |
| super("Football Game");  setSize(300, 300);  setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);  setLayout(new FlowLayout());  milan.setFont(fnt1);  madrid.setFont(fnt1);  result.setFont(fnt1);  lastScorer.setFont(fnt1);  winner.setFont(fnt2);  milan.setBorderPainted(false);  milan.setFocusPainted(false);  milan.setBackground(Color.RED);  milan.setForeground(Color.BLACK);  madrid.setBorderPainted(false);  madrid.setFocusPainted(false);  madrid.setBackground(Color.WHITE);  madrid.setForeground(Color.BLUE);  add(milan);  add(madrid);  add(result);  add(lastScorer);  add(winner);  milan.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  milanCnt++;  result.setText("Result: " + milanCnt + " X " + madridCnt);  lastScorer.setText("Last Scorer: AC Milan");  if (milanCnt == madridCnt)  winner.setText("Winner: DRAW");  else if (milanCnt > madridCnt)  winner.setText("Winner: AC Milan");  else  winner.setText("Winner: Real Madrid");  }  });  madrid.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  madridCnt++;  result.setText("Result: " + milanCnt + " X " + madridCnt); |

*Продолжение Листинга 1*

|  |
| --- |
| lastScorer.setText("Last Scorer: Real Madrid");  if (milanCnt == madridCnt)  winner.setText("Winner: DRAW");  else if (milanCnt > madridCnt)  winner.setText("Winner: AC Milan");  else  winner.setText("Winner: Real Madrid");  }  });  }  public static void main(String[] args) {  new FIFA().setVisible(true);  }  } |

*Результат выполнения программ*:

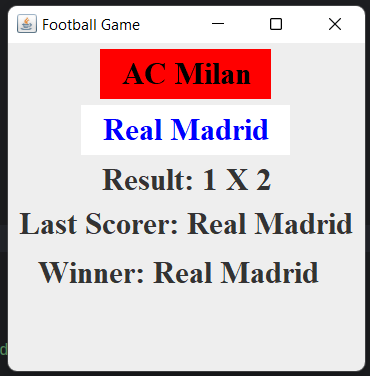
****

Рисунок 7.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились создавать приложение на Java c использованием следующих элементов GUI.

# **Практическая работа №8**

**Цель работы:**

Введение в событийное программирование на языке Java.

**Теоретическое введение:**

Научиться создавать графический интерфейс пользователя, освоить на практике работу с различными объектами для создания ГИП, менеджерами размещения компонентов.

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java c использованием следующих элементов GUI:

* Текстовые поля и области ввода текста;
* Менеджеры компоновки компонентов;
* Слушатель мыши;
* Создание меню.

*Задание №1:*

Создать окно, нарисовать в нем 20 случайных фигур, случайного цвета. Классы фигур должны наследоваться от абстрактного класса Shape, в котором описаны свойства фигуры: цвет, позиция.

Создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.

Создать окно, реализовать анимацию, с помощью картинки, состоящей из нескольких кадров.

*Решение*:

Данное решение представлено на Листинге – 1.

*Листинг 1 – App.java*

|  |
| --- |
| package app;  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  import java.io.File;  class App extends JFrame {  int WINDOW\_WIDTH = 600;  int WINDOW\_HEIGHT = 600;  String background\_image\_path;  String animation\_images\_path = "C:\\Users\\super\\IdeaProjects\\MIREA\\practice\_\_8\\src\\app\\frames";  int method;  // Init  App() {  super("Some shapes");  setSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);  setLocation(300, 300);  setLayout(null);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setVisible(true);  this.method = 0;  Button btn = new Button("Start animation");  btn.setSize(200, 100);  btn.setLocation(0, 0);  btn.addActionListener(  new ActionListener () {  public void actionPerformed(ActionEvent event) { |

*Продолжение Листинга 1*

|  |
| --- |
| method = 1;  setTitle("Animation");  setSize(WINDOW\_WIDTH + 1, WINDOW\_HEIGHT);  setSize(WINDOW\_WIDTH - 1, WINDOW\_HEIGHT);  }  }  );  add(btn);  }  void set\_background\_image\_path(String path) {  this.background\_image\_path = path;  }  void set\_animation\_images\_path(String path) {  this.animation\_images\_path = path;  }  @Override  public void paint(Graphics g2) {  Graphics2D g = (Graphics2D) g2;  Image img = Toolkit.getDefaultToolkit().getImage(this.background\_image\_path);  g.drawImage(img, 0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, this);  switch (this.method) {  case 0:  random\_shapes\_paint(50, g);  break;  case 1:  animate(this.animation\_images\_path, g);  break;  default:  break;  }  }  void random\_shapes\_paint(int shapes, Graphics2D g) {  for (int i = 0; i < shapes; i++) {  int choice = (int) (Math.random() \* 7);  switch (choice) {  // Oval  case 1:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.fillOval( |

*Продолжение Листинга 1*

|  |
| --- |
| (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* 200),  (int) (Math.random() \* 200)  );  break;  // Rect  case 2:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.fillRect(  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* 200),  (int) (Math.random() \* 200)  );  break;  // Line  case 3:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.drawLine(  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* WINDOW\_HEIGHT)  );  break;  // Arc  case 4:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.drawArc(  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* 360),  );  break; |

*Продолжение Листинга 1*

|  |
| --- |
| // Rounded rect  case 5:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.fillRoundRect(  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  (int) (Math.random() \* 200),  (int) (Math.random() \* 200),  (int) (Math.random() \* 200),  (int) (Math.random() \* 200)  );  break;  // Circle  case 6:  g.setColor(new Color((int) (Math.random() \* 0x1000000)));  g.fillOval(  (int) (Math.random() \* WINDOW\_WIDTH),  (int) (Math.random() \* WINDOW\_HEIGHT),  100,  100  );  break;  default:  break;  }  }  }  void animate(String frames\_path, Graphics2D g) {  File dir = new File(frames\_path);  for (File file : dir.listFiles()) {  Image frame = Toolkit.getDefaultToolkit().getImage(file.getPath());  g.drawImage(frame, 0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT, this);  try { Thread.sleep(500); } catch (InterruptedException e) {}  }  } |

*Продолжение Листинга 1*

|  |
| --- |
| public static void main(String[] args) {  App app = new App();  if (args.length == 1) {  app.set\_background\_image\_path(args[0]);  //System.out.println(app.background\_image\_path);  }  else if (args.length == 2) {  app.set\_background\_image\_path(args[0]);  app.set\_animation\_images\_path(args[1]);  }  else {  System.out.println("[!] Background image path is empty");  }  }  } |

*Результат выполнения программ*:

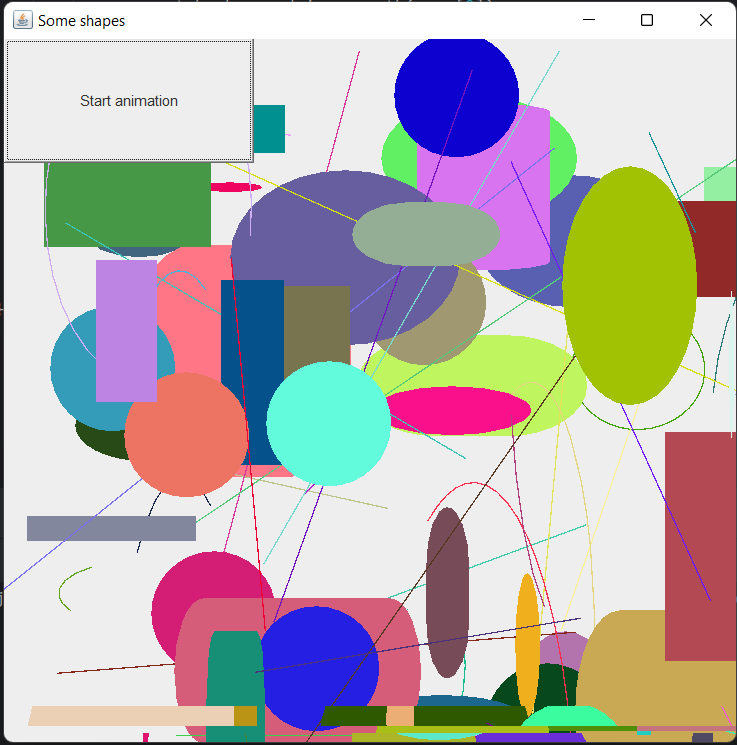
****

Рисунок 8.1 – Результат задачи №1

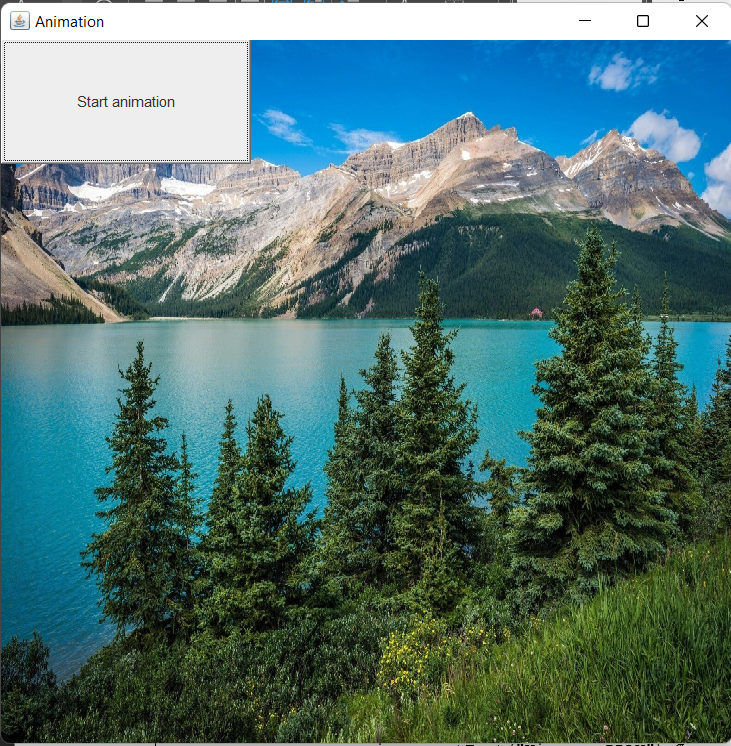
****

Рисунок 8.2 – Результат задачи №1

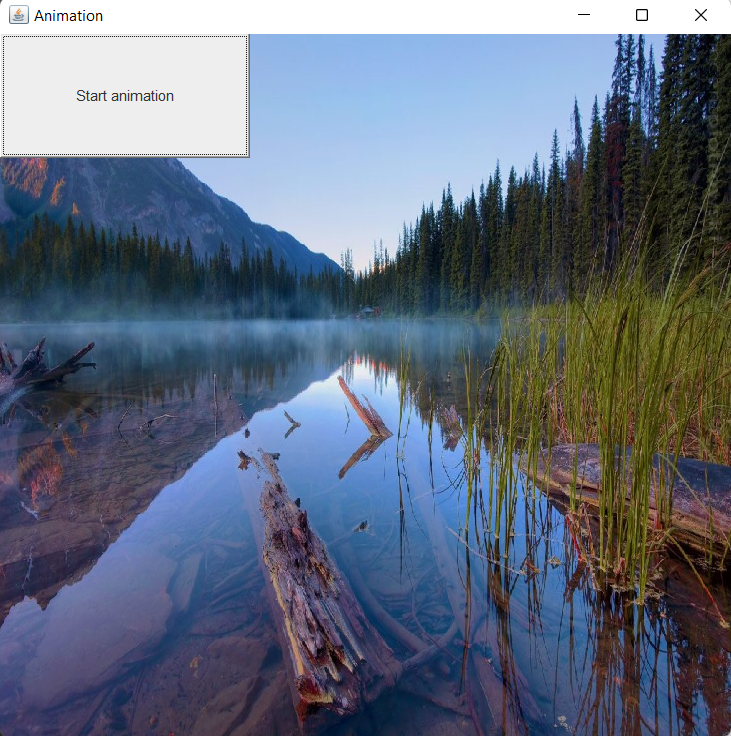
****

Рисунок 8.3 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились создавать анимацию и разные фигуры на языке Java, использовав GUI.

# **Практическая работа №9**

**Цель работы:**

Цель данной практической работы - изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

**Теоретическое введение:**

Механизм наследования очень удобен, но он имеет свои ограничения. В частности, мы можем наследовать только от одного класса, в отличие, например, от языка С++, где имеется множественное наследование.

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. Чтобы определить интерфейс, используется ключевое слово interface.

Интерфейс может определять различные методы, которые, так же, как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод. Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации. И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа. Интерфейс может определять различные методы, которые, так же, как и абстрактные методы абстрактных классов не имеют реализации. В данном случае объявлен только один метод.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации. И также при объявлении интерфейса надо учитывать, что только один интерфейс в файле может иметь тип доступа public. А его название должно совпадать с именем файла. Остальные интерфейсы (если такие имеются в файле java) не должны иметь модификаторов доступа.

*Задание №1:*

Создать интерфейс Nameable, с методом getName(), возвращающим имя объекта, реализующего интерфейс. Проверить работу для различных объектов (например, можно создать классы, описывающие разные сущности, которые могут иметь имя: планеты, машины, животные и т. д.).

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3 и Листинге – 4.

*Листинг 1 – Nameable.java*

|  |
| --- |
| package Name;  public interface Nameable {  String getName();  } |

*Листинг 2 – Planet.java*

|  |
| --- |
| package Name;  public class Planet implements Nameable {  private String name;  Planet(String name){  this.name = name;  }  public String getName() {  return name;  }  } |

*Листинг 3 – Cars.java*

|  |
| --- |
| package Name;  public class Cars implements Nameable{  private String model;  Cars(String model){  this.model = model;  }  public String getName() {  return model;  }  } |

*Листинг 4 – Test.java*

|  |
| --- |
| package Name;  public class Test {  public static void main(String[] args) {  Nameable printable = new Planet("Земля");  System.out.println("Планета: " + printable.getName());  printable = new Cars("Lexus");  System.out.println("Марка автомобиля: " + printable.getName());  }  } |

*Результат выполнения программ*:

****

Рисунок 9.1 – Результат задачи №1

*Задание №2:*

Реализовать интерфейс Priceable, имеющий метод getPrice(),

возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3 и Листинге – 4.

*Листинг 1 – Priceable.java*

|  |
| --- |
| package Price;  public interface Priceable {  int getPrice();  } |

*Листинг 2 – Guitar.java*

|  |
| --- |
| package Price;  public class Guitar implements Priceable {  private int price;  Guitar(int price) {  this.price = price;  }  public int getPrice() {  return price;  }  } |

*Листинг 3 – Home.java*

|  |
| --- |
| package Price;  public class Home implements Priceable {  private int price;  Home(int price) {  this.price = price;  }  public int getPrice() {  return price;  }  } |

*Листинг 4 – Test.java*

|  |
| --- |
| package Price;  public class Test {  public static void main(String[] args) {  Priceable printable = new Guitar(5000);  System.out.println("Gibson Les Paul стоит: " + printable.getPrice());  printable = new Home(100000);  System.out.println("Дом стоит: " + printable.getPrice());  }  } |

*Результат выполнения программ*:

****

Рисунок 9.2 – Результат задачи №2

*Выводы по работе:*

В результате выполнения изучили понятие интерфейса, научились создавать интерфейсы в Java и применять их в программах.

# **Практическая работа №10**

**Цель работы:**

Разработка и программирование рекурсивных алгоритмов на языке Java.

**Теоретическое введение:**

В контексте языка программирования рекурсия — это некий активный метод (или подпрограмма) вызываемый сам по себе непосредственно, или вызываемой другим методом (или подпрограммой) косвенно. В первую очередь надо понимать, что рекурсия — это своего рода перебор. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции.

Так же, как и у перебора (цикла) у рекурсии должно быть условие остановки — базовый случай (иначе также, как и цикл, рекурсия будет работать вечно — infinite). Это условие и является тем случаем, к которому рекурсия идет (шаг рекурсии). При каждом шаге вызывается рекурсивная функция до тех пор, пока при следующем вызове не сработает базовое условие и не произойдет остановка рекурсии (а точнее возврат к последнему вызову функции). Всё решение сводится к поиску решения для базового случая. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор, пока не получим базовое решение.

Итак, рекурсивная функция состоит из:

* условие остановки или же базового случая или условия;
* условие продолжения или шага рекурсии — способ сведения сложной задачи к более простым подзадачам.

*Задание №1:*

Разворот числа

Дано число n, десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке.

При решении этой задачи нельзя использовать циклы, строки, списки, массивы, разрешается только рекурсия и целочисленная арифметика.

Функция должна возвращать целое число, являющееся результатом работы программы, выводить число по одной цифре нельзя.

*Решение*:

Данное решение представлено на Листинге – 1.

*Листинг 1 – reverse.java*

|  |
| --- |
| package mirea.recursion.Reverse;  public class reverse  {  public static int rev(int n, int a)  {  if (n == 0) {  return a;  }  else {  return rev(n / 10, 10 \* a + n % 10);  }  }  public static void main(String[] args)  {  System.out.println(rev(123, 0));  }  } |

*Результат выполнения программ*:

****

Рисунок 10.1 – Результат задачи №1

*Задание №2:*

Вывести нечетные числа последовательности

Дана последовательность натуральных чисел (одно число в строке), завершающаяся числом 0. Выведите все нечетные числа из этой последовательности, сохраняя их порядок.

В этой задаче нельзя использовать глобальные переменные и передавать какие-либо параметры в рекурсивную функцию. Функция получает данные, считывая их с клавиатуры. Функция не возвращает значение, а сразу же выводит результат на экран. Основная программа должна состоять только из вызова этой функции.

*Решение*:

Данное решение представлено на Листинге – 2.

*Листинг 2 – odd.java*

|  |
| --- |
| package mirea.recursion.Odd;  public class odd {  public static void o() {  java.util.Scanner in = new java.util.Scanner(System.in);  int n = in.nextInt();  if (n > 0) {  if (n % 2 == 1) {  System.out.println(n);  o();  } else {  o();  }  }  }  public static void main(String[] args) {  o();  }  } |

*Результат выполнения программ*:

****

Рисунок 10.2 – Результат задачи №2

*Задание №*3*:*

Цифры числа слева направо

Дано натуральное число N. Выведите все его цифры по одной, в обычном порядке, разделяя их пробелами или новыми строками.

При решении этой задачи нельзя использовать строки, списки, массивы (ну и циклы, разумеется). Разрешена только рекурсия и целочисленная арифметика

*Решение*:

Данное решение представлено на Листинге – 3.

*Листинг 3 – left\_right.java*

|  |
| --- |
| package mirea.recursion.left\_right;  public class left\_right {  public static String recursion(int n) {  if (n < 10) {  return Integer.toString(n);  }  else {  return recursion(n / 10) + " " + n % 10;  }  }  public static void main(String[] args) {  System.out.println(recursion(179));  }  } |

*Результат выполнения программ*:

****

Рисунок 10.3 – Результат задачи №3

*Выводы по работе:*

В результате выполнения научились программировать рекурсивные алгоритмы на языке Java.

# **Практическая работа №11**

**Цель работы:**

Освоение на практике методов сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

**Теоретическое введение:**

Сортировка — это процесс упорядочивания списка элементов (организация в определенном порядке) исходного списка элементов, который возможно организован в виде контейнера или храниться в виде коллекции.

Процесс сортировки основан на упорядочивании конкретных значений, например:

* сортировка списка результатов экзаменов баллов в порядке возрастания результата;
* сортировка списка людей в алфавитном порядке по фамилии.

Есть много алгоритмов для сортировки списка элементов, которые различаются по эффективности.

*Задание №1:*

Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber и сортирует его вставками.

*Решение*:

Данное решение представлено на Листинге – 11.1 и Листинге – 11.2.

*Листинг 11.1 – Sudent.java*

|  |
| --- |
| package student;  public class Student {  private int id;  private String name;  public Student(int id, String name) {  this.id = id;  this.name = name;  }  @Override  public String toString() {  return "Student{" +  "id=" + id +  ", name=" + name +  '}';  }  public int getid() {  return id;  }  public void setid(int id) {  this.id = id;  }  public String getname() {  return name;  }  public void setname(String name) {  this.name = name;  }  public int compareTo(Student otherStudent) {  return Integer.compare(this.getid(), otherStudent.getid());  }  } |

*Листинг 11.2 – TestSudent.java*

|  |
| --- |
| package student;  public class TestStudent {  public static void main(String[] args) {  Student[] students = {  new Student(10, "Кирилл"),  new Student(19, "Дима"),  new Student(17, "Сеня"),  new Student(5, "Тёма"),  new Student(0, "Богдан")  };  for (Student s : students) {  System.out.println(s);  }  selectionSort(students);  System.out.println();  for (Student s : students) {  System.out.println(s);  }  }  public static void selectionSort (Student[] list) {  int min;  for (int index = 0; index < list.length - 1; index++) {  min = index;  Student tmp = list[index];  for (int scan = index + 1; scan < list.length; scan++) {  if (list[scan].compareTo(list[min]) < 0) {  min = scan;  }  tmp = list[min];  list[min] = list[index];  list[index] = tmp;  }  }  }  } |

*Результат выполнения программ*:

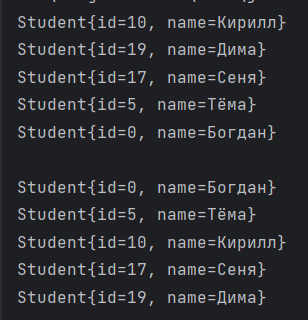
****

Рисунок 11.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения освоили на практике методы сортировки с использованием приемов программирования на объектно-ориентированном языке Java.

# **Практическая работа №12**

**Цель работы:**

Изучение на практике приемов работы со стандартными контейнерными классами Java Collection Framework.

**Теоретическое введение:**

Java Collections Framework — это набор связанных классов и интерфейсов, реализующих широко используемые структуры данных —коллекции. На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие в фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Vector появился в JDK версии Java 1.0, но, как и Hashtable, эту коллекцию не рекомендуется использовать, если не требуется достижения потокобезопасности. Потому как в Vector, в отличии от других реализаций List, все операции с данными являются синхронизированными. В качестве альтернативы часто применяется аналог — ArrayList.

Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out). Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

ArrayList — как и Vector является реализацией динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Как можно догадаться из названия, его реализация основана на обычном массиве. Данную реализацию следует применять, если в процессе работы с коллекцией предполагается частое обращение к элементам по

индексу. Из-за особенностей реализации обращение к элементам по индексу, которое выполняется за константное время O(1). Использование данной коллекции рекомендуется избегать, если требуется частое удаление/добавление элементов в середине коллекции.

LinkedList — вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка (каждый элемент списка имеет ссылки на предыдущий и следующий). Добавление и удаление элемента из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное время O(1). Ввиду реализации, данную коллекцию можно использовать как абстрактный тип данных — стек или очередь. Для этого в ней реализованы соответствующие методы.

Интерфейс Set — Представляет собой неупорядоченную коллекцию, которая не может содержать одинаковые элементы и является программной моделью математического понятия «множество».

Интерфейс Queue — Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO (first-in-first-out). Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь.

PriorityQueue — является единственной прямой реализацией интерфейса Queue (была добавлена, как и интерфейс Queue, в Java 1.5), не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. По умолчанию, элементы сортируются с использованием «natural ordering», но это поведение может быть переопределено при помощи объекта Comparator, который задаётся при создании очереди. Данная коллекция не поддерживает null в качестве элементов.

ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null.

Как заявлено в документации, эта коллекция работает быстрее чем Stack, если используется как LIFO коллекция, а также быстрее чем LinkedList, если используется как FIFO коллекция

*Задание №1:*

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9».

Карточная игра “ В пьяницу”. В этой игре карточная колода раздается поровну двум игрокам. Далее они открывают по одной верхней карте, и тот, чья карта старше, забирает себе обе открытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт, - проигрывает.

Для простоты будем считать, что все карты различны по номиналу и что самая младшая карта побеждает самую старшую карту (“шестерка берет туз”).

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Входные данные.

Программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая - 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Выходные данные.

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.

Пример ввода.

13579 24680

Пример вывода.

second 5

*Решение*:

Данное решение представлено на Листинге – 1.

*Листинг 1 – Game.java*

|  |
| --- |
| package ex1;  import java.util.Stack;  public class Game {  Stack<Integer> fp, sp;  public Game(String fp, String sp) {  this.fp = new Stack<>();  this.sp = new Stack<>();  for (int i = 4; i >= 0; i--) {  this.fp.push(Integer.parseInt(fp.substring(i, i + 1)));  this.sp.push(Integer.parseInt(sp.substring(i, i + 1)));  }  }  private void pushBack(Stack<Integer> s, Integer item) {  Stack<Integer> ns = new Stack<>();  while (!s.isEmpty()){  ns.push(s.pop());  }  s.push(item);  while (!ns.isEmpty()){  s.push(ns.pop());  }  }  public String play() {  int count = 0;  while (!fp.isEmpty() && !sp.isEmpty() && count <= 106) {  if (fp.peek() > sp.peek() && sp.peek() != 0) {  pushBack(fp, fp.pop());  pushBack(fp, sp.pop());  }  else {  pushBack(sp, fp.pop());  pushBack(sp, sp.pop());  }  count++;  }  String res = "";  if (fp.isEmpty()) { |

*Продолжение Листинга – 1*

|  |
| --- |
| res += "second ";  }  else {  res += "first ";  }  res += count;  if (count >= 106) {  res += " botva";  }  return res;  }  public static void main(String[] args) {  System.out.println(new Game("13579", "24680").play());  System.out.println(new Game("12345", "67890").play());  System.out.println(new Game("13029", "42685").play());  }  } |

*Результат выполнения программ*:

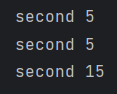
****

Рисунок 12.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения изучили на практике приемы работы со стандартными контейнерными классами Java Collection Framework

# **Практическая работа №13**

**Цель работы:**

Освоить на практике работу с файлами на языке Java. Получить практические навыки по чтению и записи данных в файл.

**Теоретическое введение:**

Класс FileWriter является производным от класса Writer. Он

используется для записи текстовых файлов.

Класс FileReader наследуется от абстрактного класса Reader и предоставляет функциональность для чтения текстовых файлов.

*Задание №1:*

1. Реализовать запись в файл введённой с клавиатуры информации
2. Реализовать вывод информации из файла на экран
3. Заменить информацию в файле на информацию, введённую с клавиатуры
4. Добавить в конец исходного файла текст, введённый с клавиатуры

*Решение*:

Данное решение представлено на Листинге – 1 и Листинге – 2.

*Листинг 1 – File.java*

|  |
| --- |
| package file;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  import java.util.Scanner;  public class File {  public void t\_1(String name) {  Scanner sc = new Scanner(System.in);  System.out.println("Введите текст: ");  String text = sc.nextLine();  try(FileWriter writer = new FileWriter(name, false)) {  writer.append(text);  writer.flush();  }  catch(IOException ex) {  System.out.println(ex.getMessage());  }  }  public void t\_2(String name) {  try(FileReader reader = new FileReader(name))  {  // читаем посимвольно  int c;  while((c=reader.read())!=-1){  System.out.print((char)c);  }  }  catch(IOException ex){ |

*Продолжение Листинга 1*

|  |
| --- |
| System.out.println(ex.getMessage());  }  }  public void t\_4(String name) {  try(FileWriter writer = new FileWriter(name, true)) {  Scanner sc = new Scanner(System.in);  System.out.println("Введите текст: ");  String text = sc.nextLine();  writer.append('\n');  writer.append(text);  writer.flush();  }  catch(IOException ex) {  System.out.println(ex.getMessage());  }  }  } |

*Листинг 2 – TestFile.java*

|  |
| --- |
| package file;  public class TestFile {  public static void main(String[] args) {  String name\_txt = "C:\\Users\\super\\IdeaProjects\\MIREA\\practice\_\_13\\src\\file\\pr13.txt";  File task = new File();  System.out.println("1: Реализовать запись в файл введённой с клавиатуры информации");  task.t\_1(name\_txt);  System.out.println("2: Реализовать вывод информации из файла на экран");  task.t\_2(name\_txt);  System.out.println('\n' + "3: Заменить информацию в файле на информацию, введённую с клавиатуры");  task.t\_1(name\_txt);  System.out.println("4: Добавить в конец исходного файла текст, введённый с клавиатуры");  task.t\_4(name\_txt);  System.out.println("Result");  task.t\_2(name\_txt);  }  } |

*Результат выполнения программ*:

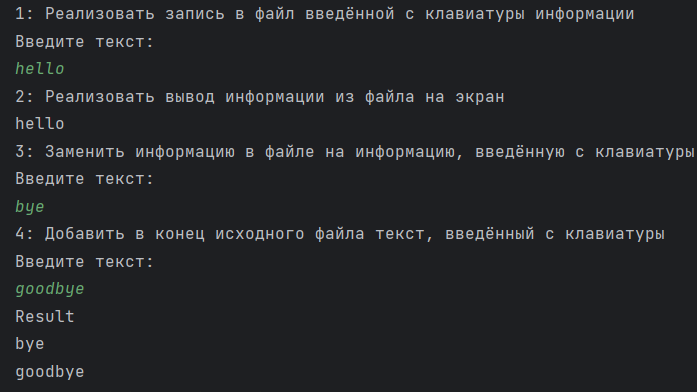
****

Рисунок 13.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения получил практические навыки по чтению и записи данных в файл.

# **Практическая работа №14**

**Цель работы:**

Изучаем различные виды списков ожидания.

**Теоретическое введение:**

Вначале, рассмотрите класс, представленный на диаграмме – общий класс список ожидания. Допустим мы решили, что нам понадобиться еще два вида списков ожидания:

• BoundedWaitList: Этот список ожидания имеет ограниченную емкость, указываемую в момент создания. Он не принимает более элементов, чем заранее задано (возможное количество потенциальных элементов в списке ожидания).

• UnfairWaitList: В этом списке ожидания, можно удалить элемент,

который не является первым в очереди - и помните он не может вернуться

обратно! (Возможны различные реализации, но в вашей реализации

необходимо удалить первое вхождение данного элемента.) Также возможно,

чтобы, например, первый элемент будет отправлен обратно в конец списка.

После описания всей задачи в целом, мы сможем решить, что мы нам

нужен интерфейс IWaitList, и затем нужно создать три разных класса для трех

списков ожидания. Также предполагается, что один из списков ожидания

должен быть супер-классом для двух других списков ожидания.

*Задание №1:*

• Исследуйте UML диаграмму классов на рисунке 1 и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.

• Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!

• Изучение работу метода main(), которая использует ваши новые классы и интерфейс.

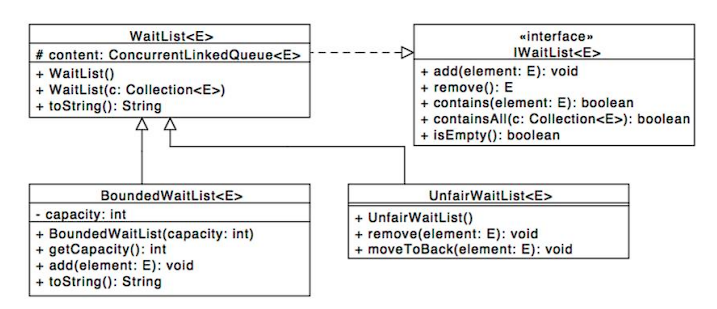


Рисунок 14.0 – UML диаграмма классов

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3, Листинге – 4 и Листинге – 5.

*Листинг 1 – IWaitList.java*

|  |
| --- |
| package wait;  import java.util.Collection;  public interface IWaitList<E> {  void add(E element);  E remove();  boolean contains(E element);  boolean containsAll(Collection<E> collection);  boolean isEmpty();  } |

*Листинг 2 – WaitList.java*

|  |
| --- |
| package wait;  import java.util.ArrayList;  import java.util.Collection;  import java.util.concurrent.ConcurrentLinkedQueue;  public class WaitList<E> implements IWaitList<E> {  protected ConcurrentLinkedQueue<E> components;  public WaitList() {  components = new ConcurrentLinkedQueue<>();  }  public WaitList(Collection<E> collection) {  components = new ConcurrentLinkedQueue<>(collection);  }  @Override  public String toString() {  return "WaitList{" +  "components=" + components +  '}';  }  @Override  public void add(E element) {  components.add(element);  }  @Override  public E remove() {  if (isEmpty()) throw new IllegalStateException("Очередь пуста!");  return components.remove();  }  @Override  public boolean contains(E element) {  boolean res = false;  for (int i = 0; i < components.size(); i++) {  E el = components.remove();  if (el.equals(element)) res = true;  components.add(el);  }  return res;  }  @Override  public boolean containsAll(Collection<E> collection) {  ArrayList<E> al = new ArrayList<>(collection);  for (int i = 0; i < collection.size(); i++) { |

*Продолжение Листинга 2*

|  |
| --- |
| boolean res = false;  for (int j = 0; j < components.size(); j++) {  E el = components.remove();  if (el.equals(al.get(i))) res = true;  components.add(el);  }  if (!res) return false;  }  return true;  }  @Override  public boolean isEmpty() {  return components.isEmpty();  }  } |

*Листинг 3 – BoundedWaitList.java*

|  |
| --- |
| package wait;  import java.util.Collection;  public class BoundedWaitList<E> extends WaitList<E> {  private int capacity;  public BoundedWaitList(int capacity) {  super();  if (capacity <= 0)  throw new IllegalArgumentException("Максимальный размер должен быть больше 0! Получено значение: " + capacity);  this.capacity = capacity;  }  public BoundedWaitList(Collection<E> col) {  super(col);  this.capacity = col.size();  }  public int getCapacity() {  return capacity;  }  @Override  public void add(E element) {  if (components.size() == capacity) throw new IllegalStateException("Очередь заполнена!");  components.add(element);  } |

*Продолжение Листинга 3*

|  |
| --- |
| @Override  public String toString() {  return "BoundedWaitList{" +  "capacity=" + capacity +  ", components=" + components +  '}';  }  } |

*Листинг 4 – UnfairWaitList.java*

|  |
| --- |
| package wait;  public class UnfairWaitList<E> extends WaitList<E> {  public UnfairWaitList() {  super();  }  public void remove(E element) {  boolean removed = false;  for (int i = 0; i < components.size(); i++) {  E el = components.remove();  if (!removed && el.equals(element)) {  removed = true;  } else {  components.add(el);  }  }  if (removed) {  components.add(components.remove());  }  }  public void moveToBack(E element) {  int prevSize = components.size();  remove(element);  if (components.size() < prevSize) {  components.add(element);  }  }  } |

*Листинг 5 – Main.java*

|  |
| --- |
| package wait;  import java.util.ArrayList;  public class Main {  public static void main(String[] args) {  ArrayList<String> al = new ArrayList<>();  al.add("Hi2");  al.add("Hi3");  WaitList<String> wl = new WaitList<>();  wl.add("Hi");  wl.add("Hi2");  wl.add("Hi3");  System.out.println(wl);  System.out.println(wl.remove());  System.out.println(wl);  System.out.println(wl.isEmpty());  System.out.println(wl.contains("Hi"));  System.out.println(wl.containsAll(al));  System.out.println();  BoundedWaitList<String> bwl = new BoundedWaitList<>(3);  bwl.add("foo");  bwl.add("foo2");  bwl.add("bar");  System.out.println(bwl);  try{  bwl.add("bar2");  }catch (Exception e){  e.printStackTrace();  }  System.out.println();  System.out.println(bwl);  System.out.println(bwl.getCapacity());  System.out.println();  bwl = new BoundedWaitList<String>(al);  System.out.println(bwl.getCapacity());  try{  bwl.add("bar2");  }catch (Exception e){  e.printStackTrace();  }  System.out.println();  UnfairWaitList<Integer> uwl = new UnfairWaitList<>(); |

*Продолжение Листинга 5*

|  |
| --- |
| uwl.add(1);  uwl.add(2);  uwl.add(23);  uwl.add(5);  uwl.add(5);  uwl.add(6);  System.out.println(uwl);  uwl.remove(5);  System.out.println(uwl);  uwl.moveToBack(23);  System.out.println(uwl);  }  } |

*Результат выполнения программ*:

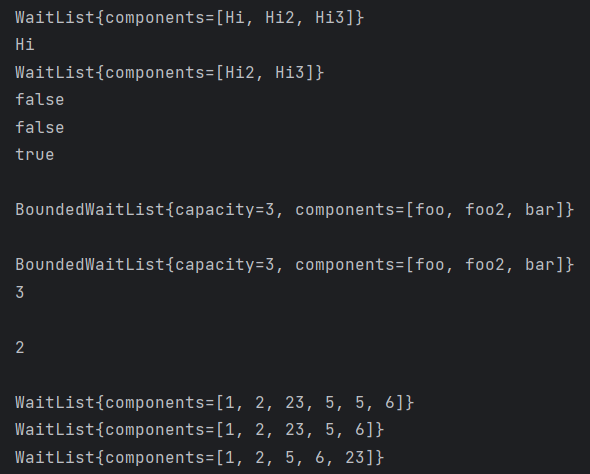
****

Рисунок 14.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения изучили различные виды списков ожидания.

# **Практическая работа №15**

**Цель работы:**

Введение в разработку программ с использованием событийного программирования на языке программирования Джава с использованием паттерна MVC.

**Теоретическое введение:**

Шаблон проектирования в программной инженерии — это метод решения часто возникающей проблемы при разработке программного обеспечения. Проектирование по модели указывает, какой тип архитектуры вы используете для решения проблемы или разработки модели.

Существует два типа моделей проектирования:

* архитектура модели 1
* архитектура модели 2 (MVC).

**Архитектура MVC в Джава приложениях**

Проекты моделей, основанные на архитектуре MVC (model-view controller), следуют шаблону проектирования MVC и отделяют логику приложения от пользовательского интерфейса при разработке программного обеспечения.

Как следует из названия, шаблон MVC имеет три уровня:

* Модель — представляет бизнес-уровень приложения (model).
* Вид — определяет представление приложения (view).
* Контроллер — управляет потоком приложения (controller).

*Задание №1:*

Напишите реализацию программного кода по UML диаграмме, представленной на рисунке 15.1. Программа должна демонстрировать использование паттерна MVC.

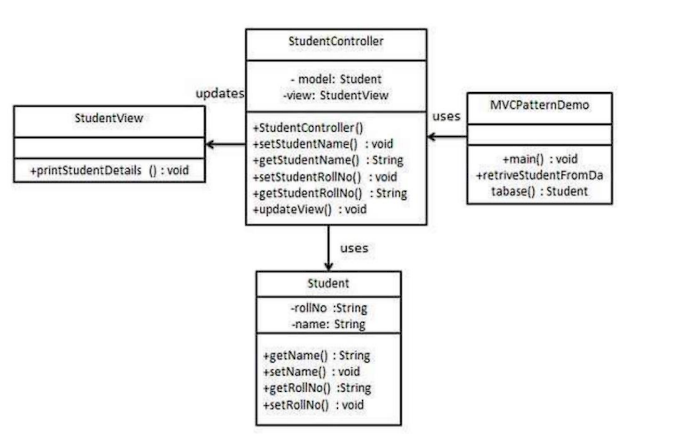


Рисунок 15.1 – UML - диаграмма классов проекта, реализующего MVC

*Решение*:

Данное решение представлено на Листинге – 1, Листинге – 2, Листинге – 3 и Листинге – 4.

*Листинг 1 – StudentController.java*

|  |
| --- |
| public class StudentController {  private Student model;  private StudentView view;  public StudentController(Student model, StudentView view) {  this.model = model;  this.view = view;  }  public void setStudentId(String id) {  model.setId(id);  }  public String getStudentId() {  return model.getId();  }  public void setStudentName(String name) { |

*Продолжение Листинга 1*

|  |
| --- |
| model.setName(name);  }  public String getStudentName() {  return model.getName();  }  public void updateView() {  view.printStudentDetails(getStudentName(), getStudentId());  }  } |

*Листинг 2 – Student.java*

|  |
| --- |
| public class Student {  private String id;  private String name;  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  } |

*Листинг 3 – StudentView.java*

|  |
| --- |
| public class StudentView {  public void printStudentDetails(String name, String id) {  System.out.println("Student: ");  System.out.println("Name: " + name);  System.out.println("ID: " + id);  }  } |

*Листинг 4 – MVCPatternDemo.java*

|  |
| --- |
| public class MVCPatternDemo {  private static Student retrieveStudentFromDB() {  Student student = new Student();  student.setName("Дин");  student.setId("001");  return student;  }  public static void main(String[] args) {  Student model = retrieveStudentFromDB();  StudentView view = new StudentView();  StudentController controller = new StudentController(model, view);  controller.updateView();  controller.setStudentName("Сэм");  controller.setStudentId("002");  controller.updateView();  }  } |

*Результат выполнения программ*:

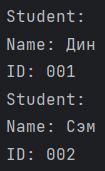
****

Рисунок 15.2 – Результат задачи №1

*Выводы по работе:*

В результате выполнения изучили событийное программирование на языке программирования Джава с использованием паттерна MVC.

# **Практическая работа №16**

**Цель работы:**

Целью данной практической работы являются получение практических навыков разработки программ, изучение синтаксиса языка Java, освоение основных конструкций языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

Ключевые слова: try, catch, finally, throw, throws.

**Теоретическое введение:**

Механизм исключительных ситуаций в Java поддерживается пятью ключевыми словами:

* try
* catch
* finally
* throw
* throws

В языке Джава Java всего около 50 ключевых слов, и пять из них связано как раз с исключениями, это- try, catch, finally, throw, throws.

Из них catch, throw и throws применяются к экземплярам класса, причём работают они только с Throwable и его наследниками. *Задание №1:*

Шаг 1. Выполните следующую программу и посмотрите, что происходит:

*Листинг 16.1 – Пример обработки деления на ноль*

|  |
| --- |
| public class Exception1 {  public void exceptionDemo() {  System.out.println( 2 / 0 );  }  } |

Описание работы

Вам необходимо инстанцировать класс и выполнить exceptionDemo(). Что произойдет?

Шаг 2. Измените программу следующим образом.

Замените 2/0 на 2,0 / 0,0 и повторно вызовите метод. Что произойдет? Теперь измените код в классе Exception1 и включите блок try-catch следующим образом:

*Листинг 16.2 – Пример обработки исключения:*

|  |
| --- |
| public class Exception1 {  public void exceptionDemo() {  try{  System.out.println( 2/0 );  } catch ( ArithmeticException e ) {  System.out.println("Attempted division by zero");  }  }  } |

Шаг 3. Запустите программу и обратите внимание на новое поведение. Объясните поведение программы.

*Решение*:

Данное решение представлено на Листинге – 16.3.

*Листинг 16.3 – Exception1.java*

|  |
| --- |
| package Exp;  public class Exception1 {  public void exceptionDemo() { |

*Продолжение Листинга 16.3*

|  |
| --- |
| System.out.println(2 / 0);  }  public void exceptionDemo2() {  System.out.println(2.0 / 0.0);  }  public void exceptionDemo3() {  try {  System.out.println(2 / 0);  } catch (ArithmeticException e) {  System.out.println("Attempted division by zero");  }  }  public static void main(String[] args) {  Exception1 q = new Exception1();  q.exceptionDemo();  //q.exceptionDemo2();  //q.exceptionDemo3();  }  } |

Результат выполнения программ:

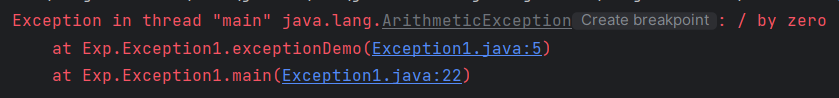


Рисунок 16.1 – Результат задачи №1, шаг 1



Рисунок 16.2 – Результат задачи №1, шаг 2



Рисунок 16.3 – Результат задачи №1, шаг 3

*Задание № 2:*

Шаг 1. Измените код в листинге 16.4 на следующий:

*Листинг 16.4 – Пример программы*

|  |
| --- |
| public class Exception2 {  public void exceptionDemo() {  Scanner myScanner = new Scanner( System.in);  System.out.print( "Enter an integer "); |

*Продолжение Листинга 16.4*

|  |
| --- |
| String intString = myScanner.next();  int i = Integer.parseInt(intString);  System.out.println( 2/i );  }  } |

Шаг 2. Запустите эту программу со следующими выводами: Qwerty 0 1.2 1. Посмотрите на вывод.

Объясните какие исключения выбрасываются?

Шаг 3. Измените код, добавив блоки try – catch, чтобы иметь дело с определяемыми исключениями.

Объясните поведение программы

*Решение*:

Данное решение представлено на Листинге – 16.5.

*Листинг 16.5 – Exception2.java*

|  |
| --- |
| package Exp;  import java.util.Scanner;  public class Exception2 {  public void exceptionDemo() {  Scanner myScanner = new Scanner(System.in);  System.out.print("Enter an integer ");  String intString = myScanner.next();  int i = Integer.parseInt(intString);  System.out.println(2 / i);  }  public void exceptionDemo1() {  Scanner myScanner = new Scanner(System.in);  System.out.print("Enter an integer ");  String intString = myScanner.next();  int i = 2;  try {  i = Integer.parseInt(intString);  System.out.println(2 / i);  } catch (ArithmeticException e) {  System.out.println("Attempted division by zero");  } catch (NumberFormatException e) {  System.out.println("The program is waiting for the input of an |

*Продолжение Листинга 16.5*

|  |
| --- |
| integer value");  }  }  public static void main(String[] args) {  Exception2 q = new Exception2();  q.exceptionDemo();  //q.exceptionDemo1();  }  } |

Результат выполнения программ:

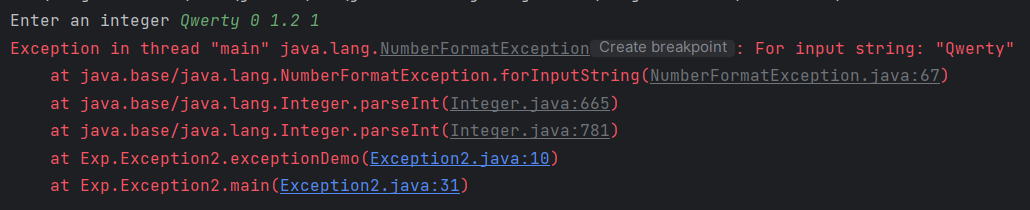


Рисунок 16.4 – Результат задачи №1, шаг 2

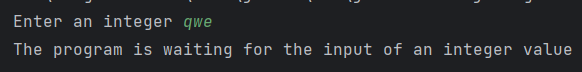


Рисунок 16.5 – Результат задачи №1, шаг 3

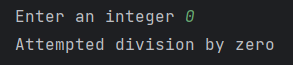


Рисунок 16.6 – Результат задачи №1, шаг 3

Задание №3:

С помощью перехватывания исключений можно оказывать влияние на поведение программы. В вашем решении в предыдущем упражнении вы можете добавить новый пункт - catch в начале списка пунктов catch.

Шаг 1. Выполите это действие, чтобы поймать общее исключение класса Exception.

Шаг 2. Перезапустите программу с приведенными выше данными и обратите внимание на ее поведение.

Объясните новое поведение программы.

*Решение*:

Данное решение представлено на Листинге – 16.6.

*Листинг 16.6 – Exception3.java*

|  |
| --- |
| package Exp;  import java.util.Scanner;  public class Exception3 {  public void exceptionDemo() {  Scanner myScanner = new Scanner(System.in);  System.out.print("Enter an integer ");  String intString = myScanner.next();  int i = Integer.parseInt(intString);  System.out.println(2 / i);  }  public void exceptionDemo1() {  Scanner myScanner = new Scanner(System.in);  System.out.print("Enter an integer ");  String intString = myScanner.next();  int i = 2;  try {  i = Integer.parseInt(intString);  System.out.println(2 / i);  } catch (NumberFormatException e) {  System.out.println("The program is waiting for the input of an integer value");  } catch (ArithmeticException e) {  System.out.println("Attempted division by zero");  } catch (Exception e1) {  System.out.println("Exception");  }  }  public static void main(String[] args) {  Exception3 q = new Exception3();  // q.exceptionDemo();  q.exceptionDemo1();  }  } |

Задание №4:

Шаг 1. Добавьте блок finally к решению Задания №2.

Шаг 2. Повторно запустите программу, чтобы проверить ее поведение.

Объясните новое поведение программы

*Решение*:

Данное решение представлено на Листинге – 16.7.

*Листинг 16.7 – Exception4.java*

|  |
| --- |
| package Exp;  import java.util.Scanner;  public class Exception4 {  public void exceptionDemo() {  Scanner myScanner = new Scanner(System.in);  System.out.print("Enter an integer ");  String intString = myScanner.next();  int i = 2;  try {  i = Integer.parseInt(intString);  System.out.println(2 / i);  } catch (ArithmeticException e) {  System.out.println("Attempted division by zero");  } catch (NumberFormatException e) {  System.out.println("The program is waiting for the input of an integer value");  } finally {  System.out.println("Этот кусок программы выведется в любом случае");  }  }  public static void main(String[] args) {  Exception4 q = new Exception4();  q.exceptionDemo();  }  } |

Результат выполнения программ:

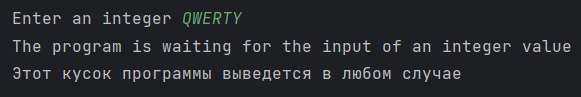


Рисунок 16.7 – Результат задачи №4, шаг 2

*Выводы по работе:*

В результате выполнения было изучено понятие исключения, а в частности блоки try, catch, finally.

# **Практическая работа №17**

**Цель работы:**

Целью данной практической работы – научиться создавать собственные исключения.

**Теоретическое введение:**

Язык Java предоставляет исчерпывающий набор классов исключений, но иногда при разработке программ вам потребуется создавать новые – свои собственные исключения, которые являются специфическими для потребностей именно вашего приложения. В этой практической работе вы научитесь создавать свои собственные пользовательские классы исключений. Как вы уже знаете, в Java есть два вида исключений- проверяемы и непроверяемые. Для начала рассмотрим создание пользовательских проверяемых исключений. *Задание №1:*

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

*Решение*:

Данное решение представлено на Листинге – 17.1 и на Листинге – 17.2.

*Листинг 17.1 – InnNotValidException.java*

|  |
| --- |
| package INN;  import java.math.BigInteger;  public class InnNotValidException extends Exception {  private BigInteger innNum;  public InnNotValidException(BigInteger inn){  super("ИНН с номером "+ inn + " не действителен!");  innNum = inn;  }  public BigInteger getInnNum() {  return innNum;  }  } |

*Листинг 17.2 – InnChecker.java*

|  |
| --- |
| package INN;  import java.math.BigInteger;  import java.util.Scanner;  public class InnChecker {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  while(true){  System.out.println("Введите номер ИНН: ");  try{  BigInteger inn = new BigInteger(sc.nextLine());  checkInn(inn);  break;  }catch (InnNotValidException e){  System.out.println(e.getLocalizedMessage());  }  }  System.out.println("ИНН действителен!"); |

*Продолжение Листинга 17.2*

|  |
| --- |
| }  public static boolean checkInn(BigInteger inn) throws InnNotValidException{  int i = 0;  BigInteger cInn = new BigInteger(inn.toByteArray());  while (!cInn.equals(new BigInteger("0"))){  i++;  cInn = new BigInteger(cInn.divide(new BigInteger("10")).toByteArray());  }  if(i != 12) throw new InnNotValidException(inn);  return true;  }  } |

Результат выполнения программ:

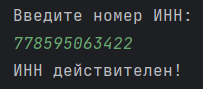


Рисунок 17.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились создавать собственные исключения.

# **Практическая работа №18**

**Цель работы:**

Целью данной практической работы – научиться работать с обобщенными типами в Java и применять прием стирание типов разработке программ на Джаве.

**Теоретическое введение:**

Понятие дженериков.

Введение в Дженерики. В JDK представлены дженерики (перевод с англ. generics), которые поддерживают абстрагирование по типам (или параметризованным типам). В объявлении классов дженерики представлены обобщенными типами, в то время как пользователи классов могут быть конкретными типами, например во время создания объекта или вызова метода.

Вы, конечно, знакомы с передачей аргументов при вызове методов в языке С++. Когда вы передаете аргументы внутри круглых скобок () в момент вызова метода, то аргументы подставляются вместо формальных параметров, с которыми объявлен и описан метод. Схожим образом в generics вместо передаваемых аргументов мы передаем информацию только о типах аргументов внутри угловых скобок <> (так называемая diamond notation или алмазная запись).

Основное назначение использования дженериков — это абстракция работы над типами при работе с коллекциями («Java Collection Framework»). *Задание №1:*

Написать обобщенный класс MinMax, который содержит методы для нахождения минимального и максимального элемента массива. Массив является переменной класса. Массив должен передаваться в класс через конструктор.

*Решение*:

Данное решение представлено на Листинге – 18.1 и на Листинге – 18.2.

*Листинг 18.1 – MinMax.java*

|  |
| --- |
| package MinMax;  import java.util.Arrays;  public class MinMax<T extends Comparable<T>> {  private T[] arr;  public MinMax(T[] arr) {  this.arr = arr;  }  public T getMinElement() {  if (arr.length > 0) {  T min = arr[0];  for (T t : arr) {  if (t.compareTo(min) < 0) {  min = t;  }  }  return min;  }  return null;  }  public T getMaxElement() {  if (arr.length > 0) {  T max = arr[0];  for (T t : arr) {  if (t.compareTo(max) > 0) {  max = t;  }  }  return max;  }  return null;  } |

*Продолжение Листинга 18.1*

|  |
| --- |
| @Override  public String toString() {  return "MinMax<" + arr.getClass().getSimpleName() + ">{" + Arrays.toString(arr) + '}';  }  } |

*Листинг 18.2 – TestMinMax.java*

|  |
| --- |
| package MinMax;  public class TestMinMax {  public static void main(String[] args) {  MinMax<Double> aDouble = new MinMax<>(new Double[]{68.54, 71.75, 19.21, 7.88});  printMinMaxInfo(aDouble);  }  public static void printMinMaxInfo(MinMax<?> arr) {  System.out.println(arr);  System.out.println("Минимальный элемент: " + arr.getMinElement());  System.out.println("Максимальный элемент: " + arr.getMaxElement());  }  } |

Результат выполнения программ:

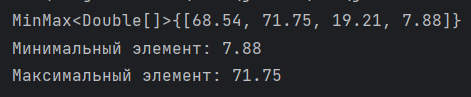


Рисунок 18.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились работать с обобщенными типами в Java и применять прием стирание типов разработке программ на Джаве.

# **Практическая работа №19**

**Цель работы:**

Целью данной практической работы – научиться работать с обобщенными типами в Java и применять прием стирание типов разработке программ на Джаве.

**Теоретическое введение:**

Из предыдущего примера может создаться видимость того, что компилятор заменяет параметризованный тип <E> актуальным или фактическим типом (таким как String, Integer) во время создания экземпляра объекта типа класс. Если это так, то компилятору необходимо будет создавать новый класс для каждого актуального или фактического типа (аналогично шаблону C++).

На самом же деле происходит следующее - компилятор заменяет всю ссылку на параметризованный тип E на ссылку на Object, выполняет проверку типа и вставляет требуемые операторы, обеспечивающие понижающее приведения типов.

Таким образом, для всех типов используется одно и то же определение класса. Самое главное, что байт-код всегда совместим с теми классами, у которых нет дженериков. Этот процесс называется стиранием типа.

*Задание №1:*

Написать метод для конвертации массива строк/чисел в список.

*Решение*:

Данное решение представлено на Листинге – 19.1.

*Листинг 19.1 – con\_arr\_str\_int.java*

|  |
| --- |
| import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class con\_arr\_str\_int {  public static <T> List<T> convertToList(T[] array) {  List<T> list = new ArrayList<>();  Collections.addAll(list, array);  return list;  }  public static void main(String[] args) {  Integer[] number = new Integer[]{1, 2, 3, 4, 5};  List<Integer> list1 = convertToList(number);  String[] string = new String[]{"w1", "w2", "w3"};  List<String> list2 = convertToList(string);  System.out.println("List1: " + list1);  System.out.println("List2: " + list2);  }  } |

Результат выполнения программ:

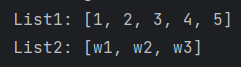


Рисунок 19.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились работать с обобщенными типами в Java и применять прием стирание типов разработке программ на Джаве.

# **Практическая работа №20**

**Цель работы:**

Целью данной практической работы – научиться разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

**Теоретическое введение:**

Стек — это линейная структура данных, которая следует принципу LIFO (Last In First Out). Стек имеет один конец, куда мы можем добавлять элементы и извлекать их оттуда, в отличие от очереди, которая имеет два конца (спереди и сзади). Стек содержит только один указатель top (верхушка стека), указывающий на самый верхний элемент стека. Всякий раз, когда элемент добавляется в стек, он добавляется на вершину стека, и этот элемент может быть удален только из стека только сверху. Другими словами, стек можно определить как контейнер, в котором вставка и удаление элементов могут выполняться с одного конца, известного как вершина стека.

Примеры стеков – пирамида, стопка тарелок или книг, магазин в пистолете

Стеку присущи следующие характеристики:

* Стек — это абстрактный тип данных с заранее определенной емкостью, что означает, что эта структура данных имеет ограниченный размер, то есть может хранить количество элементов, определенное размерностью стека.
  + Это структура данных, в которой строго определен порядок вставки и удаления элементов, и этот порядок может быть LIFO или FILO.

*Задание №1:*

Напишите программу-калькулятор арифметических выражений, записанных в обратной польской нотации (RPN-калькулятор).

*Решение*:

Данное решение представлено на Листинге – 20.1.

*Листинг 20.1 – Main.java*

|  |
| --- |
| import java.util.Scanner;  import java.util.Stack;  public class Main {  public static boolean isNum(String symb) {  try {  double res = Double.parseDouble(symb);  return true;  } catch (Exception e) {  return false;  }  }  public static boolean isOper(String symb) {  if (symb.equals("+") || symb.equals("-") || symb.equals("\*") || symb.equals("/")) {  return true;  }  return false;  }  public static double oper(Double num1, Double num2, String oper) {  switch (oper) {  case "+":  return num1 + num2;  case "-":  return num1 - num2;  case "\*":  return num1 \* num2;  case "/":  if (num2 == 0) {  throw new ArithmeticException("Деление на ноль");  }  return num1 / num2;  default:  throw new IllegalArgumentException("Неверный оператор");  } |

*Продолжение Листинга 20.1*

|  |
| --- |
| }  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  String str = in.nextLine();  String[] expression = str.split(" ");  Stack<Double> stack = new Stack<>();  for (String elem : expression) {  if (isNum(elem)) {  stack.push(Double.parseDouble(elem));  } else if (isOper(elem)) {  if (stack.size() < 2) {  throw new IllegalArgumentException("Недостаточное количество чисел");  } else {  double first = stack.pop();  double second = stack.pop();  stack.push(oper(first, second, elem));  }  }  }  System.out.println(stack.pop());  }  } |

Результат выполнения программ:



Рисунок 20.1 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

# **Практическая работа №21**

**Цель работы:**

Целью данной практической работы – научиться разрабатывать программы с абстрактными типами данных на языке Джава.

**Теоретическое введение:**

1. Очередь можно пределить как упорядоченный список, который позволяет выполнять операции вставки на одном конце, называемом REAR, и операции удаления, которые выполняются на другом конце, называемом FRONT

2. Очередь называется работает по дисциплине обслуживания «первый пришел — первый обслужен» (FCFS, first come first served)

3. Например, люди, стоящие в кассу магазина, образуют оччередь оплаты покупок.

Использование очередей в разработке программ.

Очереди широко используются

* в качестве списков ожидания для одного общего ресурса, такого как принтер, диск, ЦП;
* при асинхронной передаче данных (когда данные не передаются с одинаковой скоростью между двумя процессами), например. трубы, файловый ввод-вывод, сокеты;
* в качестве буферов в большинстве приложений, таких как медиаплеер MP3, проигрыватель компакт-дисков и т. д.;
* для ведения списка воспроизведения в медиаплеерах, чтобы добавлять и удалять песни из списка воспроизведения;
* в операционных системах для обработки прерываний и при реализации работы алгоритмов планирования и диспетчеризации.

*Задание №1:*

Реализовать очередь на массиве.

Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.

Реализуйте классы, представляющие циклическую очередь с применением массива.

Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).

Должны быть реализованы следующие функции(процедуры)/методы:

* enqueue – добавить элемент в очередь;
* element – первый элемент в очереди;
* dequeue – удалить и вернуть первый элемент в очереди;
* size – текущий размер очереди;
* isEmpty – является ли очередь пустой;
* clear – удалить все элементы из очереди.

Инвариант, пред- и постусловия записываются в исходном коде в виде

комментариев.

Обратите внимание на инкапсуляцию данных и кода во всех трех

реализациях.

Напишите тесты реализованным классам.

*Решение*:

Данное решение представлено на Листинге – 21.1.

*Листинг 21.1 – ArrayQueueModule.java*

|  |
| --- |
| package arr;  import java.util.Queue; |

*Продолжение Листинга 21.1*

|  |
| --- |
| import java.util.LinkedList;  class ArrayQueueModule {  private int a = 13;  private int b = 27;  private int c = 35;  public int getA() {  return a; }  public int getB() {  return b; }  public int getC() {  return c; }  public Queue clear(Queue current) {  current.clear();  return current; }  public Queue enqueue(Queue current, int toAdd) {  current.add(toAdd);  return current;  }  public int dequeue(Queue current) {  int toRes = (int) current.element();  current.poll();  return toRes;  }  public int size(Queue current) {  return current.size();  }  public boolean isEmpty(Queue current) {  return current.isEmpty();  }  public static void main(String[] args) {  ArrayQueueModule first = new ArrayQueueModule();  Queue<Integer> numbers = new LinkedList<>();  numbers.add(first.getA());  numbers.add(first.getB());  numbers.add(first.getC());  System.out.println("Queue: " + numbers);  int removedNumber = first.dequeue(numbers);  System.out.println("Removed Element: " + removedNumber);  System.out.println("Queue after deletion: " + numbers);  }  } |

*Листинг 21.2 – ArrayQueueADT.java*

|  |
| --- |
| package arr;  import java.util.Queue;  import java.util.LinkedList;  public class ArrayQueueADT {  Queue number = new LinkedList<>();  public Queue clear(Queue current) {  current.clear();  return current; }  public ArrayQueueADT() {  this.number.add(13);  this.number.add(64);  this.number.add(67); }  public Queue enqueue(int toAdd) {  this.number.add(toAdd);  return number; }  public Queue getNumber() {  return this.number; }  public int dequeue() {  int toRes = (int) this.number.element();  this.number.poll();  return toRes;  }  public int size() {  return this.number.size();  }  public boolean isEmpty() {  return this.number.isEmpty();  }  public void out() {  System.out.println(this.number);  }  public static void main(String[] args) {  ArrayQueueADT adt = new ArrayQueueADT();  System.out.print("Queue is ");  adt.out();  int removedNumber = adt.dequeue();  System.out.println("Removed Element: " + removedNumber);  System.out.print("Queue after deletion: ");  adt.out();  }  } |

*Листинг 21.3 – ArrayQueue.java*

|  |
| --- |
| package arr;  import java.util.ArrayList;  import java.util.LinkedList;  import java.util.Queue;  public class ArrayQueue {  Queue number = new LinkedList<>();  public Queue clear(Queue current) {  current.clear();  return current; }  public Queue enqueue(int toAdd, ArrayQueue cur) {  getNumber(cur).add(toAdd);  return number; }  public Queue add(ArrayQueue cur, int toAdd) {  cur.getNumber(cur).add(toAdd);  return cur.getNumber(cur);  }  public Queue getNumber(ArrayQueue cur) {  return cur.number;  }  public int dequeue(ArrayQueue cur) {  int toRes = (int) cur.getNumber(cur).element();  cur.getNumber(cur).poll();  return toRes;  }  public int size(ArrayQueue cur) {  return cur.getNumber(cur).size();  }  public boolean isEmpty(ArrayQueue cur) {  return cur.getNumber(cur).isEmpty();  }  public static void main(String[] args) {  ArrayQueue adt = new ArrayQueue();  adt.add(adt, 23);  adt.add(adt, 24);  adt.add(adt, 25);  System.out.println("Queue: " + adt.getNumber(adt));  int removedNumber = adt.dequeue(adt);  System.out.println("Removed Element: " + removedNumber);  System.out.println("Queue after deletion: " + adt.getNumber(adt));  }  } |

Результат выполнения программ:

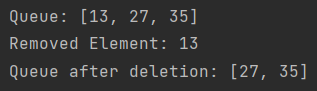


Рисунок 21.1 – Результат задачи №1 в файле ArrayQueueModule.java

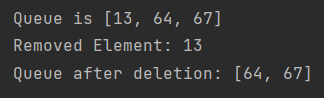


Рисунок 21.2 – Результат задачи №1 в файле ArrayQueueADT.java

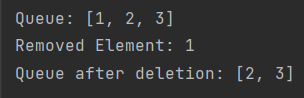


Рисунок 21.3 – Результат задачи №1 в файле ArrayQueue.java

*Выводы по работе:*

В результате выполнения научились научиться разрабатывать программы с абстрактными типами данных на языке Джава.

# **Практическая работа №22**

**Цель работы:**

Цель научиться применять порождающие паттерны при разработке программ на Java. В данной практической работе рекомендуется использовать следующие паттерны: Абстрактная фабрика и фабричный метод.

**Теоретическое введение:**

Паттерны (или шаблоны) проектирования описывают типичные способы решения часто встречающихся проблем при проектировании программ.

Некоторые из преимуществ использования шаблонов проектирования:

Шаблоны проектирования уже заранее определены и обеспечивают стандартный отраслевой подход к решению повторяющихся в программном коде проблем, вследствие этого разумное применение шаблона проектирования экономит время на разработку.

Использование шаблонов проектирования способствует реализации одного из преимуществ ООП – повторного использования кода, что приводит к более надежному и удобному в сопровождении коду. Это помогает снизить общую стоимость владения программного продукта.

Поскольку шаблоны проектирования заранее определены то это упрощает понимание и отладку нашего кода. Это приводит к более быстрому развитию проектов, так как новые члены команды понимают код.

Шаблоны проектирования Джава делятся на три категории: порождающие, структурные и поведенческие шаблоны проектирования. Так же есть еще шаблоны проектирования, например MVC – model-view-controller.

Шаблон проектирования Фабрика (factory), его также называют фабричный метод используется, когда у нас есть суперкласс с несколькими подклассами, и на основе ввода нам нужно вернуть один из подклассов. Этот

шаблон снимает с себя ответственность за создание экземпляра класса из клиентской программы в класс фабрики. Давайте сначала узнаем, как реализовать фабричный шаблон проектирования в java, а затем мы рассмотрим преимущества фабричного шаблона. Мы увидим некоторые примеры использования фабричного шаблона проектирования в JDK. Обратите внимание, что этот шаблон также известен как шаблон проектирования фабричный метод.

*Задание №1:*

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

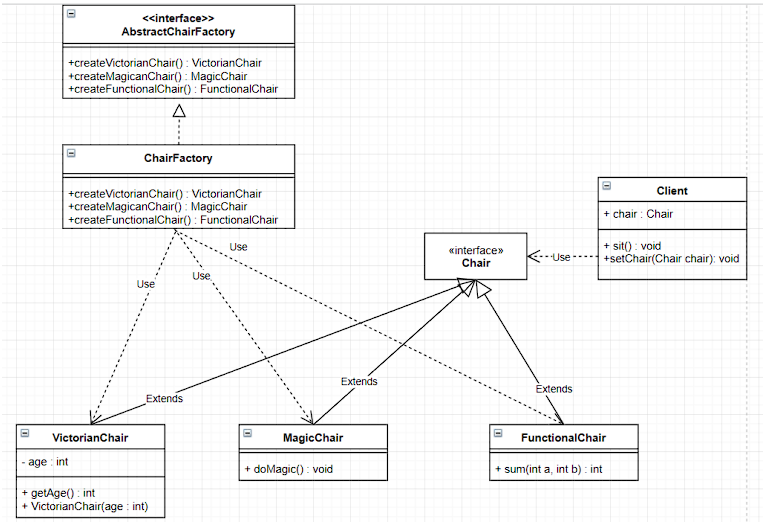


Рисунок 21.1 – UML диаграмма

*Решение*:

Данное решение представлено на Листинге – 22.1.

*Листинг 22.1 – Chair.java*

|  |
| --- |
| package factory;  public interface Chair {  void sitOn();  } |

*Листинг 22.2 – AbstarctChairFactory.java*

|  |
| --- |
| package factory;  public interface AbstractChairFactory {  VictorianChair createVictorianChair();  FunctionalChair createFunctionalChair();  MagicChair createMagicChair();  } |

*Листинг 22.3 – Client.java*

|  |
| --- |
| package factory  public class Client {  Chair chair;  public void sit(Chair chair) {  chair.sitOn();  }  public void setChair(Chair chair) {  this.chair = chair;  }  } |

*Листинг 22.4 – ChairFactory.java*

|  |
| --- |
| package factory;  public class ChairFactory implements AbstractChairFactory {  @Override  public VictorianChair createVictorianChair() {  return new VictorianChair();  }  @Override  public FunctionalChair createFunctionalChair() {  return new FunctionalChair();  }  @Override  public MagicChair createMagicChair() {  return new MagicChair();  }  } |

*Листинг 22.5 – VictorianChair.java*

|  |
| --- |
| package factory;  public class VictorianChair implements Chair {  private int age;  public int getAge() {  return age;  }  @Override  public void sitOn() {  System.out.println("Вы сидите на Викторианском стуле возрастом" + getAge());  }  } |

*Листинг 22.6 – MagicChair.java*

|  |
| --- |
| package factory;  public class MagicChair implements Chair {  public void doMagic() {  System.out.println("Магия совершена!");  }  @Override  public void sitOn() {  System.out.println("Вы сидите на Магическом стуле");  }  } |

*Листинг 22.7 – FunctionalChair.java*

|  |
| --- |
| package factory;  public class FunctionalChair implements Chair {  public int sum(int a, int b) {  return a + b;  }  @Override  public void sitOn() {  System.out.println("Вы сидите на Многофункциональном стуле");  }  } |

*Листинг 22.8 – Factory.java*

|  |
| --- |
| package factory;  public class Factory {  public static void main(String[] args){  Factory fac = new Factory();  MagicChair mag = new MagicChair();  mag.sitOn();  }  } |

Результат выполнения программ:



Рисунок 22.2 – Результат задачи №1

*Выводы по работе:*

В результате выполнения научились разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

# **Практические работы №23 и №24**

**Цель работы:**

Цель ознакомиться с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов.

*Задание №1:*

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

Рисунок 23.1 – UML диаграмма

*Решение*:

Данное решение представлено на Листинге – 23.1.

*Листинг 23.1 – Chair.java*

|  |
| --- |
| package factory;  public interface Chair {  void sitOn();  } |

# **Используемая литература**

Конспект лекций по дисциплине «Программирование на языке Джава», РТУ МИРЭА, лектор – старший преподаватель Зорина Н.В.