



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

---

---

## Институт информационных технологий

КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО  
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

---

ПРАКТИЧЕСКИЕ РАБОТЫ  
ПО ДИСЦИПЛИНЕ «Программирование на языке Джава»

Выполнил студент группы ИКБО-25-22

*Ракитин В.А.*

Принял старший преподаватель

*Рачков А.В.*

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

«Зачтено» «\_\_»\_\_\_\_2023г.

Москва 2023

## ОГЛАВЛЕНИЕ

Практическая работа № 1. Классы, как новые типы данных. Поля данных и методы.....	4
Практическая работа № 2. Циклы, условия, переменные и массивы в Java...	11
Практическая работа № 3. Использование UML диаграмм в объектно-ориентированном программировании.....	16
Практическая работа № 4. ООП в Java. Понятие класса.....	21
Практическая работа № 5. Наследование. Абстрактные суперклассы и их подклассы в Java.....	28
Практическая работа № 6. Наследование в java.....	38
Практическая работа № 7. Создание GUI. Событийное программирование в java.....	46
Практическая работа № 8. Создание программ с графическим интерфейсом пользователя на Java.....	50
Практическая работа № 9. Интерфейсы в Java.....	56
Практическая работа № 10. Программирование рекурсии в Java.....	60
Практическая работа № 11. Техники сортировки в Java.....	66
Практическая работа № 12. Использование стандартных контейнерных классов при программировании на Java.....	72
Практическая работа № 13. Работа с файлами.....	76
Практическая работа № 14. Различные виды списков ожидания.....	81
Практическая работа № 15. Разработка интерактивных программ на языке Java с использованием паттерна MVC.....	88
Практическая работа № 16. Исключения и работа с ними в Java.....	99
Практическая работа № 17. Создание пользовательских исключений.....	111
Практическая работа № 18. Работа с дженериками.....	120
Практическая работа № 19. Стирание типов в Джава.....	126
Практическая работа № 20. Абстрактные типы данных. Стек.....	130
Практическая работа № 21. Абстрактные типы данных. Очередь.....	132

Практическая работа № 22. Паттерны проектирования, порождающие паттерны: абстрактная фабрика, фабричный метод.....	138
Практическая работа № 23. Реализация функционала ресторана.....	150
Практическая работа № 24. Реализация функционала ресторана.	
Продолжение.....	158

## **Практическая работа №1**

### **Цель работы**

Введение в разработку программ на языке программирования Джава.

### **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.

Чтобы объявить переменную, необходимо указать тип переменной и ее имя. Тип переменной может быть разным: целочисленный (long, int, short, byte), число с плавающей запятой (double, float), логический (boolean), перечисление, объектный (Object).

Массив — это конечная последовательность элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Цикл — это конструкция, позволяющая выполнять определенную часть кода несколько раз. В Джава есть три типа циклов for, while, do while.

Для ввода данных используется класс Scanner из пакета java.util. Этот класс надо импортировать в той программе, где он будет использоваться. Это делается до начала определения класса в коде программы.

Методы позволяют выполнять блок кода, из любого другого места, где это доступно. Методы определяются внутри классов. Методы могут быть статическими (можно выполнять без создания экземпляра класса), не статическими (не могут выполняться без создания экземпляра класса). Методы могут быть открытыми (public), закрытыми (private). Закрытые методы могут вызываться только внутри того класса, в котором они определены. Открытые методы можно вызывать для объекта внутри других классов.

### **Выполнение лабораторной работы**

#### **Задание:**

1. Создать проект в IntelliJ IDEA

2. Написать программу, в результате которой массив чисел создается с помощью инициализации (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.
3. 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.
4. Написать программу, в результате которой выводятся на экран аргументы командной строки в цикле for.
5. Написать программу, в результате работы которой выводятся на экран первые 10 чисел гармонического ряда (форматировать вывод).
6. Написать программу, которая с помощью метода класса, вычисляет факториал числа (использовать управляющую конструкцию цикла), проверить работу метода.

Решение:

1. Создание проекта в IntelliJ IDEA (рисунок 1.1).

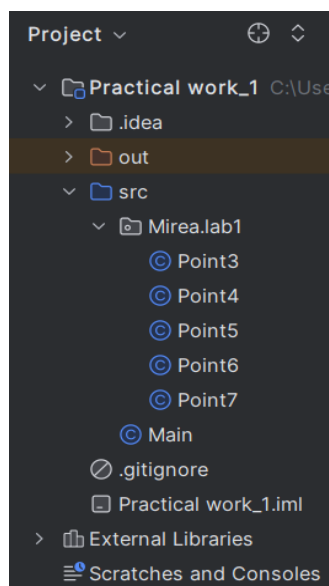


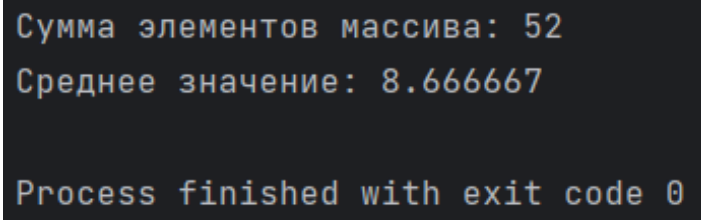
Рисунок 1.1 – Скриншот структуры созданного проекта в среде IntelliJ IDEA

2. Программа, реализующая инициализацию целочисленного массива с вычислением суммы и среднего арифметического его элементов при помощи цикла for (листинг 1.1) и результат тестирования (Рисунок 1.2).

#### Листинг 1.1 – Код программы

```
package Mirea.lab1;

public class Point3 {
    public static void main(String[] args) {
        int[] arr = {1, 3, 3, 38, 0, 7};
        int sum = 0;
        for(int i = 0; i < arr.length; i++){
            sum += arr[i];
        }
        float average = (float) sum/ arr.length;
        System.out.println("Сумма элементов массива: "+ sum);
        System.out.println("Среднее значение: "+ average);
    }
}
```



```
Сумма элементов массива: 52
Среднее значение: 8.666667

Process finished with exit code 0
```

Рисунок 1.2 – Результат работы программы

3. Программа, реализующая ввод массива с клавиатуры с последующим вычислением суммы элементов, нахождением минимального и максимального значений среди элементов массива при помощи цикла do while, while (листинг 1.2) и результат тестирования (рисунок 1.3).

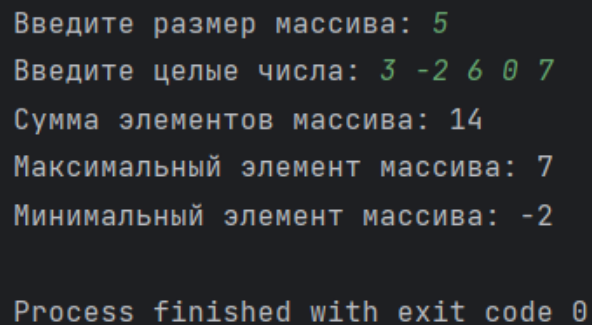
#### Листинг 1.2 – Код программы

```
package Mirea.lab1;
import java.util.Scanner;

public class Point4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        if(!scanner.hasNextInt()) {
            System.out.print("Некорректный ввод размера массива. Введите число от 1 до 2147483647");
            return;
        }
        int size = scanner.nextInt();
        if(size <= 0){
            System.out.print("Некорректный ввод размера массива. Введите число от 1 до 2147483647");
            return;
        }
    }
}
```

## Продолжение листинга 1.2

```
int[] arr = new int[size];
System.out.print("Введите целые числа: ");
int i = 0;
if(!scanner.hasNextInt()){
    System.out.print("Некорректный ввод");
    return;
}
arr[i] = scanner.nextInt();
long sum = arr[i];
int max = arr[i];
int min = arr[i];
i++;
while (i < size){
    if(!scanner.hasNextInt()){
        System.out.print("Некорректный ввод");
        return;
    }
    arr[i] = scanner.nextInt();
    sum += arr[i];
    if (arr[i] > max){
        max = arr[i];
    }
    if (arr[i] < min){
        min = arr[i];
    }
    i++;
}
System.out.println("Сумма элементов массива: " + sum);
System.out.println("Максимальный элемент массива: " + max);
System.out.println("Минимальный элемент массива: " + min);
}
```



```
Введите размер массива: 5
Введите целые числа: 3 -2 6 0 7
Сумма элементов массива: 14
Максимальный элемент массива: 7
Минимальный элемент массива: -2

Process finished with exit code 0
```

Рисунок 1.3 – Результат работы программы

4. Программа, реализующая вывод аргументов командной строки в цикле for (листинг 1.3) и результат тестирования (рисунки 1.4-1.5).

## Листинг 1.3 – Код программы

```
package Mirea.lab1;
public class Point5 {
    public static void main(String[] args) {
        for(int i = 0; i < args.length; i++){
            System.out.println(args[i]);
        }
    }
}
```

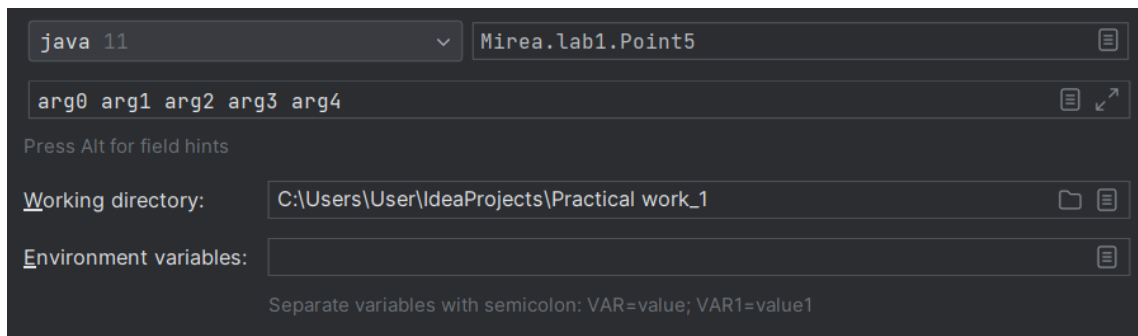


Рисунок 1.4 – Ввод аргументов

```
arg0
arg1
arg2
arg3
arg4

Process finished with exit code 0
```

Рисунок 1.5 – Результат работы программы

5. Программа, реализующая вывод 10-ти первых чисел гармонического ряда с форматированным выводом (листинг 1.4) и результат тестирования (рисунок 1.6).

Листинг 1.4 – Код программы

```
package Mirea.lab1;

public class Point6 {
    public static void main(String[] args) {
        System.out.println("Первые 10 чисел гармонического ряда:");
        for(int i = 1; i <= 10; i++){
            System.out.printf("%f\n", 1.0/i);
        }
    }
}
```

```
Первые 10 чисел гармонического ряда:
1,000000
0,500000
0,333333
0,250000
0,200000
0,166667
0,142857
0,125000
0,111111
0,100000

Process finished with exit code 0
```

Рисунок 1.6 – Результат работы программы



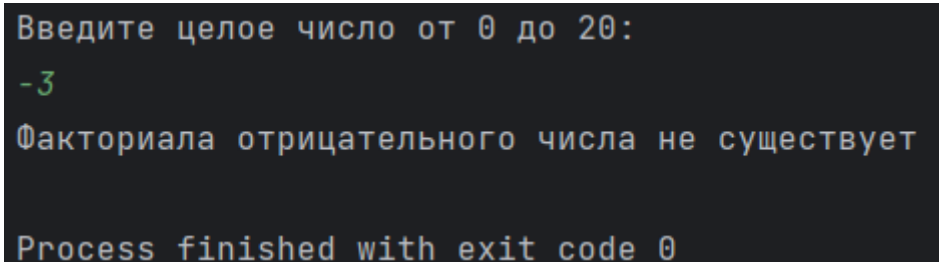
6. Программа, реализующая вычисление факториала числа при помощи метода класса и конструкции цикла (листинг 1.5). Результаты тестирования приведены на рисунках 1.7 – 1.9.

#### Листинг 1.5 – Код программы

```
package Mirea.lab1;

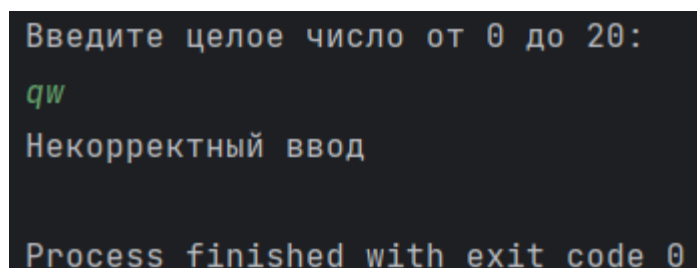
import java.util.Scanner;

public class Point7 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите целое число от 0 до 20: ");
        if(!scanner.hasNextInt()){
            System.out.println("Некорректный ввод");
            return;
        }
        int num = scanner.nextInt();
        if(num < 0){
            System.out.println("Факториала отрицательного числа не существует");
            return;
        }
        else if (num > 20){
            System.out.println("Введите число от 0 до 20");
            return;
        }
        long factorial = 1;
        for(int i = 1; i <= num; i++){
            factorial *= i;
        }
        System.out.println(num + "! = " + factorial);
    }
}
```



```
Введите целое число от 0 до 20:
-3
Факториала отрицательного числа не существует
Process finished with exit code 0
```

Рисунок 1.7 – Реакция программы на ввод отрицательного числа



```
Введите целое число от 0 до 20:
qw
Некорректный ввод
Process finished with exit code 0
```

Рисунок 1.8 – Реакция программы на некорректный ввод

```
Введите целое число от 0 до 20:  
6  
6! = 720  
  
Process finished with exit code 0
```

Рисунок 1.9 – Результат работы программы

### **Выводы по работе:**

Во время выполнения работы были получены навыки работы с различными типами переменных, массивами, циклическими и условными конструкциями, методами языка Java.

## Практическая работа №2

### Цель работы

Освоить на практике работу с классами языка программирования Java.

### Теоретическое введение

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. Графически можно представить класс в виде UML диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса. Пример общего представления диаграммы класса представлен на рисунке 2.1.

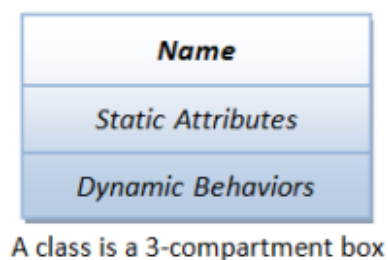


Рисунок 2.1 – Диаграмма класса. Общее представление

Рассмотрим подробнее диаграмму класса:

Имя (или сущность) – определяет класс.

Переменные (или атрибуты, состояние, поля данных класса) – содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными) – описывают динамическое поведение класса.

### Выполнение лабораторной работы

#### Задание:

Необходимо реализовать простейший класс на языке программирования Java. Не забудьте добавить метод `toString()` к вашему классу. Также в программе необходимо предусмотреть класс-тестер для тестирования класса и вывода информации об объекте.

1. Реализуйте простейший класс «Собака».
2. Реализуйте простейший класс «Мяч».
3. Реализуйте простейший класс «Книга».

Решение:

1. Программы, реализующие работу класса «Собака» (листинг 2.1) и работу тестового класса «Собака» (листинг 2.2). Результат работы программы представлен на рисунке 2.2.

Листинг 2.1 – Код программы класса «Собака»

```
public class Dog {
    private String name;
    private int age;
    public Dog(String n, int a){
        name = n;
        age = a;
    }
    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() {
        return name;
    }
    public String toString() {
        return this.name+", age "+this.age;
    }
    public void intoHumanAge(){
        System.out.println(name+"'s age in human years is "+age*7+" years");
    }
}
```

Листинг 2.2 – Код тестового класса «Собака»

```
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(); d3.setAge(1);
        System.out.println(d1);
        d1.intoHumanAge();
        d2.intoHumanAge();
        d3.intoHumanAge();
        d2.setName("Ivan");
        System.out.println(d2);
        System.out.println(d1.getName());
    }
}
```

```

Mike, age 2
Mike's age in human years is 14 years
Helen's age in human years is 49 years
Pup's age in human years is 7 years
Ivan, age 7
Mike

Process finished with exit code 0

```

Рисунок 2.2 – Результат тестирования класса «Собака»

2. Программы, реализующие работу класса «Мяч» (листинг 2.3) и работу тестового класса «Мяч» (листинг 2.4). Результат работы программы представлен на рисунке 2.3.

Листинг 2.3 – Код программы класса «Мяч»

```

public class Ball {
    private String type; //Для чего этот мяч? Для баскетбола, футбола,
    волейбола и т.д.
    private String color; //Цвет мяча
    public Ball(String t, String c){ //Параметризированный конструктор
        type = t;
        color = c;
    }
    public Ball(){ //Конструктор
        type = "basketball";
        color = "orange";
    }
    public void setType(String type){ //Установка типа мяча
        this.type = type;
    }
    public void setColor(String color){ //Установка цвета мяча
        this.color = color;
    }
    public String getColor() {
        return color;
    }
    public String getType() {
        return type;
    }
    public void infoBall(){ //Метод, выводящий информацию о мяче
        System.out.println("Ball for "+type+", this color - "+color);
    }
}

```

Листинг 2.4 – Код программы тестового класса «Мяч»

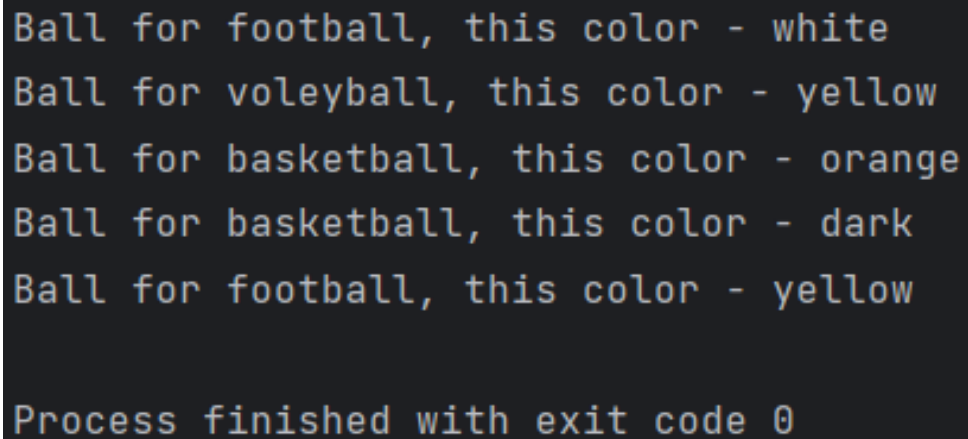
```

public class TestBall {
    public static void main(String[] args) {
        Ball b1 = new Ball("football", "white");
        Ball b2 = new Ball("volleyball", "yellow");
        Ball b3 = new Ball();
        b1.infoBall();
        b2.infoBall();
        b3.infoBall();
    }
}

```

## Продолжение листинга 2.4

```
b3.setColor("dark");  
b3.infoBall();  
b2.setType("football");  
b2.infoBall();  
}  
}
```



```
Ball for football, this color - white  
Ball for volleyball, this color - yellow  
Ball for basketball, this color - orange  
Ball for basketball, this color - dark  
Ball for football, this color - yellow  
  
Process finished with exit code 0
```

Рисунок 2.3 – Результат тестирования класса «Мяч»

3. Программы, реализующие работу класса «Книга» (листинг 2.5) и работу тестового класса «Книга» (листинг 2.6). Результат работы программы представлен на рисунке 2.4.

## Листинг 2.5 – Код программы класса «Книга»

```
public class Book {  
    private String name; //Название книги  
    private String author; //Автор книги  
    private int page; //Количество страниц в книге  
  
    public Book(String n, String a, int c){  
        name = n;  
        author = a;  
        page = c;  
    }  
    public Book(){  
        name = "War and Peace";  
        author = "Tolstoy";  
        page = 2319;  
    }  
    public void setName(String name){  
        this.name = name;  
    }  
    public void setAuthor(String author){  
        this.author = author;  
    }  
    public void setPage(int page){  
        this.page = page;  
    }  
    public int getPage() {  
        return page;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

## Продолжение листинга 2.5

```
    }  
    public String getAuthor() {  
        return author;  
    }  
    public void infoBook(){ //Информация о книге  
        System.out.println("Book name - "+name+", author this book -  
"+author+", number of pages - "+page);  
    }  
    public void timeReadBook(){ //За сколько минут можно прочитать книгу? 1  
        //страница читается 3 минуты  
        System.out.println("time read book <"+name+"> - "+page*3+"  
minutes");  
    }  
}
```

## Листинг 2.6 – Код программы тестового класса «Книга»

```
public class TestBook {  
    public static void main(String[] args) {  
        Book b1 = new Book("Gore ot yma", "Griboedov", 341);  
        Book b2 = new Book("Fathers and Sons", "Tyrgenew", 442);  
        Book b3 = new Book();  
        b1.infoBook();  
        b2.infoBook();  
        b3.infoBook();  
        b1.setName("Captain's daughter"); b1.setAuthor("Pyshkin");  
        b1.setPage(178);  
        b1.infoBook();  
        b3.timeReadBook();  
    }  
}
```

```
Book name - Gore ot yma, author this book - Griboedov, number of pages - 341  
Book name - Fathers and Sons, author this book - Tyrgenew, number of pages - 442  
Book name - War and Peace, author this book - Tolstoy, number of pages - 2319  
Book name - Captain's daughter, author this book - Pyshkin, number of pages - 178  
time read book <War and Peace> - 6957 minutes  
  
Process finished with exit code 0
```

Рисунок 2.4 – Результат тестирования класса «Книга»

### Выводы по работе:

Во время выполнения работы были получены навыки работы с классами, различными полями и методами класса.

## 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.

Графически представляем класс в виде прямоугольника, разделенного на три области – область именования класса, область инкапсуляции данных и область операций (методы). На рисунке 3.1 приведен общий вид UML диаграммы класса.

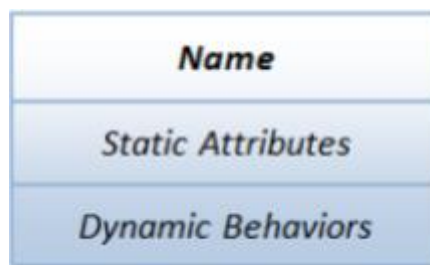


Рисунок 3.1 – Представление класса

Имя (или сущность) – определяет класс.

Переменные (или атрибуты, состояние, поля данных класса) – содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными) – описывают динамическое поведение класса.

### Выполнение лабораторной 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 3.2.



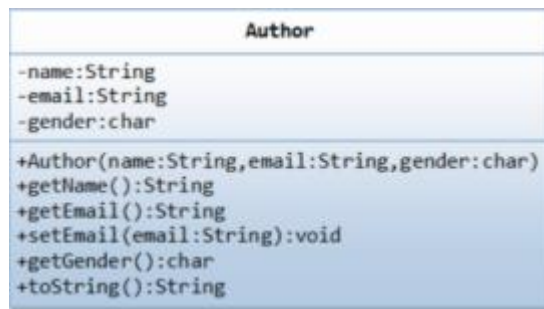


Рисунок 3.2 – Диаграмма класса Author

2. По UML диаграмме класса, представленной на рисунке 3.3. 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.

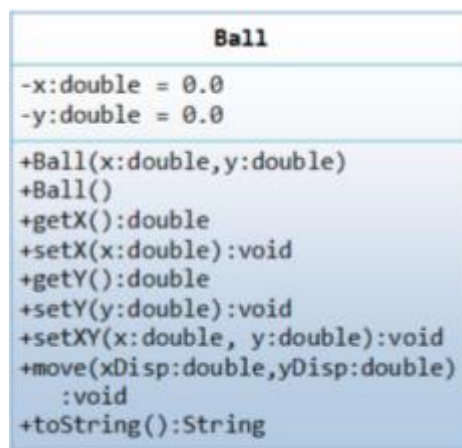


Рисунок 3.3 – Диаграмма класса Ball

### Решение:

1. Программы, реализующие работу класса Author (листинг 3.1) и тестового класса Author (листинг 3.2). Результат работы программы представлен на рисунке 3.4.

### Листинг 3.1 – Код программы класса Author

```

public class Author {
    private String name;
    private String email;
    private char gender; // 'M', 'F', 'U'

    public Author(String n, String e, char g){
        name = n;
        email = e;
        gender = g;
    }
    public String getName() {
  
```

### Продолжение листинга 3.1

```
        return name;
    }
    public String getEmail() {
        return email;
    }
    public char getGender() {
        return gender;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    public String toString() {
        return (name+" (" +gender+" ) at "+email);
    }
}
```

### Листинг 3.2 – Код программы тестового класса Author

```
public class TestAuthor {
    public static void main(String[] args) {
        Author d1 = new Author("Ivan Popov", "ivPopov@somewhere.com", 'M');
        System.out.println(d1.toString());
        d1.setEmail("IVP@somewhere.com");
        System.out.println(d1.toString());
        System.out.println(d1.getEmail());
        System.out.println(d1.getName());
        Author d2 = new Author("Anna Ivanova", "anIvanova@somewhere.com",
        'F');
        System.out.println(d2.getEmail());
        System.out.println(d2.getGender());
        System.out.println(d2.toString());
    }
}
```



```
Ivan Popov (M) at ivPopov@somewhere.com
Ivan Popov (M) at IVP@somewhere.com
IVP@somewhere.com
Ivan Popov
anIvanova@somewhere.com
F
Anna Ivanova (F) at anIvanova@somewhere.com

Process finished with exit code 0
```

Рисунок 3.4 – Результат тестирования класса Author

2. Программы, реализующие работу класса Ball (листинг 3.3) и тестового класса Ball (листинг 3.4). Результат работы программы представлен на рисунке 3.5.

### Листинг 3.3 – Код программы класса Ball

```
public class Ball {
    private double x = 0.0;
    private double y = 0.0;
    public Ball(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public Ball() {}
    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) {
        x += xDisp;
        y += yDisp;
    }
    public String toString() {
        return "Ball @ (" + this.x + ", " + this.y + ").";
    }
}
```

### Листинг 3.4 – Код программы тестового класса Ball

```
public class TestBall {
    public static void main(String[] args) {
        Ball b1 = new Ball(30.0, 30.0);
        System.out.println(b1.toString());
        b1.setX(38.0);
        b1.setY(45.0);
        System.out.println(b1);
        b1.setXY(44.0, 50.0);
        System.out.println(b1);
        b1.move(12.0, 5.0);
        System.out.println(b1);
        Ball b2 = new Ball();
        b2.setXY(15.0, 15.0);
        System.out.println(b2);
        System.out.println(b1.getX());
        System.out.println(b2.getY());
    }
}
```

```
Ball @ (30.0, 30.0).  
Ball @ (38.0, 45.0).  
Ball @ (44.0, 50.0).  
Ball @ (56.0, 55.0).  
Ball @ (15.0, 15.0).  
56.0  
15.0  
  
Process finished with exit code 0
```

Рисунок 3.5 – Результат тестирования класса Ball

**Выводы по работе:**

Во время выполнения работы были получены навыки написания программ по UML-диаграммам.

## **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 экземпляр класса. Данные объекта скрыты от остальной программы. Скрытие данных называется инкапсуляцией. Полиморфизм в ООП — возможность единообразно обрабатывать объекты с различной реализацией при условии наличия общего интерфейса.

Класс в ООП — это в чистом виде абстрактный тип данных, создаваемый программистом. Для того чтобы создать класс в языке Java, необходимо создать файл с расширением java. Имя файла должно быть таким же, как и имя создаваемого класса. В созданном файле должен описываться класс. В качестве модификатора доступа можно указать ключевое слово `public` или `private`. Если указано слово `public`, то класс будет доступен из других пакетов. Если указано слово `private`, то класс будет доступен только внутри того пакета, в котором он находится.

В теле класса можно описать методы, переменные, константы, конструкторы класса. Конструктор – это специальный метод, который вызывается при создании нового объекта. Конструктор инициализирует объект непосредственно во время создания.

Для того чтобы создать экземпляр класса необходимо объявить переменную, тип которой соответствует имени класса или суперкласса. После чего нужно присвоить этой переменной значение, вызвав конструктор создаваемого класса с помощью оператора `new`. После этого можно вызывать методы этого класса для объекта, указав имя метода через точку.

### **Выполнение лабораторной 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).

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.

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:

1. Программы, реализующие работу класса Circle (листинг 4.1) и работу тестового класса CircleTest (листинг 4.2). Результат работы программы представлен на рисунке 4.1.

Листинг 4.1 – Код программы класса Circle

```
public class Circle {
    private double radius;

    public Circle(double radius){
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

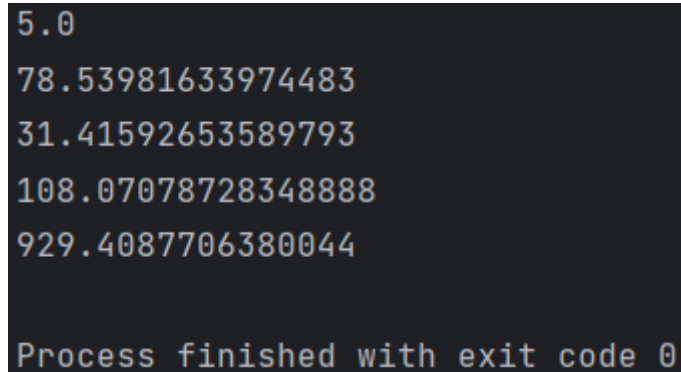
    public void setRadius(double radius) {
        this.radius = radius;
    }
    public double getAtea(){
        return Math.PI * radius * radius;
    }
    public double getPerimeter(){
        return 2 * Math.PI * radius;
    }
}
```

Листинг 4.2 – Код программы тестового класса CircleTest

```
public class CircleTest {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        System.out.println(circle.getRadius());
    }
}
```

## Продолжение листинга 4.2

```
        System.out.println(circle.getAtea());  
        System.out.println(circle.getPerimeter());  
        circle.setRadius(17.2);  
        System.out.println(circle.getPerimeter());  
        System.out.println(circle.getAtea());  
    }  
}
```



```
5.0  
78.53981633974483  
31.41592653589793  
108.07078728348888  
929.4087706380044  
  
Process finished with exit code 0
```

Рисунок 4.1 – Результат тестирования класса Circle

2. Программы, реализующие работу класса Human (листинг 4.3), класса Head (листинг 4.4), класса Leg (листинг 4.5), класса Hand (листинг 4.6) и тестового класса HumanTest (листинг 4.7). Результат работы программы представлен на рисунке 4.2.

## Листинг 4.3 – Код программы класса Human

```
public class Human {  
    private Head head;  
    private Leg leg;  
    private Hand hand;  
    public Human() {  
        this.head = new Head();  
        this.hand = new Hand();  
        this.leg = new Leg();  
    }  
    public Human(Head head, Hand hand, Leg leg) {  
        this.head = head;  
        this.hand = hand;  
        this.leg = leg;  
    }  
    public Hand getHand() {  
        return hand;  
    }  
    public Head getHead() {  
        return head;  
    }  
    public Leg getLeg() {  
        return leg;  
    }  
    public void setHand(Hand hand) {  
        this.hand = hand;  
    }  
    public void setLeg(Leg leg) {  
        this.leg = leg;  
    }  
}
```

### Продолжение листинга 4.3

```
}  
    public void setHead(Head head) {  
        this.head = head;  
    }  
}
```

### Листинг 4.4 – Код программы класса Head

```
public class Head {  
    private String hair;  
    private String eyes;  
    public Head() {  
        this.eyes = "blue";  
        this.hair = "black";  
    }  
    public Head(String hair, String eyes) {  
        this.hair = hair;  
        this.eyes = eyes;  
    }  
    public String getEyes() {  
        return eyes;  
    }  
    public String getHair() {  
        return hair;  
    }  
    public void setEyes(String eyes) {  
        this.eyes = eyes;  
    }  
    public void setHair(String hair) {  
        this.hair = hair;  
    }  
}
```

### Листинг 4.5 – Код программы класса Leg

```
public class Leg {  
    private int size;  
    private String colorBoots;  
    public Leg() {  
        this.size = 43;  
        this.colorBoots = "green";  
    }  
    public Leg(int size, String colorBoots) {  
        this.size = size;  
        this.colorBoots = colorBoots;  
    }  
    public int getSize() {  
        return size;  
    }  
  
    public String getColorBoots() {  
        return colorBoots;  
    }  
  
    public void setSize(int size) {  
        this.size = size;  
    }  
  
    public void setColorBoots(String colorBoots) {  
        this.colorBoots = colorBoots;  
    }  
}
```



#### Листинг 4.6 – Код программы класса Hand

```
public class Hand {
    private String colorBracelet;
    private String colorNails;
    public Hand() {
        this.colorBracelet = "red";
        this.colorNails = "black";
    }
    public Hand(String colorBracelet, String colorNails) {
        this.colorBracelet = colorBracelet;
        this.colorNails = colorNails;
    }
    public String getColorBracelet() {
        return colorBracelet;
    }
    public String getColorNails() {
        return colorNails;
    }
    public void setColorBracelet(String colorBracelet) {
        this.colorBracelet = colorBracelet;
    }
    public void setColorNails(String colorNails) {
        this.colorNails = colorNails;
    }
}
```

#### Листинг 4.7 – Код программы тестового класса HumanTest

```
public class HumanTest {
    public static void main(String[] args) {
        Head head = new Head("blonde", "green");
        Leg leg = new Leg(39, "brown");
        Hand hand = new Hand("silver", "pink");
        Human human = new Human(head, hand, leg);

        System.out.println("Hair color: " + human.getHead().getHair());
        System.out.println("Eye color: " + human.getHead().getEyes());
        System.out.println("Leg size: " + human.getLeg().getSize());
        System.out.println("Boot color: " + human.getLeg().getColorBoots());
        System.out.println("Bracelet color: " +
human.getHand().getColorBracelet());
        System.out.println("Nail color: " +
human.getHand().getColorNails());

        Head newHead = new Head();
        Leg newLeg = new Leg();
        Hand newHand = new Hand();

        human.setHead(newHead);
        human.setLeg(newLeg);
        human.setHand(newHand);

        System.out.println("Hair color: " + human.getHead().getHair());
        System.out.println("Eye color: " + human.getHead().getEyes());
        System.out.println("Leg size: " + human.getLeg().getSize());
        System.out.println("Boot color: " + human.getLeg().getColorBoots());
        System.out.println("Bracelet color: " +
human.getHand().getColorBracelet());
        System.out.println("Nail color: " +
human.getHand().getColorNails());

    }
}
```

```
Hair color: blonde
Eye color: green
Leg size: 39
Boot color: brown
Bracelet color: silver
Nail color: pink
Hair color: black
Eye color: blue
Leg size: 43
Boot color: green
Bracelet color: red
Nail color: black

Process finished with exit code 0
```

Рисунок 4.2 – Результат работы класса HumanTest

3. Программы, реализующие работу класса Book (листинг 4.8) и работу тестового класса BookTest (листинг 4.9). Результат работы программы представлен на рисунке 4.3.

Листинг 4.8 – Код программы класса Book

```
public class Book {
    private String author;
    private String title;
    private int year;
    public Book(){
        this.author = "Толстой";
        this.title = "Война и мир";
        this.year = 1863;
    }
    public Book(String author, String title, int year){
        this.author = author;
        this.title = title;
        this.year = year;
    }
    public void setAuthor(String author){
        this.author = author;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public void setYear(int year) {
        this.year = year;
    }
    public String getAuthor() {
        return author;
    }
    public String getTitle() {
        return title;
    }
}
```

#### Продолжение листинга 4.8

```
}  
    public int getYear() {  
        return year;  
    }  
    public String toString(){  
        return ("Название книги - "+this.title+", автор книги -  
"+this.author+", год выпуска - "+this.year);  
    }  
}
```

#### Листинг 4.9 – Код программы тестового класса BookTest

```
public class BookTest {  
    public static void main(String[] args) {  
        Book b1 = new Book();  
        System.out.println(b1.toString());  
        b1.setAuthor("Тургенев");  
        b1.setTitle("Отцы и дети");  
        System.out.println(b1.getAuthor());  
        System.out.println(b1.getTitle());  
        Book b2 = new Book("Пушкин", "Капитанская дочка", 1809);  
        System.out.println(b2.getYear());  
        b2.setYear(1999);  
        System.out.println(b2.toString());  
    }  
}
```

```
Название книги - Война и мир, автор книги - Толстой, год выпуска - 1863  
Тургенев  
Отцы и дети  
1809  
Название книги - Капитанская дочка, автор книги - Пушкин, год выпуска - 1999  
Process finished with exit code 0
```

Рисунок 4.3 – Результат работы класса BookTest

#### Выводы по работе:

Во время выполнения работы были изучены концепции объектно-ориентированного программирования, было изучено понятие и создание классов.

## **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.

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.

### **Выполнение лабораторной 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 5.1.



Рисунок 5.1 – Диаграмма суперкласса Shape

2. Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. Объясните полученные ошибки, если таковые имеются.

3. Напишите два класса **MovablePoint** и **MovableCircle** - которые реализуют интерфейс **Movable**. Диаграмма реализации интерфейса **Movable** представлена на рисунке 5.2.

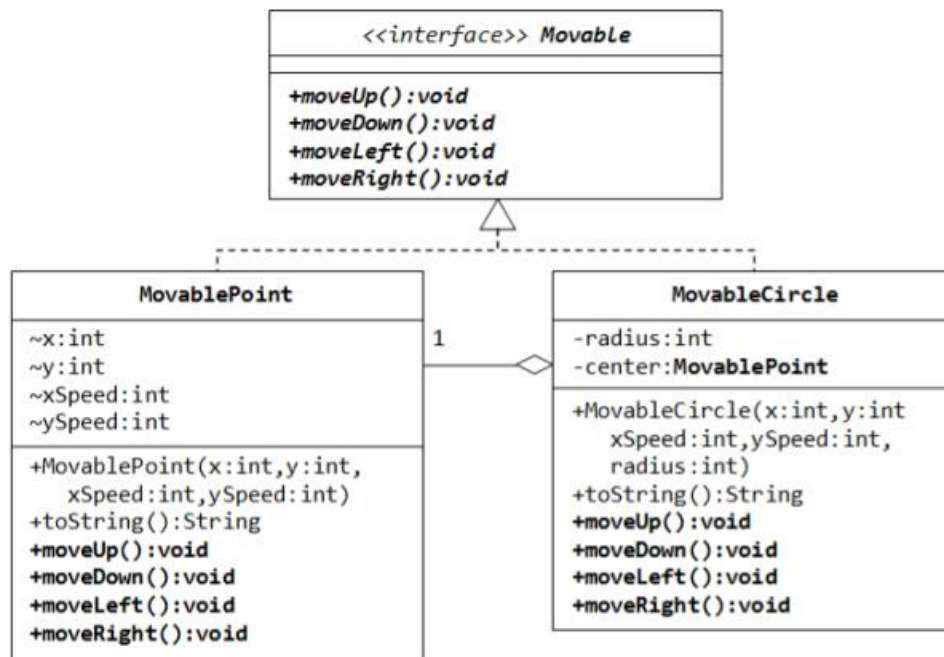


Рисунок 5.2 – Диаграмма реализации интерфейса Movable.

4. 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 имеют одну и ту же скорость (нужен метод это проверяющий). Диаграмма класса `MovableRectangle` представлена на рисунке 5.3.

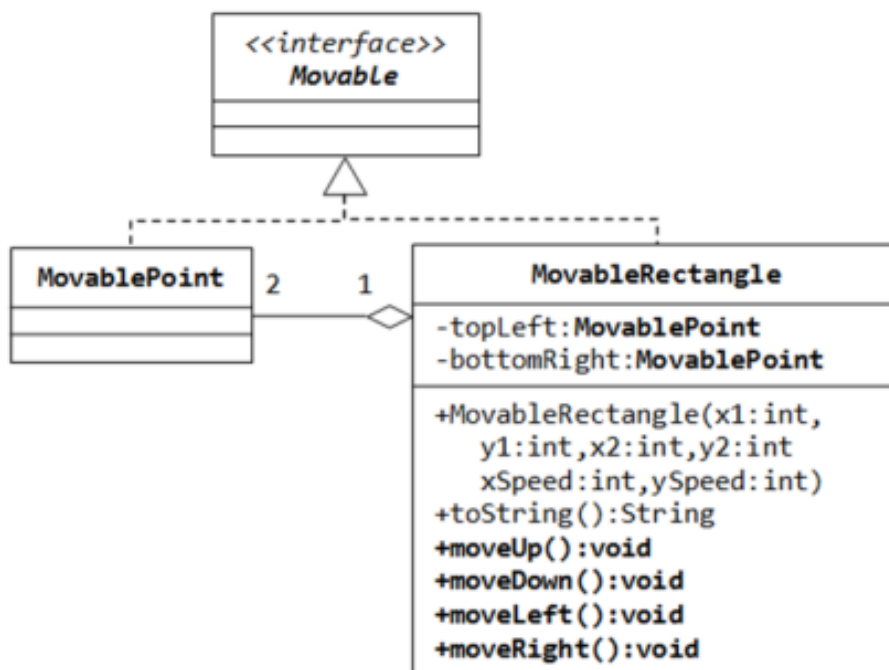


Рисунок 5.3 – Диаграмма класса MovableRectangle

Решение:

1. Программы, реализующие работу классов Shape (листинг 5.1), Circle (листинг 5.2), Rectangle (листинг 5.3) и Square (листинг 5.4).

**Листинг 5.1 – Код программы суперкласса Shape**

```
public abstract class Shape {
    protected String color;
    protected boolean filled;
    public Shape() {
        this.color = "Blue";
        this.filled = true;
    }
    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();
    public abstract String toString();
}
```

**Листинг 5.2 – Код программы класса Circle**

```
public class Circle extends Shape {
    protected double radius;
    public Circle() {
        this.color = "Green";
        this.filled = true;
        radius = 2;
    }
    public Circle(double radius) {
        this.color = "Green";
        this.filled = true;
        this.radius = radius;
    }
    public Circle(double radius, String color, boolean filled) {
        this.color = color;
        this.filled = filled;
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }
    public double getArea() {
```

## Продолжение листинга 5.2

```
        return Math.PI*radius*radius;
    }
    public double getPerimeter() {
        return Math.PI*2*radius;
    }
    public String toString(){
        return "Shape : circle, radius: "+this.radius+", color:
"+this.color;
    }
}
```

## Листинг 5.3 – Код программы класса Rectangle

```
public class Rectangle extends Shape {
    protected double width;
    protected double length;
    public Rectangle(){
        this.width = 4;
        this.length = 5;
        this.color = "Red";
        this.filled = true;
    }
    public Rectangle(double width, double length){
        this.width = width;
        this.length = length;
        this.color = "Red";
        this.filled = true;
    }
    public Rectangle(double width, double length, String color, boolean
filled){
        this.width = width;
        this.length = length;
        this.color = color;
        this.filled = filled;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getWidth() {
        return width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }
    public double getArea() {
        return width*length;
    }
    public double getPerimeter() {
        return 2*(length+width);
    }
    public String toString() {
        return "Shape : rectangle, length: "+this.length+", width:
"+this.width+", color: "+this.color;
    }
}
```



## Листинг 5.4 – Код программы класса Square

```
public class Square extends Rectangle {
    protected double side;
    public Square() {
        this.side = 3;
        this.color = "Red";
        this.filled = true;
    }
    public Square(double side) {
        this.side = side;
        this.color = "Red";
        this.filled = true;
    }
    public Square(double side, String color, boolean filled) {
        this.side = side;
        this.filled = filled;
        this.color = color;
    }

    public double getSide() {
        return side;
    }
    public void setSide(double side) {
        this.side = side;
    }
    public void setWidth(double side) {
        this.side = side;
    }
    public void setLength() {
        this.side = side;
    }
    public String toString() {
        return "Rectangle : square, side: "+this.side;
    }
}
```

2. Программа, реализующая работу класса testClass (листинг 5.5), выполняющая тестирование классов Shape, Circle, Rectangle и Square. Результат работы класса представлен на рисунке 5.3.

## Листинг 5.5 – Код программы тестового класса testClass

```
public class testClass {
    public static void main(String[] args) {
        Shape s1 = new Circle(5.5, "RED", false);
        System.out.println(s1);
        System.out.println(s1.getArea());
        System.out.println(s1.getPerimeter());
        System.out.println(s1.getColor());
        System.out.println(s1.isFilled());
        System.out.println(((Circle)s1).getRadius()); //Тут была ошибка

        Circle c1 = (Circle)s1;
        System.out.println(c1);
        System.out.println(c1.getArea());
        System.out.println(c1.getPerimeter());
        System.out.println(c1.getColor());
        System.out.println(c1.isFilled());
        System.out.println(c1.getRadius());
    }
}
```

## Продолжение листинга 5.5

```
// Shape s2 = new Shape(); - Нельзя создать объект абстрактного класса, но  
// можно создать объект подкласса абстрактного класса  
  
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());  
System.out.println(((Rectangle)s3).getLength()); //Тут была ошибка  
  
Rectangle r1 = (Rectangle)s3;  
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);  
System.out.println(s4);  
System.out.println(s4.getArea());  
System.out.println(s4.getColor());  
System.out.println(((Square)s4).getSide()); //Тут была ошибка  
  
Rectangle r2 = (Rectangle)s4;  
System.out.println(r2);  
System.out.println(r2.getArea());  
System.out.println(r2.getColor());  
System.out.println(((Square)r2).getSide()); //Тут была ошибка  
System.out.println(r2.getLength());  
  
Square sq1 = (Square)r2;  
System.out.println(sq1);  
System.out.println(sq1.getArea());  
System.out.println(sq1.getColor());  
System.out.println(sq1.getSide());  
System.out.println(sq1.getLength());  
}
```

```

Shape : circle, radius: 5.5, color: RED
95.03317777109123
34.55751918948772
RED
false
5.5
Shape : circle, radius: 5.5, color: RED
95.03317777109123
34.55751918948772
RED
false
5.5
Shape : rectangle, length: 2.0, width: 1.0, color: RED
2.0
6.0
RED
2.0
Shape : rectangle, length: 2.0, width: 1.0, color: RED
2.0
RED
2.0
Rectangle : square, side: 6.6
20.0
Red
6.6
Rectangle : square, side: 6.6
20.0
Red
6.6
5.0
Rectangle : square, side: 6.6
20.0
Red
6.6
5.0

Process finished with exit code 0

```

Рисунок 5.3 – Результат работы класса testClass

3. Программы, реализующие работу интерфейса Movable (листинг 5.6), классов MovablePoint (листинг 5.7) и MovableCircle (листинг 5.8).

Листинг 5.6 – Код программы интерфейса Movable

```

public interface Movable {
    public abstract void moveUp();
    public abstract void moveDown();
    public abstract void moveRight();
    public abstract void moveLeft();
}

```

### Листинг 5.7 – Код программы класса MovablePoint

```
public class MovablePoint implements Movable {
    protected int x;
    protected int y;
    protected int ySpeed;
    protected int xSpeed;
    public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
        this.x = x;
        this.y = y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
    }
    public String toString(){
        return "Movable : MovablePoint, x - "+this.x+", y - "+this.y+",
xSpeed - "+this.xSpeed+", ySpeed - "+this.ySpeed;
    }
    public void moveUp(){
        y += ySpeed;
    }

    public void moveDown() {
        y -= ySpeed;
    }

    public void moveRight() {
        x += xSpeed;
    }

    public void moveLeft() {
        x -= xSpeed;
    }
    public boolean equalitySpeeds(MovablePoint other){
        return xSpeed==other.xSpeed && ySpeed==other.ySpeed;
    }
}
```

### Листинг 5.8 – Код программы класса MovableCircle

```
public class MovableCircle implements Movable{
    private int radius;
    private MovablePoint center;
    public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius){
        center = new MovablePoint(x, y, xSpeed, ySpeed);
        this.radius = radius;
    }
    public String toString(){
        return "Movable : MovableCircle, x - "+center.x+", y - "+center.y+",
xSpeed - "+center.xSpeed+", ySpeed - "+center.ySpeed+", radius -
"+this.radius;
    }
    public void moveDown() {
        center.moveDown();
    }
    public void moveLeft() {
        center.moveLeft();
    }
    public void moveRight() {
        center.moveRight();
    }
    public void moveUp() {
        center.moveUp();
    }
}
```

4. Программа, реализующая работу класса MovableRectangle (листинг 5.9).

Листинг 5.9 – Код программы класса MovableRectangle

```
public class MovableRectangle implements Movable{
    private MovablePoint topLeft;
    private MovablePoint bottomRight;
    public MovableRectangle(int x1, int y1, int x2, int y2, int xSpeed, int
ySpeed) {
        topLeft = new MovablePoint(x1, y1, xSpeed, ySpeed);
        bottomRight = new MovablePoint(x2, y2, xSpeed, ySpeed);
    }
    public void moveUp() {
        topLeft.moveUp();
        bottomRight.moveUp();
    }
    public void moveDown() {
        topLeft.moveDown();
        bottomRight.moveDown();
    }

    public void moveRight() {
        topLeft.moveRight();
        bottomRight.moveRight();
    }
    public void moveLeft() {
        topLeft.moveLeft();
        bottomRight.moveLeft();
    }
    public boolean equalitySpeeds() {
        return topLeft.equalitySpeeds(bottomRight);
    }
}
```

### Выводы по работе:

Во время выполнения работы была изучена работа с абстрактными классами и реализация наследования классов.

## **Практическая работа №6**

### **Цель 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.

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово `extends`, после которого идет имя базового класса. Способность к изменению функциональности, унаследованной от базового класса, называется полиморфизмом и является одним из ключевых аспектов объектно-ориентированного программирования наряду с наследованием и инкапсуляцией.

Кроме обычных классов в Java есть абстрактные классы. 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. При определении абстрактных классов используется ключевое слово `abstract`. Кроме обычных методов абстрактный класс может содержать абстрактные методы. Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе.

### **Выполнение лабораторной работы**

#### **Задание:**

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

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

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

Решение:

1. Программы, реализующие работу абстрактного класса Dish (листинг 6.1), классов-наследников Glass (листинг 6.2) и Plate (листинг 6.3), и тестового класса DishTest (листинг 6.4). Результат тестирования представлен на рисунке 6.1.

Листинг 6.1 – Код программы абстрактного класса Dish

```
public abstract class Dish {
    private String material;
    private int price;
    public Dish(String material, int price) {
        this.material = material;
        this.price = price;
    }
    public String getMaterial() {
        return material;
    }
    public void setMaterial(String material) {
        this.material = material;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public int getPrice() {
        return price;
    }
    public abstract String toString();
}
```

Листинг 6.2 – Код программы класса-наследника Glass

```
public class Glass extends Dish{
    private int volume;
    public Glass(String material, int price, int volume){
        super(material, price);
        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }
    public void setVolume(int volume) {
```

## Продолжение листинга 6.2

```
        this.volume = volume;
    }
    public String toString() {
        return "Glass: material - "+getMaterial()+" , price - "+getPrice()+" ,
volume - "+getVolume();
    }
}
```

## Листинг 6.3 – Код программы класса-наследника Plate

```
public class Plate extends Dish {
    private String size;
    public Plate(String material, int price, String size) {
        super(material, price);
        this.size = size;
    }

    public String getSize() {
        return size;
    }

    public void setSize(String size) {
        this.size = size;
    }

    public String toString(){
        return "Plate: material - "+getMaterial()+" , price - "+getPrice()+" ,
size - "+getSize();
    }
}
```

## Листинг 6.4 – Код программы тестового класса DishTest

```
public class DishTest {
    public static void main(String[] args) {
        Dish plate = new Plate("ceramic", 1800, "smoll");
        System.out.println(plate.getPrice());
        System.out.println(plate.getMaterial());
        System.out.println(((Plate)plate).getSize());
        System.out.println(plate.toString());
        plate.setPrice(1500);
        ((Plate)plate).setSize("big");
        System.out.println(plate.toString());

        Dish glass = new Glass("ceramic", 1200, 200);
        glass.setMaterial("glass");
        System.out.println(((Glass)glass).getVolume());
        System.out.println(glass.toString());
        ((Glass)glass).setVolume(250);
        System.out.println(glass.toString());
    }
}
```



```

1800
ceramic
smoll
Plate: material - ceramic, price - 1800, size - smoll
Plate: material - ceramic, price - 1500, size - big
200
Glass: material - glass, price - 1200, volume - 200
Glass: material - glass, price - 1200, volume - 250

Process finished with exit code 0

```

Рисунок 6.1 – Результат работы класса DishTest

2. Программы, реализующие работу абстрактного класса Dog (листинг 6.5), классов-наследников Poodle (листинг 6.6) и Bulldog (листинг 6.7), и тестового класса DogTest (листинг 6.8). Результат тестирования представлен на рисунке 6.2.

Листинг 6.5 – Код программы абстрактного класса Dog

```

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 int getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public abstract String toString();
}

```

Листинг 6.6 – Код программы класса-наследника Poodle

```

public class Poodle extends Dog{
    public Poodle(String name, int age){
        super(name, age);
    }
    public String toString(){
        return "Poodle: name - "+getName()+" , age - "+getAge();
    }
}

```

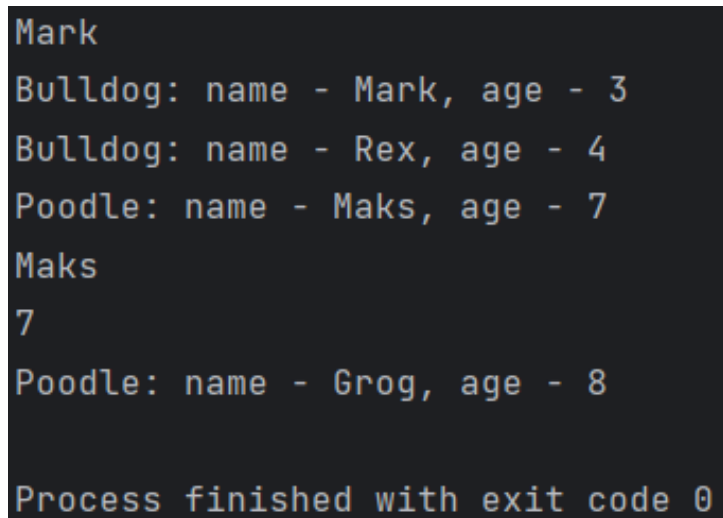
### Листинг 6.7 – Код программы класса-наследника Bulldog

```
public class Bulldog extends Dog{
    public Bulldog(String name, int age){
        super(name, age);
    }
    public String toString(){
        return "Bulldog: name - "+getName()+" , age - "+getAge();
    }
}
```

### Листинг 6.8 – Код программы тестового класса DogTest

```
public class DogTest {
    public static void main(String[] args) {
        Dog d1 = new Bulldog("Mark", 3);
        System.out.println(d1.getName());
        System.out.println(d1.toString());
        d1.setName("Rex");
        d1.setAge(4);
        System.out.println(d1.toString());

        Dog d2 = new Poodle("Maks", 7);
        System.out.println(d2.toString());
        System.out.println(d2.getName());
        System.out.println(d2.getAge());
        d2.setAge(8);
        d2.setName("Grog");
        System.out.println(d2.toString());
    }
}
```



```
Mark
Bulldog: name - Mark, age - 3
Bulldog: name - Rex, age - 4
Poodle: name - Maks, age - 7
Maks
7
Poodle: name - Grog, age - 8

Process finished with exit code 0
```

Рисунок 6.2 – Результат работы класса DogTest

3. Программы, реализующие работу абстрактного класса Furniture (листинг 6.9), классов-наследников Table (листинг 6.10) и Chair (листинг 6.11), класса FurnitureShop (листинг 6.12), реализующий магазин, и тестовых классов FurnitureTest (листинг 6.13) и FurnitureShopTest (листинг 6.14). Результаты тестирования представлены на рисунках 6.3 и 6.4.

### Листинг 6.9 – Код программы абстрактного класса Furniture

```
public abstract class Furniture {
    private int price;
    private String color;
    private String material;
    public Furniture(int price, String color, String material){
        this.price = price;
        this.color = color;
        this.material = material;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getColor() {
        return color;
    }
    public String getMaterial() {
        return material;
    }
    public void setMaterial(String material) {
        this.material = material;
    }
    public abstract String toString();
}
```

### Листинг 6.10 – Код программы класса-наследника Table

```
public class Table extends Furniture {
    public Table(int price, String color, String material){
        super(price, color, material);
    }
    public String toString(){
        return "Table : price - "+getPrice()+" , color - "+getColor()+" ,
material - "+getMaterial();
    }
}
```

### Листинг 6.11 – Код программы класса-наследника Chair

```
public class Chair extends Furniture{
    public Chair(int price, String color, String materiale){
        super(price, color, materiale);
    }
    public String toString(){
        return "Chair : price - "+getPrice()+" , color - "+getColor()+" ,
material - "+getMaterial();
    }
}
```

### Листинг 6.12 – Код программы класса FurnitureShop

```
public class FurnitureShop {
    private Furniture[] furniturePieces;

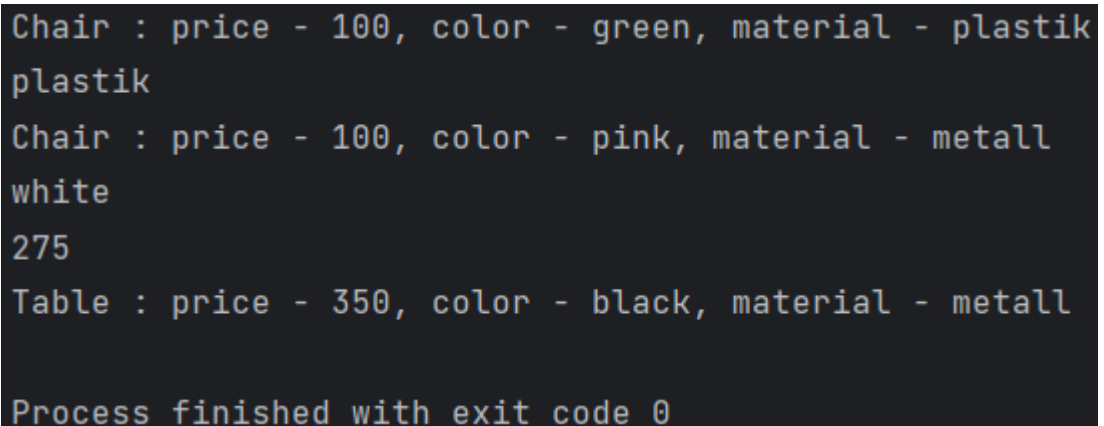
    public FurnitureShop(Furniture[] furniturePieces) {
        this.furniturePieces = furniturePieces;
    }
}
```

## Продолжение листинга 6.12

```
}  
    public void displayFurniture() {  
        for (Furniture furniture : furniturePieces) {  
            System.out.println(furniture.toString());  
        }  
    }  
}
```

## Листинг 6.13 – Код программы тестового класса FurnitureTest

```
public class FurnitureTest {  
    public static void main(String[] args) {  
        Furniture f1 = new Chair(100, "green", "plastik");  
        System.out.println(f1.toString());  
        System.out.println(f1.getMaterial());  
        f1.setColor("pink");  
        f1.setMaterial("metall");  
        System.out.println(f1.toString());  
  
        Furniture f2 = new Table(275, "white", "metall");  
        System.out.println(f2.getColor());  
        System.out.println(f2.getPrice());  
        f2.setPrice(350);  
        f2.setColor("black");  
        System.out.println(f2.toString());  
    }  
}
```



```
Chair : price - 100, color - green, material - plastik  
plastik  
Chair : price - 100, color - pink, material - metall  
white  
275  
Table : price - 350, color - black, material - metall  
  
Process finished with exit code 0
```

Рисунок 6.3 – Результат работы класса FurnitureTest

## Листинг 6.14 – Код программы тестового класса FurnitureShopTest

```
public class FurnitureShopTest {  
    public static void main(String[] args) {  
        Furniture[] furniturePieces = new Furniture[5];  
        furniturePieces[0] = new Chair(100, "green", "metall");  
        furniturePieces[1] = new Table(200, "white", "plastik");  
        furniturePieces[2] = new Chair(150, "black", "plastik");  
        furniturePieces[3] = new Chair(200, "black", "metall");  
        furniturePieces[4] = new Table(200, "black", "metall");  
  
        FurnitureShop furnitureShop = new FurnitureShop(furniturePieces);  
        furnitureShop.displayFurniture();  
    }  
}
```

```
Chair : price - 100, color - green, material - metall  
Table : price - 200, color - white, material - plastik  
Chair : price - 150, color - black, material - plastik  
Chair : price - 200, color - black, material - metall  
Table : price - 200, color - black, material - metall  
  
Process finished with exit code 0
```

Рисунок 6.4 – Результат работы класса FurnitureShopTest

### **Выводы по работе:**

Во время выполнения работы было изучено понятие наследования и освоено навыки реализации наследования.

## **Практическая работа №7**

### **Цель работы**

Введение в событийное программирование на языке Java. Создание приложений на Java с использованием элементов GUI.

### **Теоретическое введение**

Text Fields – текстовое поле или поля для ввода текста (можно ввести только одну строку). Примерами текстовых полей являются поля для ввода логина и пароля, например, используемые, при входе в электронную почту.

Компонент TextArea похож на TextField, но в него можно вводить более одной строки. В качестве примера TextArea можно рассмотреть текст, который мы набираем в теле сообщения электронной почты.

Менеджер BorderLayout разделяет компонент на пять областей (WEST, EAST, NORTH, SOUTH and Center). Другие компоненты могут быть добавлены в любой из этих компонентов пятерками.

С помощью менеджера GridLayout компонент может принимать форму таблицы, где можно задать число строк и столбцов.

Мы можем реализовывать слушателей мыши на компонентах GUI с помощью методов MouseListener.

Добавление меню в программе Java происходит несложно. Java определяет три компонента для обработки:

- JMenuBar: который представляет собой компонент, который содержит меню.
- JMenu: который представляет меню элементов для выбора.
- JMenuItem: представляет собой элемент, который можно кликнуть из меню.

### **Выполнение лабораторной работы**

#### **Задание:**

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”
- 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:

Программа, реализующая работу класса GUIclass класса JFrame, представлена в листинге 7.1. Результат работы программы представлен на рисунке 7.1.

### Листинг 7.1 – Код программы класса GUIclass

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GUIclass extends JFrame {
    private int milanScore = 0;
    private int madridScore = 0;
    JButton milanButton = new JButton("AC Milan");
    JButton madridButton = new JButton("Real Madrid");
    JLabel resultLabel = new JLabel("Result: 0 X 0");
    JLabel winnerLabel = new JLabel("Winner: DRAW");
    JLabel lastScorerLabel = new JLabel("Last Scorer: N/A");

    GUIclass() {
        setLayout(new GridLayout(3, 2));
        setSize(500, 500);
        setTitle("Football Table");
        add(milanButton);
        add(madridButton);
        add(winnerLabel);
        add(lastScorerLabel);
        add(resultLabel);
        madridButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                madridScore++;
                UpdateLabel("Real Madrid");
            }
        });
    }
}
```

## Продолжение листинга 7.1

```
        milanButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                milanScore++;
                UpdateLabel("AC Milan");
            }
        });
    }

    private void UpdateLabel(String score) {
        resultLabel.setText("Result: "+milanScore+" X "+madridScore);
        lastScorerLabel.setText("Last Scorer: "+score);
        if(madridScore < milanScore){
            winnerLabel.setText("Winner: AC Milan");
        }
        else if (madridScore > milanScore){
            winnerLabel.setText("Winner: Real Madrid");
        }
        else{
            winnerLabel.setText("Winner: DRAW");
        }
    }

    public static void main(String[] args) {
        new GUIclass().setVisible(true);
    }
}
```

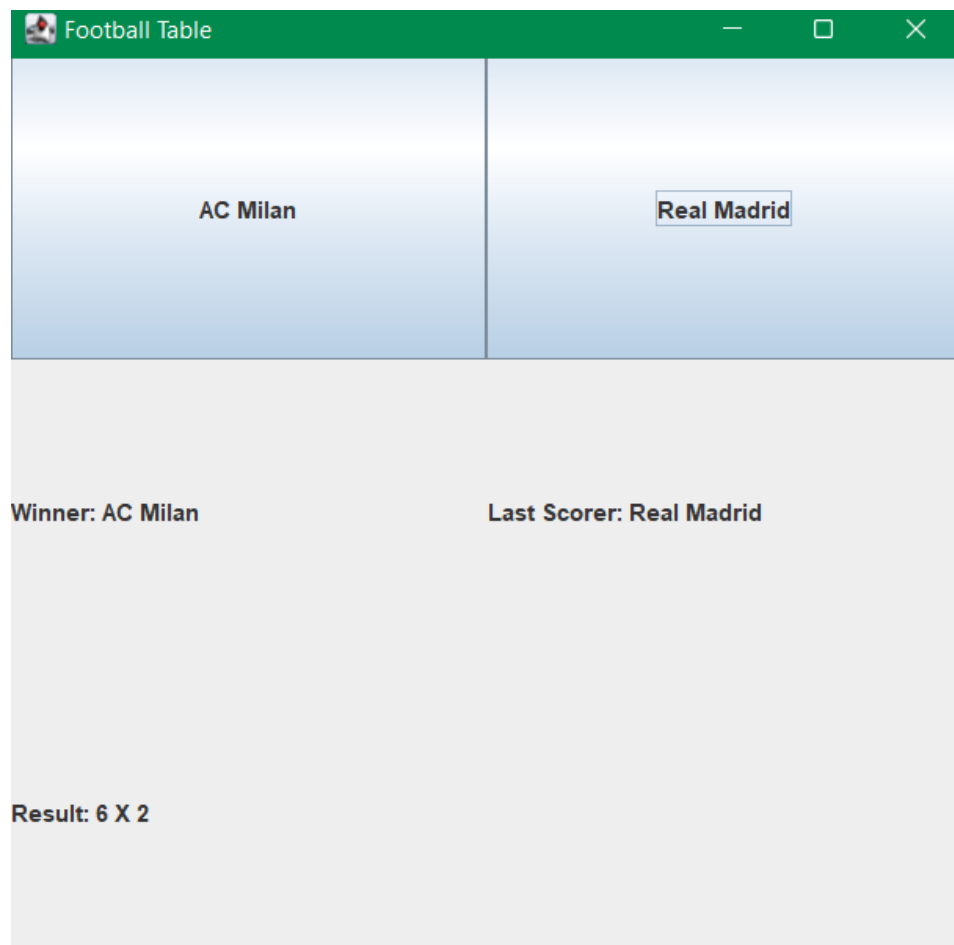


Рисунок 7.1 – Результат работы класса GUIclass



**Выводы по работе:**

Во время выполнения работы были освоены навыки работы с элементами GUI. Была реализована интерактивная программа с различными методами класса JFrame.

## Практическая работа №8

### Цель работы

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

### Теоретическое введение

Теоретические сведения указаны в практической работе №7.

### Выполнение лабораторной работы

#### 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.
2. Создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки.
3. Создать окно, реализовать анимацию, с помощью картинки, состоящей из нескольких кадров.

#### Решение:

1. Программы, реализующие работу абстрактного класса Shape (листинг 8.1), классов-наследников Circle (листинг 8.2), Rectangle (листинг 8.3) и класса WindowGUI (листинг 8.4). Результат работы программы представлен на рисунке 8.1.

#### Листинг 8.1 – Код программы абстрактного класса Shape

```
import java.awt.*;
public abstract class Shape {
    protected int x;
    protected int y;
    protected Color color;
    public abstract void draw(Graphics g);
}
```

#### Листинг 8.2 – Код программы класса-наследника Circle

```
import java.awt.*;
public class Circle extends Shape{
    private int radius;
    public Circle(int x, int y, Color color, int radius){
        this.x = x;
        this.y = y;
        this.color = color;
    }
}
```

## Продолжение листинга 8.2

```
        this.radius = radius;
    }
    public void draw(Graphics g){
        g.setColor(color);
        g.fillOval(x, y, radius, radius);
    }
}
```

## Листинг 8.3 – Код программы класса-наследника Rectangle

```
import java.awt.*;
public class Rectangle extends Shape{
    private int length;
    private int width;
    public Rectangle(int x, int y, Color color, int length, int width){
        this.x = x;
        this.y = y;
        this.color = color;
        this.length = length;
        this.width = width;
    }
    public void draw(Graphics g){
        g.setColor(color);
        g.fillRect(x, y, width, length);
    }
}
```

## Листинг 8.4 – Код программы класса WindowGUI

```
import javax.swing.*;
import java.awt.*;
import java.util.Random;

public class WindowGUI extends JFrame {
    private Shape[] shapes;
    private Random random;
    public WindowGUI() {
        random = new Random();
        Shape[] shapes = new Shape[20];
        setLayout(new FlowLayout());
        setSize(800, 800);

        for(int i = 0; i < shapes.length; i++){
            Color color = getRandomColor();
            int x = random.nextInt(700) + 50;
            int y = random.nextInt(700) + 50;
            int radius = random.nextInt(90) + 10;
            int length = random.nextInt(90) + 10;
            int width = random.nextInt(90) + 10;
            if (random.nextBoolean()){
                shapes[i] = new Circle(x, y, color, radius);
            }
            else{
                shapes[i] = new Rectangle(x, y, color, length, width);
            }
        }
    }

    private Color getRandomColor() {
        int red = random.nextInt(256);
        int green = random.nextInt(256);
        int blue = random.nextInt(256);
        return new Color(red, green, blue);
    }
}
```

## Продолжение листинга 8.4

```
}  
  
public void paint(Graphics g) {  
    super.paint(g);  
  
    for (Shape shape : shapes) {  
        shape.draw(g);  
    }  
}  
  
public static void main(String[] args) {  
    new WindowGUI().setVisible(true);  
}  
}
```

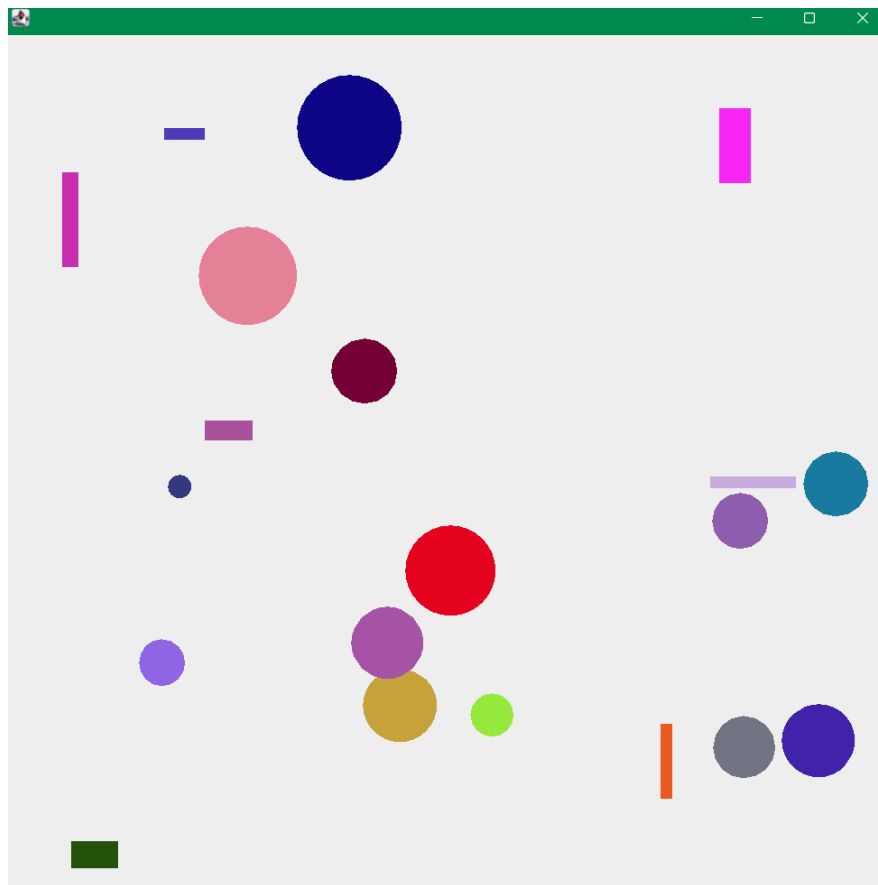


Рисунок 8.1 – Результат работы класса WindowGUI

2. Программа, реализующая работу класса WindowPicture, представлена в листинге 8.5. Результат работы программы представлен на рисунке 8.2.

## Листинг 8.5 – Код программы класса WindowPicture

```
import javax.swing.*;  
import java.awt.*;  
  
public class WindowPicture extends JFrame {  
    public WindowPicture(String imagePath) {  
        ImageIcon imageIcon = new ImageIcon(imagePath);  
    }  
}
```

## Продолжение листинга 8.5

```
        if (imageIcon.getWidth() == -1) {
            System.out.println("По этому пути не найдена картинка");
            System.exit(1);
        }
        JLabel imageLabel = new JLabel(imageIcon);
        setLayout(new BorderLayout());
        add(imageLabel);
        pack();
    }

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Нет пути");
            System.exit(0);
        }
        String imagePath = args[0];
        new WindowPicture(imagePath).setVisible(true);
    }
}
```

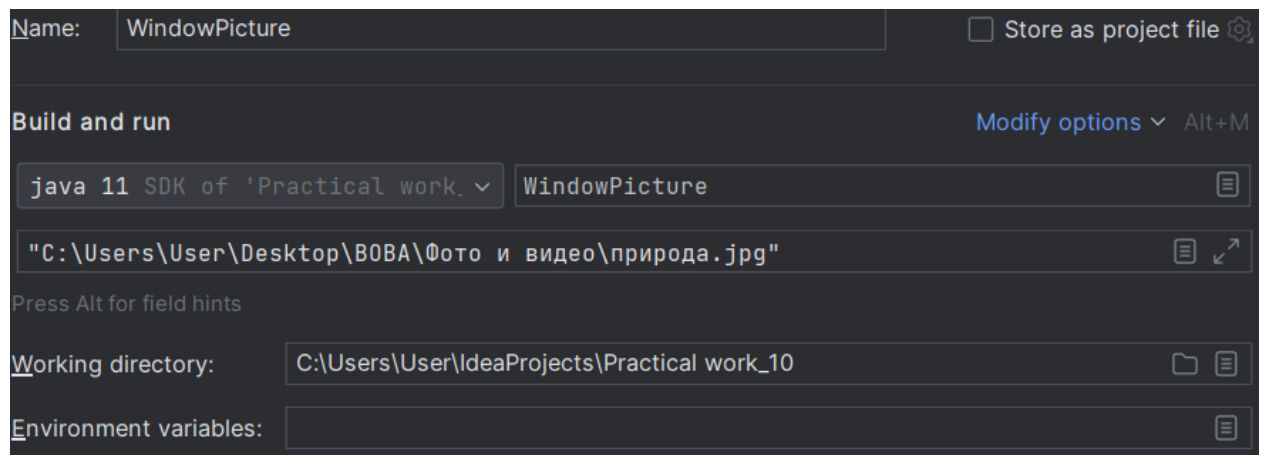


Рисунок 8.2 – Результат работы класса WindowPicture

3. Программа, реализующая работу класса PictureAnimation, представлена в листинге 8.6. Результат работы программы представлен на рисунке 8.3.

Листинг 8.6 – Код программы класса PictureAnimation

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PictureAnimation extends JFrame {
    private Timer animationTimer;
    private int currentFrameIndex;
    public PictureAnimation() {
        JLabel animationLabel = new JLabel();
        add(animationLabel);
        setSize(500, 500);
        setVisible(true);

        ImageIcon[] imageIcon = new ImageIcon[8];
        imageIcon[0] = new ImageIcon("src/cadr0.png");
        imageIcon[1] = new ImageIcon("src/cadr1.png");
        imageIcon[2] = new ImageIcon("src/cadr2.png");
        imageIcon[3] = new ImageIcon("src/cadr3.png");
        imageIcon[4] = new ImageIcon("src/cadr4.png");
        imageIcon[5] = new ImageIcon("src/cadr5.png");
        imageIcon[6] = new ImageIcon("src/cadr6.png");
        imageIcon[7] = new ImageIcon("src/cadr7.png");

        currentFrameIndex = 0;

        animationTimer = new Timer(100, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                animationLabel.setIcon(imageIcon[currentFrameIndex %
imageIcon.length]);
                currentFrameIndex++;
            }
        });
    }
    public void startAnimation() {
        animationTimer.start();
    }

    public static void main(String[] args) {
        PictureAnimation p = new PictureAnimation();
        p.startAnimation();
    }
}
```

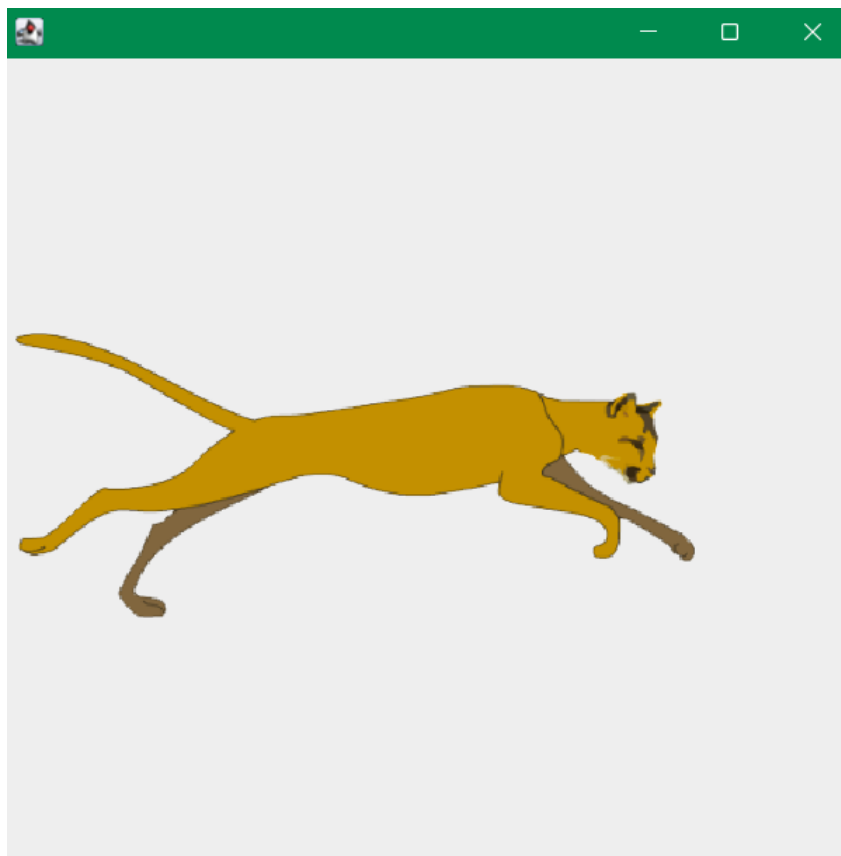


Рисунок 8.3 – Результат работы класса PictureAnimation

**Выводы по работе:**

Во время выполнения работы были получены навыки работы с графическим интерфейсом. Научились создавать рисунки в окне и выводить фотографии с компьютера.

## **Практическая работа №9**

### **Цель работы**

Изучить понятие интерфейса, научиться создавать интерфейсы в Java и применять их в программах.

### **Теоретическое введение**

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, также как и абстрактные методы абстрактных классов не имеют реализации.

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.

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.

### **Выполнение лабораторной работы**

#### 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.).

2. Реализовать интерфейс `Priceable`, имеющий метод `getPrice()`, возвращающий некоторую цену для объекта. Проверить работу для различных классов, сущности которых могут иметь цену.

#### Решение:



1. Программы, реализующие работу интерфейса Nameable (листинг 9.1), классов Planet (листинг 9.2), Car (листинг 9.3), Animal (листинг 9.4) и тестового класса NameableTest (листинг 9.5). Результат работы тестового класса представлен на рисунке 9.1.

Листинг 9.1 – Код программы интерфейса Nameable

```
public interface Nameable {  
    String getName();  
}
```

Листинг 9.2 – Код программы класса Planet

```
public class Planet implements Nameable{  
    public String name;  
    public Planet(String name){  
        this.name = name;  
    }  
    public String getName(){  
        return name;  
    }  
}
```

Листинг 9.3 – Код программы класса Car

```
public class Car implements Nameable{  
    private String name;  
    public Car(String name){  
        this.name = name;  
    }  
    public String getName(){  
        return name;  
    }  
}
```

Листинг 9.4 – Код программы класса Animal

```
public class Animal implements Nameable{  
    private String name;  
    public Animal(String name){  
        this.name = name;  
    }  
    public String getName(){  
        return name;  
    }  
}
```

Листинг 9.5 – Код программы тестового класса NameableTest

```
public class NameableTest {  
    public static void main(String[] args) {  
        Planet planet = new Planet("Sun");  
        Car car = new Car("KIA");  
        Animal animal = new Animal("Cat");  
  
        System.out.println("Planet - "+planet.getName());  
        System.out.println("Car - "+car.getName());  
        System.out.println("Animal - "+animal.getName());  
    }  
}
```

```
Planet - Sun
Car - KIA
Animal - Cat

Process finished with exit code 0
```

Рисунок 9.1 – Результат работы тестового класса NameableTest

2. Программы, реализующие работу интерфейса Priceable (листинг 9.6), классов Electronics (листинг 9.7), BookShop (листинг 9.8) и тестового класса PriceableTest (листинг 9.9). Результат работы тестового класса представлен на рисунке 9.2.

Листинг 9.6 – Код программы интерфейса Priceable

```
public interface Priceable {
    double getPrice();
    String toString();
}
```

Листинг 9.7 – Код программы класса Electronics

```
public class Electronics implements Priceable{
    private String name;
    private double price;
    public Electronics(String name, double price){
        this.name = name;
        this.price = price;
    }
    public double getPrice(){
        return price;
    }

    public String toString(){
        return "Electronics: "+name+" - "+getPrice()+"$";
    }
}
```

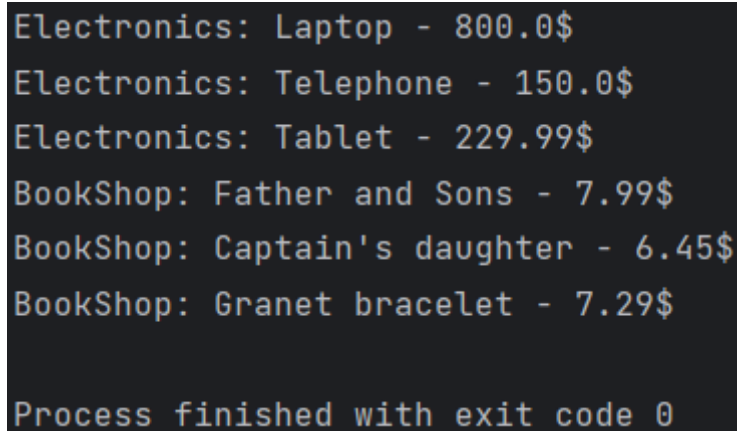
Листинг 9.8 – Код программы класса BookShop

```
public class BookShop {
    private String name;
    private double price;
    public BookShop(String name, double price){
        this.name = name;
        this.price = price;
    }
    public double getPrice(){
        return price;
    }

    public String toString(){
        return "BookShop: "+name+" - "+getPrice()+"$";
    }
}
```

### Листинг 9.9 – Код программы тестового класса PriceableTest

```
public class PriceableTest {  
    public static void main(String[] args) {  
        Electronics el1 = new Electronics("Laptop", 800);  
        Electronics el2 = new Electronics("Telephone", 150);  
        Electronics el3 = new Electronics("Tablet", 229.99);  
        BookShop bs1 = new BookShop("Father and Sons", 7.99);  
        BookShop bs2 = new BookShop("Captain's daughter", 6.45);  
        BookShop bs3 = new BookShop("Granet bracelet", 7.29);  
  
        System.out.println(el1.toString());  
        System.out.println(el2.toString());  
        System.out.println(el3.toString());  
        System.out.println(bs1.toString());  
        System.out.println(bs2.toString());  
        System.out.println(bs3.toString());  
    }  
}
```



```
Electronics: Laptop - 800.0$  
Electronics: Telephone - 150.0$  
Electronics: Tablet - 229.99$  
BookShop: Father and Sons - 7.99$  
BookShop: Captain's daughter - 6.45$  
BookShop: Granet bracelet - 7.29$  
  
Process finished with exit code 0
```

Рисунок 9.2 – Результат работы тестового класса PriceableTest

#### Выводы по работе:

Во время выполнения работы были освоены навыки создания интерфейсов и применения их при выполнении задач.

## **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.

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).

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.

### **Выполнение лабораторной работы**

#### Задание:

#### 1. Разложение на множители

Дано натуральное число  $n > 1$ . Выведите все простые множители этого числа в порядке не убывания с учетом кратности. Алгоритм должен иметь сложность  $O(\log n)$ .

#### 2. Палиндром

Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO. При решении этой задачи нельзя пользоваться циклами, в решениях на питоне нельзя использовать срезы с шагом, отличным от 1.

### 3. Без двух нулей

Даны числа  $a$  и  $b$ . Определите, сколько существует последовательностей из  $a$  нулей и  $b$  единиц, в которых никакие два нуля не стоят рядом.

### 4. 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.

### 5. Количество единиц

Дана последовательность натуральных чисел (одно число в строке), завершающаяся двумя числами 0 подряд. Определите, сколько раз в этой последовательности встречается число 1. Числа, идущие после двух нулей, необходимо игнорировать. В этой задаче нельзя использовать глобальные переменные и параметры, передаваемые в функцию. Функция получает данные, считывая их с клавиатуры, а не получая их в виде параметров.

#### Решение:

1. Программа, реализующая работу класса `number_7` (листинг 10.1), в котором реализован метод `recursion`. Результат работы программы представлен на рисунке 10.1.

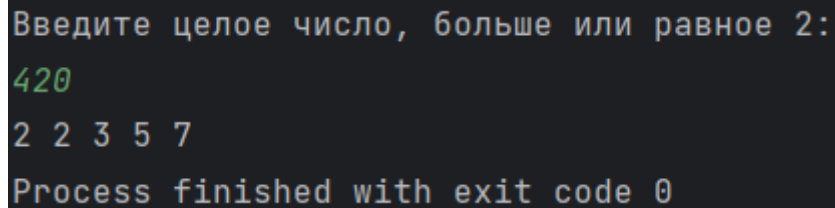
#### Листинг 10.1 – Код программы класса `number_7`

```
import java.util.Scanner;

public class number_7 {
    public static void main(String[] args) {
        System.out.println("Введите целое число, больше или равное 2: ");
        Scanner scanner = new Scanner(System.in);
        if (!scanner.hasNextInt()) {
            System.out.println("Некорректный ввод!");
            return;
        }
        int number = scanner.nextInt();
        if (number < 2) {
            System.out.println("Число меньше 2");
            return;
        }
        recursion(number, 2);
    }
}
```

### Продолжение листинга 10.1

```
public static void recursion(int n, int del){
    if (n >= del){
        if(n % del == 0){
            System.out.printf("%d ", del);
            recursion(n/del, del);
        }
        else{
            recursion(n, del+1);
        }
    }
}
```



Введите целое число, больше или равное 2:  
420  
2 2 3 5 7  
Process finished with exit code 0

Рисунок 10.1 – Результат работы класса number\_7

2. Программа, реализующая работу класса number\_8 (листинг 10.2), в котором реализован метод palindrome. Результат работы программы представлен на рисунке 10.2.

### Листинг 10.2 – Код программы класса number\_8

```
import java.util.Scanner;

public class number_8 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите палиндром: ");
        String word = scanner.nextLine();
        if(palindrome(word)){
            System.out.println("YES");
        }
        else{
            System.out.println("NO");
        }
    }

    public static boolean palindrome(String word) {
        if(word.length() <= 1){
            return true;
        }
        else if(word.charAt(0) == word.charAt(word.length() - 1)){
            return palindrome(word.substring(1, word.length() - 1));
        }
        else{
            return false;
        }
    }
}
```

```

Введите палиндром:
ргесасерг
YES

Process finished with exit code 0

```

Рисунок 10.2 – Результат работы класса number\_8

3. Программа, реализующая работу класса number\_9 (листинг 10.3), в котором реализован метод recursion. Результат работы программы представлен на рисунке 10.3.

Листинг 10.3 – Код программы класса number\_9

```

import java.util.Scanner;

public class number_9 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите число a: ");
        int a = scanner.nextInt();
        System.out.print("Введите число b: ");
        int b = scanner.nextInt();
        int count = recursion(a, b);
        System.out.println(count);
    }
    public static int recursion(int a, int b) {
        if (a > b+1) {
            return 0;
        }
        if (a == 0 || b == 0) {
            return 1;
        }
        return recursion(a, b-1) + recursion(a-1, b-1);
    }
}

```

```

Введите число a: 2
Введите число b: 3
6

Process finished with exit code 0

```

Рисунок 10.3 – Результат работы класса number\_9

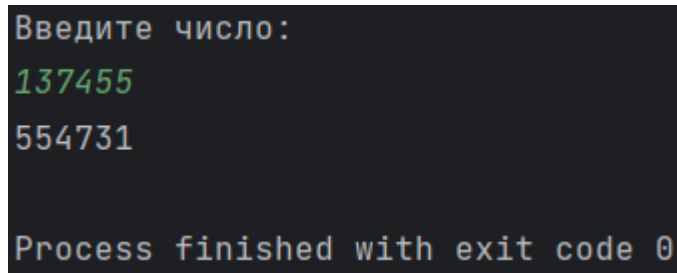
4. Программа, реализующая работу класса number\_10 (листинг 10.4), в котором реализован метод reverseNumber. Результат работы программы представлен на рисунке 10.4.

#### Листинг 10.4 – Код программы класса number\_10

```
import java.util.Scanner;

public class number_10 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите число: ");
        int number = scanner.nextInt();
        int reversed = reverseNumber(number);
        System.out.println(reversed);
    }

    public static int reverseNumber(int num) {
        if (num < 10) {
            return num;
        } else {
            int lastDigit = num % 10;
            int remainingDigits = num / 10;
            int reversedRemaining = reverseNumber(remainingDigits);
            int reversed = (int) (lastDigit * Math.pow(10,
Integer.toString(reversedRemaining).length()) + reversedRemaining);
            return reversed;
        }
    }
}
```



```
Введите число:
137455
554731

Process finished with exit code 0
```

Рисунок 10.4 – Результат работы класса number\_10

5. Программа, реализующая работу класса number\_11 (листинг 10.5), в котором реализован метод strNumber. Результат работы программы представлен на рисунке 10.5.

#### Листинг 10.5 – Код программы класса number\_11

```
import java.util.Scanner;

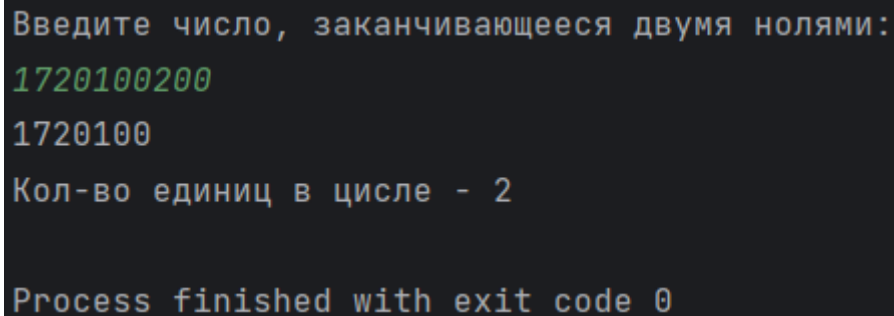
public class number_11 {
    public static void strNumber() {
        System.out.println("Введите число, заканчивающееся двумя нолями: ");
        Scanner scanner = new Scanner(System.in);
        if (!scanner.hasNextLong()) {
            System.out.println("Это не число!");
            return;
        }
        String number = scanner.nextLine();
        boolean flag = false;
        int k = 0;
        for (int i = 1; i < number.length() - 1; i++) {
            if (number.charAt(i - 1) == '0' && number.charAt(i) == '0') {
                //1720100200
                k += i;
            }
        }
    }
}
```



## Продолжение листинга 10.5

```
        flag = true;
        break;
    }
}
if(!flag){
    System.out.println("Число не заканчивается двумя нолями!");
    return;
}
String trimmednumber = number.substring(0, k+1);
System.out.println(trimmednumber);
k = 0;
for(int i = 0; i < trimmednumber.length(); i++){
    if(trimmednumber.charAt(i) == '1'){
        k += 1;
    }
}
System.out.println("Кол-во единиц в числе - "+k);
}

public static void main(String[] args) {
    strNumber();
}
}
```



```
Введите число, заканчивающееся двумя нолями:
1720100200
1720100
Кол-во единиц в числе - 2
Process finished with exit code 0
```

Рисунок 10.5 – Результат работы класса number\_11

### Выводы по работе:

Во время выполнения работы были получены навыки работы с рекурсивными алгоритмами.

## **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.

Техника программирования сортировок в Java отличается от написания алгоритмов на процедурных языках программирования. При написании кода большим преимуществом является использование основного принципа ООП – полиморфизма. Напомним, что класс, который реализует интерфейс Comparable определяет метод compareTo(), чтобы определить относительный порядок своих объектов.

Таким образом мы можем использовать полиморфизм, чтобы разработать обобщенную сортировку для любого набора Comparable объектов. При разработке класса, реализующего метод сортировки, нужно помнить, что метод принимает в качестве параметра массив объектов типа Comparable или фактически полиморфных ссылок.

### **Выполнение лабораторной работы**

#### **Задание:**

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

2. Напишите класс `SortingStudentsByGPA` который реализует интерфейс `Comparator` таким образом, чтобы сортировать список студентов по их итоговому баллам в порядке убывания с использованием алгоритма быстрой сортировки.

3. Напишите программу, которая объединяет два списка данных о студентах в один отсортированный список с использованием алгоритма сортировки слиянием.

Решение:

1. Программы, реализующие работу класса `Student` (листинг 11.1) и тестового класса `TestClass` (листинг 11.2). Результат работы программы представлен на рисунке 11.1.

Листинг 11.1 – Код программы класса `Student`

```
public class Student {
    private String surname;

    private int id;

    private double GPA;

    public Student(String surname, int id, double GPA) {
        this.surname = surname;
        this.id = id;
        this.GPA = GPA;
    }

    public double getGPA() {
        return GPA;
    }
    public void setGPA(double GPA) {
        this.GPA = GPA;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getSurname() {
        return surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }

    @Override
    public String toString() {
        return "surname - "+getSurname()+" , ID - "+getId()+" , GPA = "+getGPA();
    }
}
```

## Листинг 11.2 – Код программы тестового класса TestClass

```

public class TestClass {
    public static void main(String[] args) {
        Student[] studentsArr1 = new Student[]{
            new Student("Рубинин", 22301, 4.87),
            new Student("Кедров", 50385, 3.33),
            new Student("Выжилинин", 33078, 4.23),
            new Student("Мирзоян", 48292, 4.02),
            new Student("Иванов", 44908, 4.77)
        };

        Student[] studentsArr2 = new Student[]{
            new Student("Королев", 35689, 3.32),
            new Student("Рябинин", 13553, 4.34),
            new Student("Куropаткин", 43224, 4.23),
            new Student("Суматохин", 15702, 4.11),
            new Student("Смирнов", 33413, 4.89)
        };

        System.out.println("-----Исходный массив-----");
        for(int i = 0; i < studentsArr1.length; i++){
            System.out.println(studentsArr1[i].toString());
        }
        for (int i = 0; i < studentsArr1.length; i++){
            Student temp = studentsArr1[i];
            int j = i - 1;
            while (j >= 0 && studentsArr1[j].getId() > temp.getId()){
                studentsArr1[j+1] = studentsArr1[j];
                j--;
            }
            studentsArr1[j+1] = temp;
        }
        System.out.println("-----Массив, отсортированный вставками по ID-----");
        for (int i = 0; i < studentsArr1.length; i++){
            System.out.println(studentsArr1[i].toString());
        }

        System.out.println("-----Исходный массив-----");
        for(int i = 0; i < studentsArr2.length; i++){
            System.out.println(studentsArr2[i].toString());
        }
        SortingStudentsByGPA s = new SortingStudentsByGPA();
        s.QuickSort(studentsArr2, 0, studentsArr2.length-1);
        System.out.println("-----Массив, отсортированный слиянием по GPA в порядке убывания-----");
        for (int i = 0; i < studentsArr2.length; i++) {
            System.out.println(studentsArr2[i].toString());
        }

        Student[] newArr = new Student[studentsArr1.length +
studentsArr2.length];
        System.arraycopy(studentsArr1, 0, newArr, 0, studentsArr1.length);
        System.arraycopy(studentsArr2, 0, newArr, studentsArr1.length,
studentsArr2.length);
        System.out.println("-----Исходный объединенный массив-----");
        for (int i = 0; i < newArr.length; i++){
            System.out.println(newArr[i].toString());
        }
        MergeSortingStudentsBySurname m = new
MergeSortingStudentsBySurname();
        m.mergeSort(newArr, newArr.length);
        System.out.println("-----Отсортированный слиянием объединенный массив-----");
    }
}

```

## Продолжение листинга 11.2

```
        for (int i = 0; i < newArr.length; i++) {
            System.out.println(newArr[i].toString());
        }
    }
}
```

```
-----Исходный массив-----
surname - Рубинин, ID - 22301, GPA = 4.87
surname - Кедров, ID - 50385, GPA = 3.33
surname - Выжилинин, ID - 33078, GPA = 4.23
surname - Мирзоян, ID - 48292, GPA = 4.02
surname - Иванов, ID - 44908, GPA = 4.77
-----Массив, отсортированный вставками по ID-----
surname - Рубинин, ID - 22301, GPA = 4.87
surname - Выжилинин, ID - 33078, GPA = 4.23
surname - Иванов, ID - 44908, GPA = 4.77
surname - Мирзоян, ID - 48292, GPA = 4.02
surname - Кедров, ID - 50385, GPA = 3.33
```

Рисунок 11.1 – Результат работы тестового класса TestClass

2. Программа, реализующая работу класса `SortingStudentsByGPA`, представлена в листинге 11.3. Результат работы программы представлен на рисунке 11.2.

## Листинг 11.3 – Код программы класса `SortingStudentsByGPA`

```
import java.util.Comparator;

public class SortingStudentsByGPA implements Comparator<Student> {

    @Override
    public int compare(Student s1, Student s2) {
        return Double.compare(s1.getGPA(), s2.getGPA());
    }

    public void QuickSort(Student[] arr, int begin, int end) {
        if (begin < end) {
            int partindex = part(arr, begin, end);
            QuickSort(arr, begin, partindex-1);
            QuickSort(arr, partindex+1, end);
        }
    }

    public int part(Student[] arr, int begin, int end) {
        Student pt = arr[end];
        int i = begin-1;
        for (int j = begin; j < end; j++) {
            if (compare(arr[j], pt) >= 0) {
                i++;
                Student s = arr[i];
                arr[i] = arr[j];
                arr[j] = s;
            }
        }
    }
}
```

### Продолжение листинга 11.3

```
    }  
    Student s = arr[i+1];  
    arr[i+1] = arr[end];  
    arr[end] = s;  
    return i+1;  
}  
}
```

```
-----Исходный массив-----  
surname - Королев, ID - 35689, GPA = 3.32  
surname - Рябинин, ID - 13553, GPA = 4.34  
surname - Куропаткин, ID - 43224, GPA = 4.23  
surname - Суматохин, ID - 15702, GPA = 4.11  
surname - Смирнов, ID - 33413, GPA = 4.89  
-----Массив, отсортированный слиянием по GPA в порядке убывания-----  
surname - Смирнов, ID - 33413, GPA = 4.89  
surname - Рябинин, ID - 13553, GPA = 4.34  
surname - Куропаткин, ID - 43224, GPA = 4.23  
surname - Суматохин, ID - 15702, GPA = 4.11  
surname - Королев, ID - 35689, GPA = 3.32
```

Рисунок 11.2 – Результат работы тестового класса TestClass

3. Программа, реализующая работу класса MergeSortingStudentsBySurname, представлена в листинге 11.4. Результат работы программы представлен на рисунке 11.3.

### Листинг 11.4 – Код программы класса MergeSortingStudentsBySurname

```
import java.util.Comparator;  
  
public class MergeSortingStudentsBySurname implements Comparator<Student> {  
  
    @Override  
    public int compare(Student s1, Student s2) {  
        return s1.getSurname().compareTo(s2.getSurname());  
    }  
    public void mergeSort(Student[] s, int n) {  
        if (n < 2) {  
            return;  
        }  
        int mid = n / 2;  
        Student[] left = new Student[mid];  
        Student[] right = new Student[n - mid];  
        for (int i = 0; i < mid; i++) {  
            left[i] = s[i];  
        }  
        for (int i = mid; i < n; i++) {  
            right[i - mid] = s[i];  
        }  
        mergeSort(left, mid);  
        mergeSort(right, n - mid);  
        merge(s, left, right, mid, n - mid);  
    }  
    public void merge(Student[] a, Student[] l, Student[] rightA, int left,  
int right) {  
        int i = 0, j = 0, k = 0;
```

## Продолжение листинга 11.4

```
        while (i < left && j < right) {
            if (compare(l[i],rightA[j]) <= 0) {
                a[k++] = l[i++];
            }
            else {
                a[k++] = rightA[j++];
            }
        }
        while (i < left) {
            a[k++] = l[i++];
        }
        while (j < right) {
            a[k++] = rightA[j++];
        }
    }
}
```

```
-----Исходный объединенный массив-----
surname - Рубинин, ID - 22301, GPA = 4.87
surname - Выжилинин, ID - 33078, GPA = 4.23
surname - Иванов, ID - 44908, GPA = 4.77
surname - Мирзоян, ID - 48292, GPA = 4.02
surname - Кедров, ID - 50385, GPA = 3.33
surname - Смирнов, ID - 33413, GPA = 4.89
surname - Рябинин, ID - 13553, GPA = 4.34
surname - Куропаткин, ID - 43224, GPA = 4.23
surname - Суматохин, ID - 15702, GPA = 4.11
surname - Королев, ID - 35689, GPA = 3.32
-----Отсортированный слиянием объединенный массив-----
surname - Выжилинин, ID - 33078, GPA = 4.23
surname - Иванов, ID - 44908, GPA = 4.77
surname - Кедров, ID - 50385, GPA = 3.33
surname - Королев, ID - 35689, GPA = 3.32
surname - Куропаткин, ID - 43224, GPA = 4.23
surname - Мирзоян, ID - 48292, GPA = 4.02
surname - Рубинин, ID - 22301, GPA = 4.87
surname - Рябинин, ID - 13553, GPA = 4.34
surname - Смирнов, ID - 33413, GPA = 4.89
surname - Суматохин, ID - 15702, GPA = 4.11
```

Рисунок 11.3 – Результат работы тестового класса TestClass

### Выводы по работе:

Во время выполнения работы были освоены техники сортировки на языке Java, изучили принцип работы интерфейса Comparator и метода compareTo().

## 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.

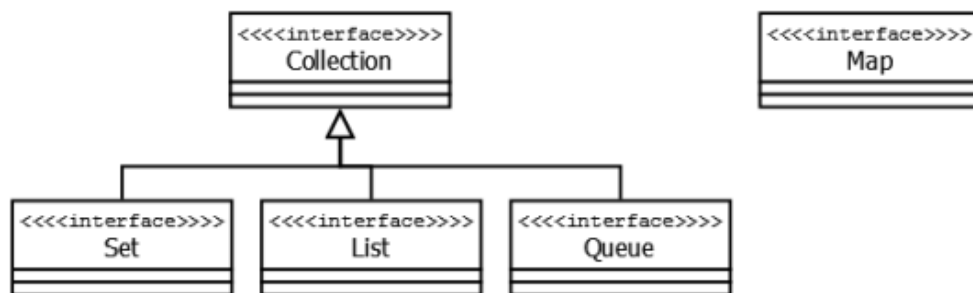


Рисунок 12.1 – Пример Java Collections

LinkedList – вид реализации List. Позволяет хранить любые данные, включая null. Данная коллекция реализована на основе двунаправленного связного списка.

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

Интерфейс Queue описывает коллекции с predetermined способом вставки и извлечения элементов, а именно очереди FIFO (first-in-first-out).

ArrayDeque – реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out).

### Выполнение лабораторной работы

#### Задание:

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).

#### Решение:

Программа, реализующая работу класса DrinkerPlay, представлена в листинге 12.1. Результаты тестирования программы представлены на рисунках 12.2 – 12.5.

#### Листинг 12.1 – Код программы класса DrinkerPlay

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class DrinkerPlay {
    public static void main(String[] args) {
        System.out.println("Игра <Пьяница>. У каждого игрока по 5 карт. Все карты уникальные от 0 до 9");
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите 5 карт первого игрока");
        if(!scanner.hasNextInt()){
            System.out.println("Ошибка ввода. Присутствуют не игровые карты");
            return;
        }
        String first = scanner.nextLine();
        System.out.println("Введите 5 карт второго игрока");
        if(!scanner.hasNextInt()){
            System.out.println("Ошибка ввода. Присутствуют не игровые карты");
            return;
        }
        String second = scanner.nextLine();
        if (first.length() != 5 || second.length() != 5){
            System.out.println("Ошибка ввода. Не у каждого игрока по 5 карт");
            return;
        }
        int p1 = Integer.parseInt(first);
        int p2 = Integer.parseInt(second);
    }
}
```

## Продолжение листинга 12.1

```
int[] arr = new int[10];
for(int i = 0; i < 5; i++){
    arr[p1%10] += 1;
    arr[p2%10] += 1;
    p1 /= 10;
    p2 /= 10;
}
for (int i = 0; i < arr.length; i++){
    if(arr[i] != 1){
        System.out.println("Ошибка ввода. Не все карты уникальные");
        return;
    }
}
Queue<Integer> firstQUEUE = new LinkedList<>();
Queue<Integer> secondQUEUE = new LinkedList<>();

for(int i = 0; i < 5; i++){
    firstQUEUE.add(Integer.valueOf(first.charAt(i)));
    secondQUEUE.add(Integer.valueOf(second.charAt(i)));
}
int count = 0;
while (count <= 106){
    int firstCard = firstQUEUE.poll();
    int secondCard = secondQUEUE.poll();
    if (firstCard == '0' && secondCard == '9'){
        firstQUEUE.add(firstCard);
        firstQUEUE.add(secondCard);
    }
    else if (firstCard == '9' && secondCard == '0'){
        secondQUEUE.add(secondCard);
        secondQUEUE.add(firstCard);
    }
    else if (firstCard > secondCard){
        firstQUEUE.add(firstCard);
        firstQUEUE.add(secondCard);
    }
    else if (secondCard > firstCard){
        secondQUEUE.add(secondCard);
        secondQUEUE.add(firstCard);
    }
    count++;

    if(firstQUEUE.isEmpty()){
        System.out.println("second "+count);
        return;
    }
    if(secondQUEUE.isEmpty()){
        System.out.println("first "+count);
        return;
    }
}
System.out.println("botva");
}
```

```
Игра <Пьяница>. У каждого игрока по 5 карт. Все карты уникальные от 0 до 9
Введите 5 карт первого игрока
14623
Введите 5 карт второго игрока
75с80
Ошибка ввода. Присутствуют не игровые карты

Process finished with exit code 0
```

Рисунок 12.2 – Реакция программы на наличие не игровых карт

```
Игра <Пьяница>. У каждого игрока по 5 карт. Все карты уникальные от 0 до 9
Введите 5 карт первого игрока
123456
Введите 5 карт второго игрока
78930
Ошибка ввода. Не у каждого игрока по 5 карт

Process finished with exit code 0
```

Рисунок 12.3 – Проверка каждого игрока на наличие 5 игровых карт

```
Игра <Пьяница>. У каждого игрока по 5 карт. Все карты уникальные от 0 до 9
Введите 5 карт первого игрока
14367
Введите 5 карт второго игрока
80945
Ошибка ввода. Не все карты уникальные

Process finished with exit code 0
```

Рисунок 12.4 – Проверка на уникальность игровых карт

```
Игра <Пьяница>. У каждого игрока по 5 карт. Все карты уникальные от 0 до 9
Введите 5 карт первого игрока
13579
Введите 5 карт второго игрока
24680
second 5

Process finished with exit code 0
```

Рисунок 12.5 – Результат работы программы

### Выводы по работе:

Во время выполнения работы были получены навыки работы с различными типами контейнеров класса Java Collections 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. Чтобы создать объект `FileWriter`, можно использовать следующий конструктор:

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(FileDescriptor fd)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)
```

Так, в конструктор передается либо путь к файлу в виде строки, либо объект `File`, который ссылается на конкретный текстовый файл. Параметр `append` указывает, должны ли данные дозаписываться в конец файла (если параметр равен `true`), либо файл должен перезаписываться.

Класс `FileReader` наследуется от абстрактного класса `Reader` и предоставляет функциональность для чтения текстовых файлов. Для создания объекта `FileReader` мы можем использовать один из его конструкторов:

```
FileReader(String fileName)
FileReader(File file)
FileReader(FileDescriptor fd)
```

Далее, используя методы, определенные в базовом классе `Reader`, произвести чтение файла.

### Выполнение лабораторной работы

#### Задание:

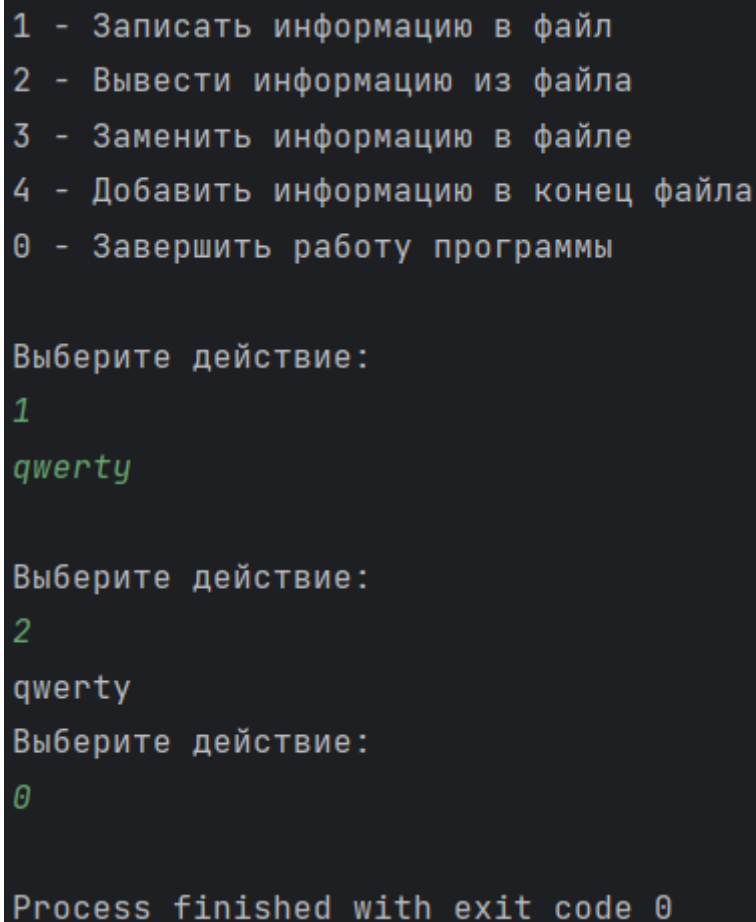
1. Реализовать запись в файл введенной с клавиатуры информации.
2. Реализовать вывод информации из файла на экран.
3. Заменить информацию в файле на информацию, введенную с клавиатуры.
4. Добавить в конец исходного файла текст, введенный с клавиатуры.

#### Решение:

1. Программа, реализующая работу метода writeFile, представлена в листинге 13.1. Результат работы метода представлен на рисунке 13.1.

Листинг 13.1 – Код программы метода writeFile

```
public static void writeFile() {  
    try(FileWriter writer = new FileWriter("fileLab13.txt", false)) {  
        Scanner scanner = new Scanner(System.in);  
        String text = scanner.next();  
        writer.write(text);  
    }  
    catch(IOException ex){  
        System.out.println(ex.getMessage());  
    }  
}
```



```
1 - Записать информацию в файл  
2 - Вывести информацию из файла  
3 - Заменить информацию в файле  
4 - Добавить информацию в конец файла  
0 - Завершить работу программы  
  
Выберите действие:  
1  
qwerty  
  
Выберите действие:  
2  
qwerty  
Выберите действие:  
0  
  
Process finished with exit code 0
```

Рисунок 13.1 – Результат работы метода writeFile

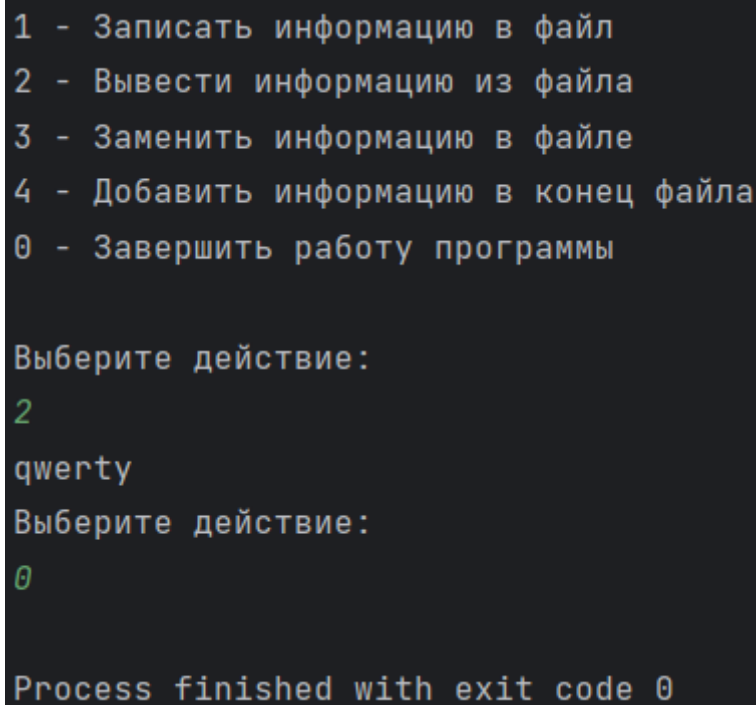
2. Программа, реализующая работу метода readFile, представлена в листинге 13.2. Результат работы метода представлен на рисунке 13.2.

Листинг 13.2 – Код программы метода readFile

```
public static void readFile() {  
    try(FileReader reader = new FileReader("fileLab13.txt"))  
    {  
        // чтение по СИМВОЛАМ  
        int c;  
        while((c=reader.read())!=-1){  
            System.out.print((char)c);  
        }  
    }  
}
```

### Продолжение листинга 13.2

```
public static void readFile() {  
    try(FileReader reader = new FileReader("fileLab13.txt"))  
    {  
        // чтение по символам  
        int c;  
        while((c=reader.read())!=-1){  
            System.out.print((char)c);  
        }  
    }  
    catch (IOException ex){  
        System.out.println(ex.getMessage());  
    }  
}
```



```
1 - Записать информацию в файл  
2 - Вывести информацию из файла  
3 - Заменить информацию в файле  
4 - Добавить информацию в конец файла  
0 - Завершить работу программы  
  
Выберите действие:  
2  
qwerty  
Выберите действие:  
0  
  
Process finished with exit code 0
```

Рисунок 13.2 – Результат работы метода readFile

3. Программа, реализующая работу метода replaceFile, представлена в листинге 13.3. Результат работы метода представлен на рисунке 13.3.

### Листинг 13.3 – Код программы метода replaceFile

```
public static void replaceFile() {  
    try(FileWriter writer = new FileWriter("fileLab13.txt", false)) {  
        Scanner scanner = new Scanner(System.in);  
        String text = scanner.next();  
        writer.write(text);  
    }  
    catch (IOException ex){  
        System.out.println(ex.getMessage());  
    }  
}
```

```

1 - Записать информацию в файл
2 - Вывести информацию из файла
3 - Заменить информацию в файле
4 - Добавить информацию в конец файла
0 - Завершить работу программы

Выберите действие:
2
qwerty
Выберите действие:
3
flabstik

Выберите действие:
2
flabstik
Выберите действие:
0

Process finished with exit code 0

```

Рисунок 13.3 – Результат работы метода replaceFile

4. Программа, реализующая работу метода appendFile, представлена в листинге 13.4. Результат работы метода представлен на рисунке 13.4.

Листинг 13.4 – Код программы метода appendFile

```

public static void appendFile() {
    try(FileWriter writer = new FileWriter("fileLab13.txt", true)) {
        Scanner scanner = new Scanner(System.in);
        String text = scanner.next();
        PrintWriter printWriter = new PrintWriter(writer);
        printWriter.println(text);
    }
    catch(IOException ex){
        System.out.println(ex.getMessage());
    }
}

```

```
1 - Записать информацию в файл
2 - Вывести информацию из файла
3 - Заменить информацию в файле
4 - Добавить информацию в конец файла
0 - Завершить работу программы

Выберите действие:
2
flabstik
Выберите действие:
4
#qwdqfqcef

Выберите действие:
2
flabstik#qwdqfqcef

Выберите действие:
0

Process finished with exit code 0
```

Рисунок 13.4 – Результат работы метода appendFile

### **Выводы по работе:**

Во время выполнения работы были получены навыки работы с файлами, изучили работу классов FileWriter, ReadWriter, PrintWriter.



## Практическая работа №14

### Цель работы

Изучить работу списков ожидания.

### Теоретическое введение

**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.

### Выполнение лабораторной работы

#### Задание:

Исследуйте UML диаграмму классов на рисунке 14.1 и понаблюдайте, как она выражает то, что мы говорили выше в словах.

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! Реализовать метод `main()`, который использует ваши новые классы и интерфейс.

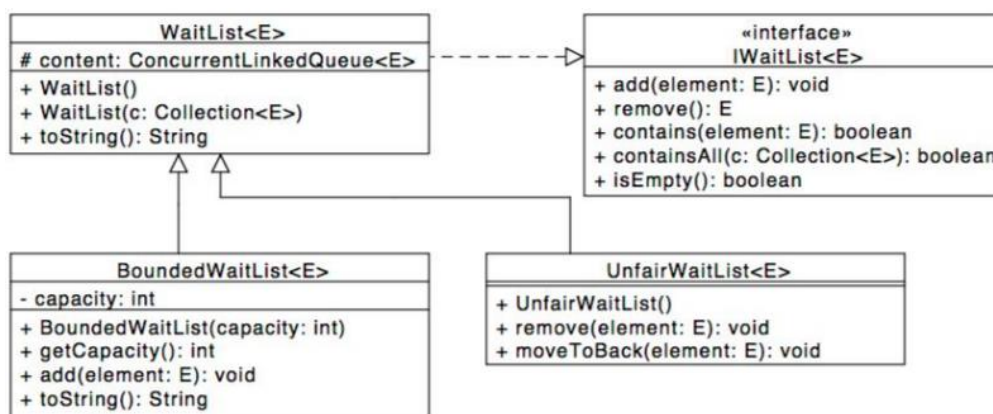


Рисунок 14.1 – UML диаграмма классов

### Решение:

Программы, реализующие работу интерфейса IWaitList (листинг 14.1), классов WaitList (листинг 14.2), BoundedWaitList (листинг 14.3), UnfairWaitList (листинг 14.4), и тестового класса Main (листинг 14.5). Результат работы программы представлен на рисунке 14.2

#### Листинг 14.1 – Код программы интерфейса IWaitList

```
import java.util.Collection;

/**
 * Interface IWaitList with methods for implementors.
 * @param <E>.
 * @author vladimirrakitin.
 * @version 1.1.
 */
public interface IWaitList<E> {

    /**
     * Method to add element into end of queue.
     * @param element element.
     */
    void add(E element);

    /**
     * Method to remove first element of queue.
     * @return Removed element.
     */
    E remove();

    /**
     * Method to check, is equal element included in queue or not.
     * @param element Value to check
     * @return True - element find, false - not find.
     */
    boolean contains(E element);

    /**
     * Method to check, list is empty or not
     * @param c Collection with values for search
     * @return True - founded all elements, false - some element was not
     found.
     */
    boolean containsAll(Collection<E> c);

    /**
     * Method to check, are there any elements in list or not.
     * @return True - no elements, false - any elements in list.
     */
    boolean isEmpty();
}
```

#### Листинг 14.2 – Код программы класса WaitList

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.concurrent.ConcurrentLinkedQueue;

/**
 * @param <E> Type of stored data.
```

```

* @author vladimirrakitin.
* @version 1.1.
*/
public class WaitList<E> implements IWaitList<E>{

    /**
     * Field for storing data.
     */
    protected ConcurrentLinkedQueue<E> content;

    /**
     * Constructor - creating new object with empty queue.
     */
    public WaitList(){
        content = new ConcurrentLinkedQueue<E>();
    }

    /**
     * Constructor - creating new object with data from some collection.
     * @param c Collection with data to store.
     */
    public WaitList(Collection<E> c){
        content = new ConcurrentLinkedQueue<E>(c);
    }

    /**
     * Method to get information about this list in String type.
     * @return String-value of this list.
     */
    public String toString(){
        return "WaitList: "+content;
    }

    /**
     * Method to add element at the end of queue.
     * @param element Value to be added.
     */
    public void add(E element){
        content.add(element);
    }

    /**
     * Method to remove first element of the queue.
     * @return Removed element.
     */
    public E remove(){
        if(!isEmpty()) {
            return content.remove();
        }
        System.out.println("Очередь пуста");
        return null;
    }

    /**
     * Method to check, is equal element included in queue or not.
     * @param element Value to check
     * @return True - element find, false - not find.
     */
    public boolean contains(E element){
        boolean flag = false;
        for(int i = 0; i < content.size(); i++){
            E e = content.remove();
            if(e.equals(element)){

```

## Продолжение листинга 14.2

```
        flag = true;
        content.add(e);
    }
}
return flag;
}

/**
 * Method to check, list is empty or not.
 * @param c Collection with values for search.
 * @return True - founded all elements, false - some element was not
found.
 */
public boolean containsAll(Collection<E> c) {
    ArrayList<E> arrayList = new ArrayList<>(c);
    for (int i = 0; i < c.size(); i++) {
        boolean flag = false;
        for (int j = 0; j < content.size(); j++) {
            E e = content.remove();
            if (e.equals(arrayList.get(i))) {
                flag = true;
                content.add(e);
            }
        }
        if (!flag) {
            return false;
        }
    }
    return true;
}

/**
 * Method to check, are there any elements in list or not.
 * @return True - no elements, false - any elements in list.
 */
public boolean isEmpty() {
    return content.isEmpty();
}
}
```

## Листинг 14.3 – Код программы класса BoundedWaitList

```
/**
 * @param <E>.
 * @author vladimirrakitin.
 * @version 1.1.
 */
public class BoundedWaitList<E> extends WaitList<E>{
    /**
     * Field for maximum capacity of the queue.
     */
    private int capacity;

    /**
     * Constructor - creating new object with empty queue and maximum
capacity.
     * @param capacity Maximum of queue.
     */
    public BoundedWaitList(int capacity){
        if(capacity > 0){
            this.capacity = capacity;
        }
    }
}
```

### Продолжение листинга 14.3

```
        else{
            System.out.println("Размер очереди должен быть больше 0");
        }
    }

    /**
     * Method to get capacity.
     * @return capacity.
     */
    public int getCapacity(){
        return capacity;
    }

    /**
     * Method to add element at the end of queue.
     * @param element Value to be added.
     */
    public void add(E element){
        if(capacity > content.size()){
            content.add(element);
        }
        else{
            System.out.println("Очередь заполнена");
        }
    }

    /**
     * Method to get information about this list in String type.
     * @return String-value of this list.
     */
    public String toString(){
        return "BoundedWaitList: "+ content;
    }
}
```

### Листинг 14.4 – Код программы класса UnfairWaitList

```
/**
 * @param <E>
 * @author vladimirrakitin.
 * @version 1.1.
 */
public class UnfairWaitList<E> extends WaitList<E>{
    /**
     * Constructor - creating new object with empty queue.
     */
    public UnfairWaitList(){
        super();
    }

    /**
     * Method to remove param element of the queue.
     * @param element Value for search and removing.
     */
    public void remove(E element) {
        for(int i = 0; i < content.size(); i = i+1){
            E e = content.remove(i);
            if(!e.equals(element) | (e.equals(element) && i == 0)){
                content.add(e);
                i++;
            }
        }
    }
}
```

## Продолжение листинга 14.4

```
}

/**
 * Method for moving element to the end of queue.
 * @param element Value to move.
 */
public void moveToBack(E element) {
    remove(element);
    content.add(element);
}
}
```

## Листинг 14.5 – Код программы тестового класса Main

```
import java.util.ArrayList;

/**
 * Main class for starting program and tests.
 */
public class Main {
    public static void main(String[] args) {
        WaitList <Integer> waitList = new WaitList<>();
        ArrayList <Integer> arrayList = new ArrayList<>();
        BoundedWaitList <Integer> boundedWaitList = new BoundedWaitList<>(5);
        UnfairWaitList <Integer> unfairWaitList = new UnfairWaitList<>();
        waitList.add(11); waitList.add(22);
        waitList.add(33); waitList.add(44);
        waitList.add(55);
        System.out.println(waitList.toString());
        System.out.println("WaitList: 33? "+waitList.contains(33));
        System.out.println("WaitList: 44? "+waitList.contains(44));
        System.out.println("WaitList: isEmpty? "+waitList.isEmpty());
        arrayList.add(0);
        System.out.println("WaitList: 0? "+waitList.containsAll(arrayList));
        System.out.println("-----");
        boundedWaitList.add(100);
        boundedWaitList.add(101);
        boundedWaitList.add(102);
        boundedWaitList.add(103);
        boundedWaitList.add(104);
        System.out.println(boundedWaitList);
        boundedWaitList.add(105);
        boundedWaitList.remove();
        System.out.println(boundedWaitList);
        boundedWaitList.add(555);
        System.out.println(boundedWaitList);
        System.out.println("Max size BoundedWaitList:
"+boundedWaitList.getCapacity());
        System.out.println("-----");
        unfairWaitList.add(10101);
        unfairWaitList.add(20202);
        unfairWaitList.add(30303);
        unfairWaitList.add(40404);
        unfairWaitList.add(50505);
        System.out.println(unfairWaitList);
        unfairWaitList.remove(20202);
        unfairWaitList.remove(30303);
        System.out.println(unfairWaitList);
        unfairWaitList.moveToBack(40404);
        System.out.println(unfairWaitList);
    }
}
```

```
WaitList: [11, 22, 33, 44, 55]
WaitList: 33? true
WaitList: 44? true
WaitList: isEmpty? false
WaitList: 0? false
-----
BoundedWaitList: [100, 101, 102, 103, 104]
Очередь заполнена
BoundedWaitList: [101, 102, 103, 104]
BoundedWaitList: [101, 102, 103, 104, 555]
Max size BoundedWaitList: 5
-----
WaitList: [10101, 20202, 30303, 40404, 50505]
WaitList: [10101, 40404, 50505]
WaitList: [10101, 50505, 40404]

Process finished with exit code 0
```

Рисунок 14.2 – Результат работы тестового класса

### **Выводы по работе:**

Во время выполнения работы были получены навыки работы со списками 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.

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

Шаблон MVC имеет три уровня:

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

### **Выполнение лабораторной работы**

#### Задание:

1. Напишите реализацию программного кода по UML диаграмме, представленной на рисунке 15.2. Программа должна демонстрировать использование паттерна MVC.



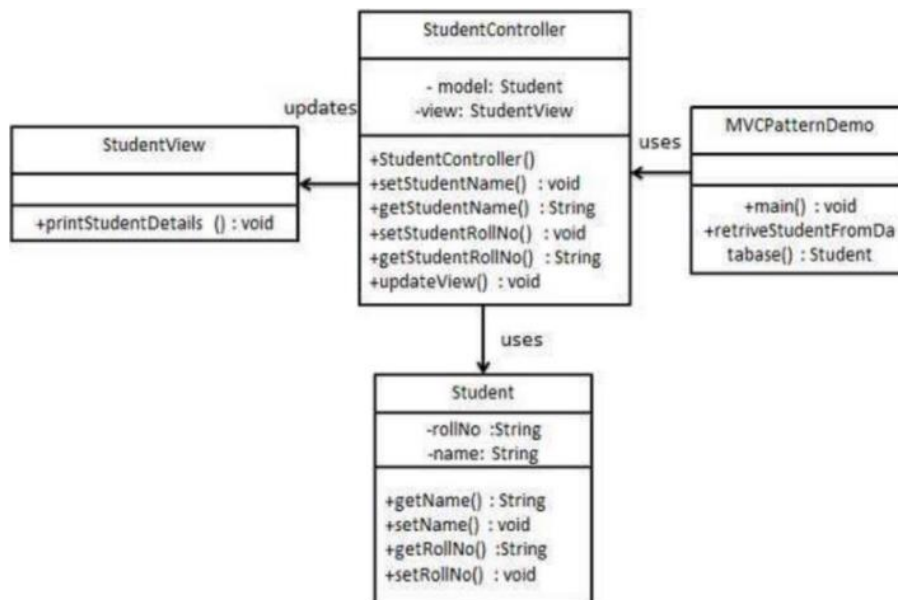


Рисунок 15.1 – UML диаграмма классов проекта, реализующего MVC

2. Напишите реализацию программного кода, с использованием паттерна MVC для расчета заработной платы сотрудника предприятия. Предлагается использовать следующие классы:

- Класс Employee – сотрудник будет выступать в качестве слоя модели
- Класс EmployeeView будет действовать как слой представления.
- Класс EmployeeContoller будет действовать как уровень контроллера.

3. Вы можете написать программную реализацию, используя собственную идею, реализуя паттерн MVC. Выполнение задания предполагает создание GUI.

### Решение:

1. Программы, реализующие работу классов Student (листинг 15.1), StudentView (листинг 15.2), StudentController (листинг 15.3) и MVCPatternDemo (листинг 15.4). Результат работы программы представлен на рисунке 15.2.

### Листинг 15.1 – Код программы класса Student

```

public class Student {
    private String rollNo;
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {

```

### Продолжение листинга 15.1

```
        this.name = name;
    }

    public String getRollNo() {
        return rollNo;
    }

    public void setRollNo(String rollNo) {
        this.rollNo = rollNo;
    }
}
```

### Листинг 15.2 – Код программы класса StudentView

```
public class StudentView {
    public void printStudentDetails(String name, String rollNo){
        System.out.println("name: "+name+", rollNo: "+rollNo);
    }
}
```

### Листинг 15.3 – Код программы класса StudentController

```
public class StudentController {
    private Student model;
    private StudentView view;
    public StudentController(Student model, StudentView view){
        this.model = model;
        this.view = view;
    }
    public void setStudentName(String name){
        model.setName(name);
    }
    public String getStudentName(){
        return model.getName();
    }
    public void setStudentRollNo(String rollNo){
        model.setRollNo(rollNo);
    }
    public String getStudentRollNo(){
        return model.getRollNo();
    }
    public void updateView(){
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

### Листинг 15.4 – Код программы класса MVCPatternDemo

```
public class MVCPatternDemo {
    public static void main(String[] args) {
        Student model = retrieveStudentFromDataBase();
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);
        controller.updateView();
        model.setRollNo("студент");
        controller.updateView();
        System.out.println(model.getName());
        model.setName("Красников");
        controller.updateView();
    }
    public static Student retrieveStudentFromDataBase(){
        Student student = new Student();
        student.setName("Комаров");
    }
}
```

## Продолжение листинга 15.4

```
        student.setRollNo("преподаватель");  
        return student;  
    }  
}
```

```
name: Комаров, rollNo: преподаватель  
name: Комаров, rollNo: студент  
Комаров  
name: Красников, rollNo: студент  
  
Process finished with exit code 0
```

Рисунок 15.2 – Результат работы класса MVCPatternDemo

2. Программы, реализующие работу классов Employee (листинг 15.5), EmployeeView (листинг 15.6), EmployeeController (листинг 15.7) и MVCPatternDemoEmployee (листинг 15.8). Результат работы программы представлен на рисунке 15.3.

## Листинг 15.5 – Код программы класса Employee

```
public class Employee {  
    private String name;  
    private int hours;  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getName() {  
        return name;  
    }  
    public int getHours() {  
        return hours;  
    }  
    public void setHours(int hours) {  
        this.hours = hours;  
    }  
}
```

## Листинг 15.6 – Код программы класса EmployeeView

```
public class EmployeeView {  
    public void printEmployeeDetails(String name, int hours) {  
        System.out.println("name: "+name+", hours: "+hours);  
    }  
}
```

## Листинг 15.7 – Код программы класса EmployeeController

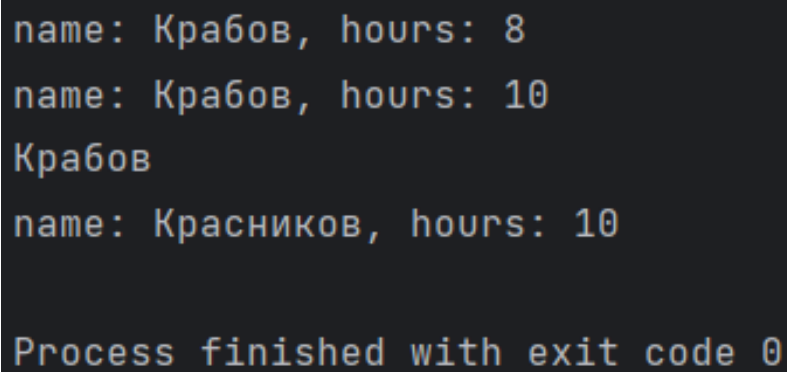
```
public class EmployeeController {  
    private Employee model;  
    private EmployeeView view;  
    public EmployeeController(Employee model, EmployeeView view) {  
        this.model = model;  
        this.view = view;  
    }  
    public void setEmployeeName(String name) {
```

## Продолжение листинга 15.7

```
        model.setName(name);
    }
    public String getEmployeeName(){
        return model.getName();
    }
    public void setEmployeeHours(int hours){
        model.setHours(hours);
    }
    public int getEmployeeHours(){
        return model.getHours();
    }
    public void updateView(){
        view.printEmployeeDetails(model.getName(), model.getHours());
    }
}
```

## Листинг 15.8 – Код программы класса MVCPatternDemoEmployee

```
public class MVCPatternDemoEmployee {
    public static void main(String[] args) {
        Employee model = retrieveEmployeeFromDataBase();
        EmployeeView view = new EmployeeView();
        EmployeeController controller = new EmployeeController(model, view);
        controller.updateView();
        model.setHours(10);
        controller.updateView();
        System.out.println(model.getName());
        model.setName("Красников");
        controller.updateView();
    }
    public static Employee retrieveEmployeeFromDataBase(){
        Employee employee = new Employee();
        employee.setName("Крабов");
        employee.setHours(8);
        return employee;
    }
}
```



```
name: Крабов, hours: 8
name: Крабов, hours: 10
Крабов
name: Красников, hours: 10

Process finished with exit code 0
```

Рисунок 15.3 – Результат работы класса MVCPatternDemoEmployee

3. Программы, реализующие работы классов Attractions (листинг 15.9), AttractionsView (листинг 15.10), AttractionsController (листинг 15.11) и MVCPatternDemoAttractions (листинг 15.12). Результат работы программы представлен на рисунках 15.4 и 15.5.

### Листинг 15.9 – Код программы класса Attractions

```
public class Attractions {
    private String name;
    private String description;

    public Attractions(String name, String description){
        this.name = name;
        this.description = description;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

### Листинг 15.10 – Код программы класса AttractionsView

```
import javax.swing.*;
import java.awt.*;

public class AttractionsView extends JFrame {
    private JLabel imageLabel;
    private JLabel titleLabel;
    private JLabel descriptionLabel;
    public AttractionsView(){
        super("Достопримечательности Москвы");

        imageLabel = new JLabel();
        titleLabel = new JLabel();
        descriptionLabel = new JLabel();

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBackground(new Color(0,200,255));
        panel.add(Box.createVerticalGlue());

        JPanel imagePanel = new JPanel();
        imagePanel.setPreferredSize(new Dimension(800, 600));
        imagePanel.setMaximumSize(new Dimension(800, 600));
        imagePanel.setBackground(new Color(0, 200, 255));
        imagePanel.add(imageLabel);

        panel.add(imagePanel);
        imagePanel.setAlignmentX(Component.CENTER_ALIGNMENT);

        panel.add(titleLabel);
        panel.add(Box.createRigidArea(new Dimension(0, 20)));

        titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        descriptionLabel.setAlignmentX(Component.CENTER_ALIGNMENT);

        panel.add(descriptionLabel);
    }
}
```

## Продолжение листинга 15.10

```
        descriptionLabel.setMaximumSize(new Dimension(1000, 250));

        panel.add(Box.createVerticalGlue());

        getContentPane().add(panel);
        setSize(800, 800);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void updateView(String imagePath, String title, String
description){
        imageLabel.setIcon(new ImageIcon(imagePath));
        titleLabel.setText(title);
        descriptionLabel.setText("<html><body style='width: 100%;'>" +
description + "</body></html>");
    }
}
```

## Листинг 15.11 – Код программы класса AttractionsController

```
public class AttractionsController {
    private Attractions model;
    private AttractionsView view;
    public AttractionsController(Attractions model, AttractionsView view) {
        this.model = model;
        this.view = view;
    }
    public void updateAttractions(String imagePath, String title, String
description){
        model.setName(title);
        model.setDescription(description);
        view.updateView(imagePath, title, description);
    }
}
```

## Листинг 15.12 – Код программы класса MVCPatternDemoAttractions

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class MVCPatternDemoAttractions {
    public static void main(String[] args) {
        Attractions model = new Attractions("Московский кремль", "Московский
Кремль - крепость в центре Москвы и древнейшая её часть, " +
        "главный общественно-политический и историко-художественный
комплекс города, официальная резиденция Президента Российской Федерации,
вплоть " +
        "до распада СССР в декабре 1991 года была официальной
резиденцией Генерального секретаря ЦК КПСС (в 1990-1991 годах - Президента
СССР). " +
        "Одно из самых известных архитектурных сооружений в мире.");
        AttractionsView view = new AttractionsView();
        AttractionsController controller = new AttractionsController(model,
view);

        controller.updateAttractions("src/picture0.jpg", model.getName(),
model.getDescription());

        view.addMouseListener(new MouseAdapter() {
            int click = 1;
            @Override
            public void mouseClicked(MouseEvent e) {
```

## Продолжение листинга 15.12

```
switch (click){
    case 0:
        controller.updateAttractions("src/picture0.jpg",
"Московский Кремль", "Московский Кремль – крепость " +
        "в центре Москвы и древнейшая её часть,
главный общественно-политический и историко-художественный комплекс города,
" +
        "официальная резиденция Президента
Российской Федерации, вплоть до распада СССР в декабре 1991 года была
официальной " +
        "резиденцией Генерального секретаря ЦК КПСС
(в 1990–1991 годах – Президента СССР). Одно из самых известных архитектурных
" +
        "сооружений в мире.");
        break;
    case 1:
        controller.updateAttractions("src/picture1.jpg",
"Третьяковская галерея", "Государственная Третьяковская галерея –" +
        "московский художественный музей,
основанный в 1856 году купцом Павлом Третьяковым. В 1867-м галерея была
открыта для посещения, а " +
        "в 1892 году передана в собственность
Москве. На момент передачи коллекция музея насчитывала 1276 картин, 471
рисунок, десять скульптур " +
        "русских художников, а также 84 картины
иностранных мастеров.");
        break;
    case 2:
        controller.updateAttractions("src/picture2.jpg",
"Храм Христа Спасителя", "Храм Христа Спасителя – кафедральный собор " +
        "и один из главных православных соборов в
России. Достопримечательность была возведена в честь победы и память о
погибших в Отечественной войне " +
        "1812 года. Огромный храмовый комплекс
включает в себя верхний храм с тремя престолами, нижний храм и стилобатную
часть, где располагается музей. " +
        "Святыня построена в византийском стиле.");
        break;
    case 3:
        controller.updateAttractions("src/picture3.jpg",
"Большой театр", "Большой театр – главный храм российской культуры,
расположенный" +
        "в самом центре Москвы на Театральной
площади. Он является одним из наиболее значимых в мире театров оперы и
балета. Сегодня Большой театр по праву " +
        "называют гордостью русской культуры. На
протяжении всей своей истории он служил центром притяжения лучших сил
русского оперного и балетного искусства.");
        break;
    case 4:
        controller.updateAttractions("src/picture4.jpg",
"Останкинская телебашня", "Останкинская телебашня – телевизионная и
радиовещательная " +
        "башня, расположенная в Останкинском районе
Москвы. Высота здания – 540,1 м. В телебашне находится двухэтажный
концертный зал Останкино. Современный " +
        "комплекс оснащён новейшим световым, видео –
и аудио-оборудованием. Зал вмещает 750 зрителей, а используют его для
проведения конференций, презентаций, " +
        "семинаров, концертов, театральных
постановок, телесъёмок и других масштабных мероприятий.");
        break;
```

## Продолжение листинга 15.12

```
        case 5:
            controller.updateAttractions("src/picture5.png",
"Могила неизвестного солдата", "Могила Неизвестного Солдата – мемориальный
архитектурный " +
                "ансамбль, расположенный у Кремлёвской стены
в Александровском саду. Монумент был построен в 1967 году в память о
безвестных российских и советских воинах, " +
                "которые погибли в боевых действиях на
территории нашей страны или за ее пределами. ");
            break;
        case 6:
            controller.updateAttractions("src/picture6.jpg",
"Московский государственный университет имени М.В.Ломоносова", "Московский
государственный " +
                "университет имени М.В.Ломоносова (МГУ) –
старейший и крупнейший вуз России, один из центров мировой науки. Основан по
замыслу и плану М.В.Ломоносова указом " +
                "императрицы Елизаветы Петровны в 1755 году.
Высота здания – 182 метра. Со шпилем – 240 метров. Главное здание МГУ –
самое высокое из сталинских небоскрёбов. " +
                "Высотка на Ленинских горах строилась как
одна из наиболее важных архитектурных доминант столицы.");
            break;
        case 7:
            controller.updateAttractions("src/picture7.jpg",
"Памятник Минину и Пожарскому", "Памятник Минину и Пожарскому – бронзовая
статуя Ивана Мартоса, " +
                "расположенная на Красной площади в Москве,
перед собором Василия Блаженного. Статуя увековечивает память князя Дмитрия
Пожарского и Кузьмы Минина, которые собрали " +
                "всероссийскую добровольческую армию и
изгнали из Москвы войска Речи Посполитой под командованием польского короля
Сигизмунда III, положив тем самым конец Смутному времени в 1612 году.");
            break;
        case 8:
            controller.updateAttractions("src/picture8.jpg",
"Мавзолей Ленина", "Мавзолей В.И. Ленина – памятник-усыпальница у
Кремлевской стены на Красной площади Москвы, в " +
                "траурном зале которого установлен саркофаг
с бальзамированным телом основателя советского социалистического государства
в России Владимира Ульянова-Ленина. ");
            break;
        case 9:
            controller.updateAttractions("src/picture9.jpg",
"Царь-колокол", "Царь-колокол в Кремле – уникальное произведение литейного
искусства. Это единственный колокол, " +
                "не издавший за все время своего
существования ни единого звука. Его высота чуть больше шести метров, а
диаметр – шесть с половиной метров. Еще он невероятно тяжелый – его вес
целых двести " +
                "тонн. Знаменитый Царь-колокол
предназначался для Ивановской звонницы.");
            break;
    }
    click++;
    if(click == 10){
        click = 0;
    }
});
}
```



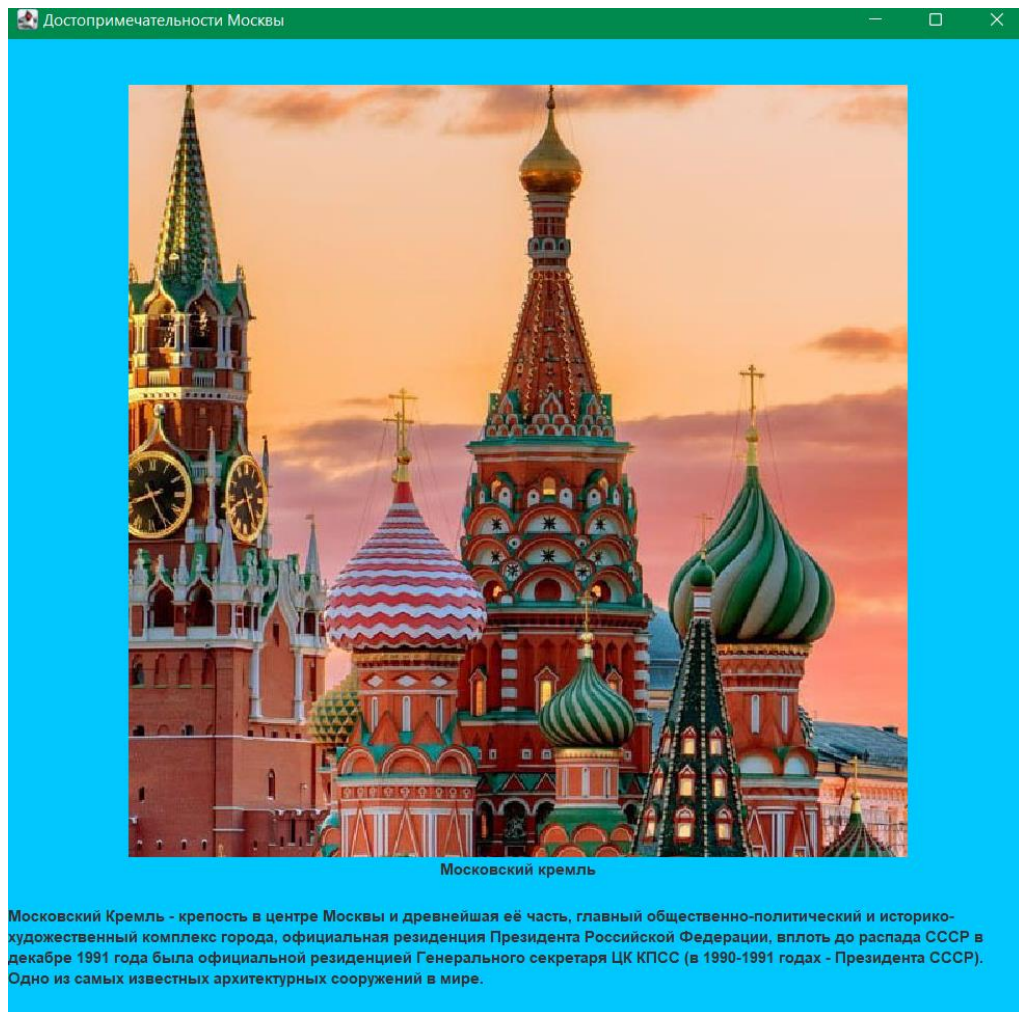


Рисунок 15.4 – Результат работы класса MVCPatternDemoAttractions

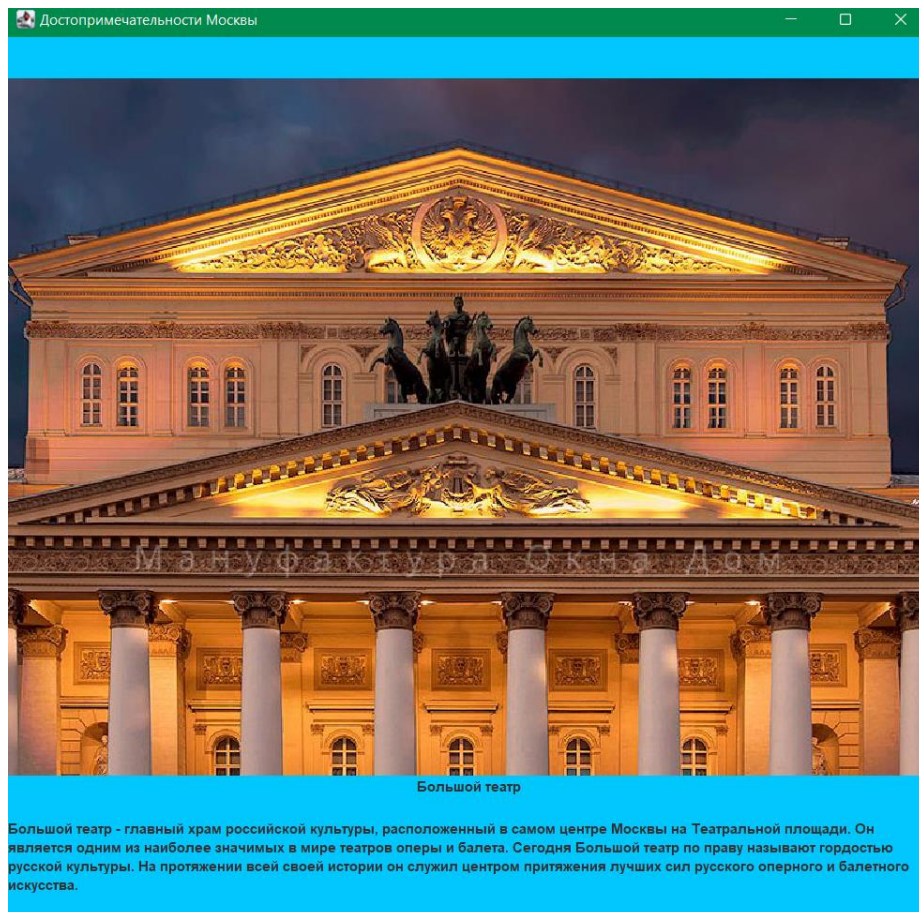


Рисунок 15.5 – Результат работы класса MVCPatternDemoAttractions

### Выводы по работе:

Во время выполнения работы были получены навыки реализации программ на языке Джава с применением паттерн MVC.

## Практическая работа №16

### Цель работы

Получить практические навыки разработки программ, изучить синтаксис языка Java, освоить основные конструкции языка Java (циклы, условия, создание переменных и массивов, создание методов, вызов методов), а также научиться осуществлять стандартный ввод/вывод данных.

### Теоретическое введение

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.

То, что исключения являются объектами важно по двум причинам:

- они образуют иерархию с корнем java.lang.Throwable (java.lang.Object – предок java.lang.Throwable, но Object – это не исключение!);
- они могут иметь поля и методы.

catch – полиморфная конструкция, т.е. catch по типу класса родителя перехватывает исключения для экземпляров объектов как родительского класса, так или его наследников (т.е. экземпляры непосредственно самого родительского класса или любого его потомка).

### Выполнение лабораторной работы

#### Задание:

1. Выполните программу (листинг 16.1) и посмотрите, что происходит. 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 (листинг 16.2). Запустите программу и обратите внимание на новое поведение. Объясните поведение программы.

Листинг 16.1 – Пример обработки деления на ноль

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

## 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");  
        }  
    }  
}
```

2. Запустите программу (листинг 16.3) со следующими выводами:  
Qwerty 0 1.2 1. Посмотрите на вывод. Объясните какие исключения выбрасываются? Измените код, добавив блоки try-catch, чтобы иметь дело с определяемыми исключениями. Объясните поведение программы.

## Листинг 16.3 – Пример программы

```
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 );  
    }  
}
```

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. Выполните это действие, чтобы поймать общее исключение класса Exception. Перезапустите программу с приведенными выше данными и обратите внимание на ее поведение.

4. Добавьте блок finally к решению Задания №2. Повторно запустите программу, чтобы проверить ее поведение. Объясните новое поведение программы.

5. Откомпилируйте класс (листинг 16.4), создайте его экземпляр и выполните метод getDetails() с нулем в качестве значения параметра. Добавьте блок try-catch таким образом, чтобы перехватить исключение и рассмотреть его внутри метода. Объясните поведение программы.

## Листинг 16.4 – Пример программы класса ThrowsDemo

```
public class ThrowsDemo {  
    public void printMessage(String key) {
```

## Продолжение листинга 16.4

```
String message = getDetails(key);
System.out.println( message );
}
public String getDetails(String key) {
    if(key == null){
        throw new NullPointerException( "null key in getDetails" );
    }
    return "data for" + key;
}
}
```

6. Откомпилируйте и запустите эту программу с правильным значением для ключа и с нулем в качестве значения. При выполнении с нулевым значением вы увидите некоторый вывод. Обобщите все вышесказанное и выполните эту программу с правильным значением для ключа и с нулем в ключе. Теперь добавьте блоки try-catch, чтобы использовать для вывода сообщений метод `printMessage()`, таким образом, чтобы исключения обрабатывались и программа не “ломалась”. Объясните ее поведение.

7. Создайте следующий класс (листинг 16.5) и попытайтесь его скомпилировать. В результате успешного пробрасывания исключение должно быть поймано или объявлено. Объясните причину.

## Листинг 16.5 – Пример программы

```
public class ThrowsDemo {
    public void getKey() {
        Scanner myScanner = new Scanner( System.in );
        String key = myScanner.next();
        printDetails( key );
    }
    public void printDetails(String key) {
        try { String message = getDetails(key);
            System.out.println( message );
        }
        catch ( Exception e){
            throw e;
        }
    }
    private String getDetails(String key) {
        if(key == "") {
            throw new Exception( "Key set to empty string" );
        }
        return "data for " + key;
    }
}
```

8. Измените код из предыдущего примера следующим образом: удалите `throws Exception` из метода `getKey()`. Измените метод `getKey()`,

добавив try-catch блок для обработки исключений. Добавьте цикл к getKey() таким образом, чтобы пользователь получил еще один шанс на ввод значение ключа.

Решение:

1. Программа, реализующая работу класса Exception1, представлена в листинге 16.6. Результаты работы программы представлены на рисунках 16.1 – 16.3.

Листинг 16.6 – Код программы класса Exception1

```
import java.util.Scanner;

public class Exception1 {
    public static void exceptionDemo1() {
        System.out.println(2/0);
    }
    public static void exceptionDemo2() {
        System.out.println(2.0/0.0);
    }
    public static void exceptionDemo3() {
        try{
            System.out.println( 2/0 );
        } catch ( ArithmeticException e ) {
            System.out.println("Attempted division by zero");
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                exceptionDemo1();
                break;
            case 2:
                exceptionDemo2();
                break;
            case 3:
                exceptionDemo3();
                break;
        }
    }
}
```

```
1
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Exception1.exceptionDemo1(Exception1.java:5)
    at Exception1.main(Exception1.java:23)

Process finished with exit code 1
```

Рисунок 16.1 – Результат работы программы

```
2
Infinity

Process finished with exit code 0
```

Рисунок 16.2 – Результат работы программы

```
3
Attempted division by zero

Process finished with exit code 0
```

Рисунок 16.3 – Результат работы программы

2. Программа, реализующая работу класса Exception2, представлена в листинге 16.7. Результаты работы программы представлены на рисунках 16.4 – 16.7.

Листинг 16.7 – Код программы класса Exception2

```
import java.util.Scanner;

public class Exception2 {
    public static void exceptionDemo1() {
        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 static void exceptionDemo2() {
        try {
            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 );
        }
        catch (NumberFormatException e) {
            System.out.println("Это не число");
        }
        catch (ArithmeticException e) {
            System.out.println("Делить на 0 нельзя");
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                exceptionDemo1();
                break;
            case 2:
                exceptionDemo2();
        }
    }
}
```

## Продолжение листинга 16.7

```
        break;  
    }  
}  
}
```

```
1  
Enter an integer qwerty  
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "qwerty"  
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.base/java.lang.Integer.parseInt(Integer.java:652)  
    at java.base/java.lang.Integer.parseInt(Integer.java:770)  
    at Exception2.exceptionDemo1(Exception2.java:8)  
    at Exception2.main(Exception2.java:31)  
  
Process finished with exit code 1
```

Рисунок 16.4 – Результат работы программы

```
1  
Enter an integer 0  
Exception in thread "main" java.lang.ArithmeticException: Create breakpoint : / by zero  
    at Exception2.exceptionDemo1(Exception2.java:9)  
    at Exception2.main(Exception2.java:31)  
  
Process finished with exit code 1
```

Рисунок 16.5 – Результат работы программы

```
2  
Enter an integer qwerty  
Это не число  
  
Process finished with exit code 0
```

Рисунок 16.6 – Результат работы программы

```
2  
Enter an integer 0  
Делить на 0 нельзя  
  
Process finished with exit code 0
```

Рисунок 16.7 – Результат работы программы



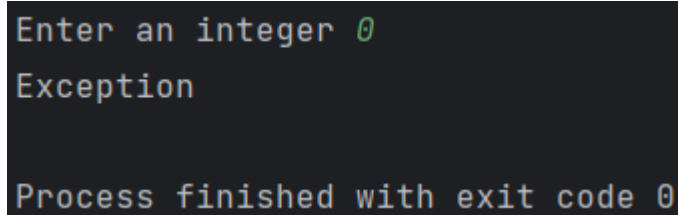
3. Программа, реализующая работу класса Exception, представлена в листинге 16.8. Результаты работы программы представлены на рисунках 16.8 – 16.9.

#### Листинг 16.8 – Код программы класса Exception3

```
import java.util.Scanner;

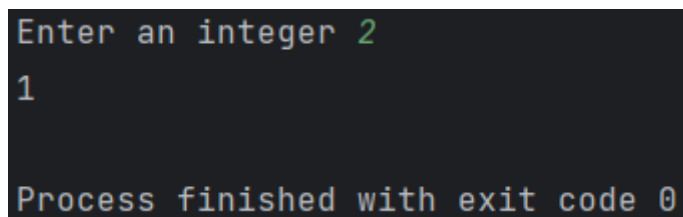
public class Exception3 {
    public static void exceptionDemo() {
        try {
            Scanner myScanner = new Scanner(System.in);
            System.out.print("Enter an integer ");
            String intString = myScanner.next();
            int i = Integer.parseInt(intString);
            System.out.println( 2/i );
        }
        catch (Exception e){
            System.out.println("Exception");
        }
    }

    public static void main(String[] args) {
        exceptionDemo();
    }
}
```



Enter an integer 0  
Exception  
Process finished with exit code 0

Рисунок 16.8 – Результат работы программы



Enter an integer 2  
1  
Process finished with exit code 0

Рисунок 16.9 – Результат работы программы

4. Программа, реализующая работу класса Exception4, представлена в листинге 16.9. Результаты работы программы представлены на рисунках 16.10 – 16.11.

#### Листинг 16.9 – Код программы класса Exception4

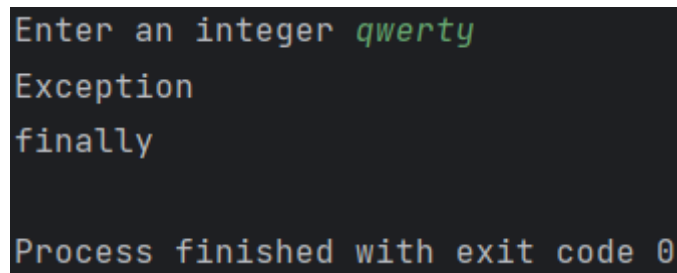
```
import java.util.Scanner;

public class Exception4 {
    public static void exceptionDemo() {
        try {
            Scanner myScanner = new Scanner(System.in);
```

## Продолжение листинга 16.9

```
        System.out.print("Enter an integer ");
        String intString = myScanner.next();
        int i = Integer.parseInt(intString);
        System.out.println( 2/i );
    }
    catch (Exception e){
        System.out.println("Exception");
    }
    finally {
        System.out.println("finally");
    }
}

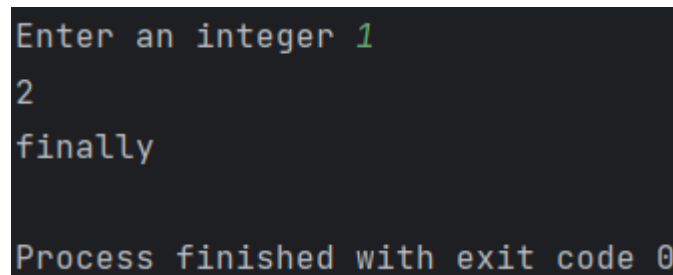
public static void main(String[] args) {
    exceptionDemo();
}
}
```



```
Enter an integer qwerty
Exception
finally

Process finished with exit code 0
```

Рисунок 16.10 – Результат работы программы



```
Enter an integer 1
2
finally

Process finished with exit code 0
```

Рисунок 16.11 – Результат работы программы

5. Программа, реализующая работу класса Exception5, представлена в листинге 16.10. Результаты работы программы представлены на рисунках 16.12 – 16.13.

## Листинг 16.10 – Код программы класса Exception5

```
import java.util.Scanner;

public class Exception5 {
    public static void getDetails1(String key) {
        if(key == null) {
            throw new NullPointerException("null key in getDetails" );
        }
        // do something with the key
    }
    public String getDetails2(String key) {
        try {
```

## Продолжение листинга 16.10

```
        if (key == null) {
            throw new NullPointerException("null key in getDetails");
        }

    } catch (NullPointerException e) {}
    return "data for " + key;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int choice = scanner.nextInt();
    switch (choice) {
        case 1:
            Exception5 exception51 = new Exception5();
            exception51.getDetails1(null);
            break;
        case 2:
            Exception5 exception52 = new Exception5();
            System.out.println(exception52.getDetails2(null));
            break;
    }
}
```

```
1
Exception in thread "main" java.lang.NullPointerException Create breakpoint : null key in getDetails
    at Exception5.getDetails1(Exception5.java:6)
    at Exception5.main(Exception5.java:27)

Process finished with exit code 1
```

Рисунок 16.12 – Результат работы программы

```
2
data for null

Process finished with exit code 0
```

Рисунок 16.13 – Результат работы программы

6. Программа, реализующая работу класса Exception6, представлена в листинге 16.11. Результаты работы программы представлены на рисунках 16.14 – 16.15.

## Листинг 16.11 – Код программы класса Exception6

```
import java.util.Scanner;

public class Exception6 {
    public void printMessage(String key) {
        try {
            key = getDetails(key);
        }
        catch (Exception e) {}
    }
}
```

## Продолжение листинга 16.11

```
        System.out.println(key);
    }
    public String getDetails(String key) {
        if(key == null) {
            throw new NullPointerException( "null key in getDetails" );
        }
        return "data for " + key;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                Exception6 exception61 = new Exception6();
                exception61.getDetails("qqqqq");
                exception61.getDetails(null);
                break;
            case 2:
                Exception6 exception62 = new Exception6();
                exception62.printMessage(null);
                exception62.printMessage("qqqqq");
                break;
        }
    }
}
```

```
1
Exception in thread "main" java.lang.NullPointerException Create breakpoint : null key in getDetails
    at Exception6.getDetails(Exception6.java:13)
    at Exception6.main(Exception6.java:25)

Process finished with exit code 1
```

Рисунок 16.14 – Результат работы программы

```
2
null
data for qqqqq

Process finished with exit code 0
```

Рисунок 16.15 – Результат работы программы

7. Программа, реализующая работу класса Exception7, представлена в листинге 16.12. Результаты работы программы представлены на рисунках 16.16 – 16.17.

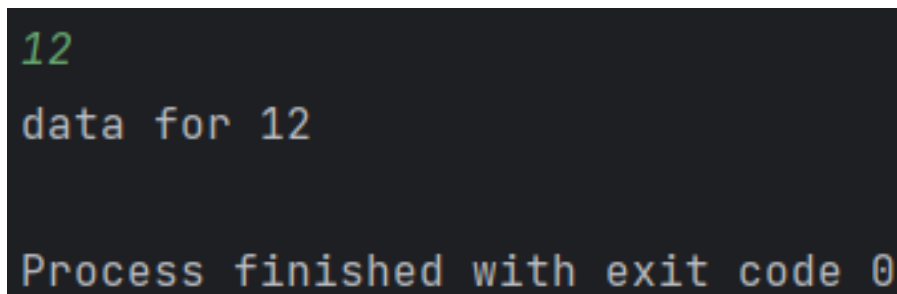
## Листинг 16.12 – Код программы класса Exception7

```
import java.util.Objects;
import java.util.Scanner;

public class Exception7 {
```

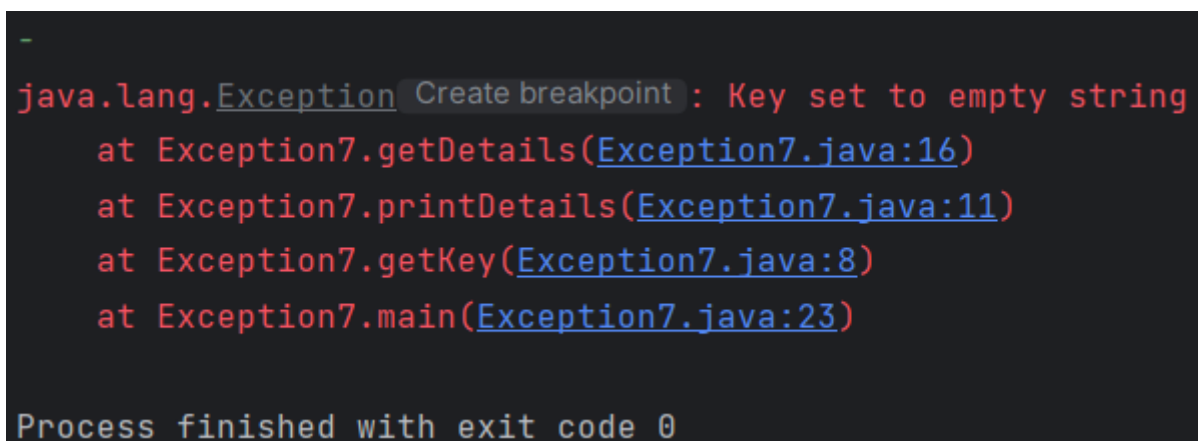
## Продолжение листинга 16.12

```
public void getKey() throws Exception {
    Scanner myScanner = new Scanner( System.in );
    String key = myScanner.next();
    printDetails(key);
}
public void printDetails(String key) throws Exception {
    String message = getDetails(key);
    System.out.println( message );
}
private String getDetails(String key) throws Exception {
    if(Objects.equals(key, "-")) {
        throw new Exception( "Key set to empty string" );
    }
    return "data for " + key;
}
public static void main(String[] args) {
    Exception7 exception7 = new Exception7();
    try {
        exception7.getKey();
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```



```
12
data for 12
Process finished with exit code 0
```

Рисунок 16.16 – Результат работы программы



```
-
java.lang.Exception Create breakpoint : Key set to empty string
    at Exception7.getDetails(Exception7.java:16)
    at Exception7.printDetails(Exception7.java:11)
    at Exception7.getKey(Exception7.java:8)
    at Exception7.main(Exception7.java:23)
Process finished with exit code 0
```

Рисунок 16.17 – Результат работы программы

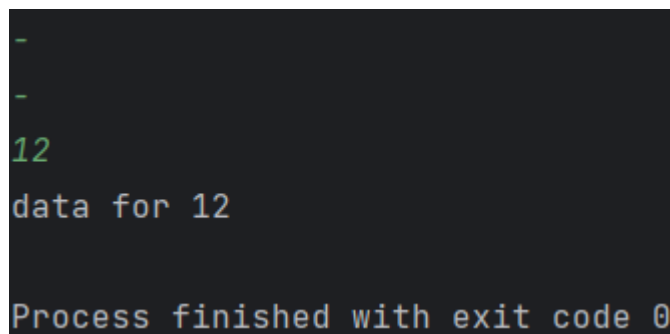
8. Программа, реализующая работу класса Exception8, представлена в листинге 16.13. Результат работы программы представлен на рисунке 16.18.

### Листинг 16.13 – Код программы класса Exception8

```
import java.util.Objects;
import java.util.Scanner;

public class Exception8 {
    public void getKey() {
        while(true){
            try {
                Scanner myScanner = new Scanner( System.in );
                String key = myScanner.next();
                printDetails(key);
                return;
            }
            catch (Exception e){}
        }
    }
    public void printDetails(String key) throws Exception {
        String message = getDetails(key);
        System.out.println( message );
    }
    private String getDetails(String key) throws Exception {
        if(Objects.equals(key, "-")) {
            throw new Exception( "Key set to empty string" );
        }
        return "data for " + key;
    }
}

public static void main(String[] args) {
    Exception8 exception8 = new Exception8();
    try {
        exception8.getKey();
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```



```
-
-
12
data for 12

Process finished with exit code 0
```

Рисунок 16.18 – Результат работы программы

#### Выводы по работе:

Во время выполнения работы были получены навыки применения исключений в реализации программ на языке Джава.

## **Практическая работа №17**

### **Цель 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.

Чтобы создать собственное пользовательское исключение, мы будем наследоваться от класса `java.lang.Exception`.

Чтобы создать собственное непроверяемое исключение, нам нужно наследоваться от класса `java.lang.RuntimeException`.

### **Выполнение лабораторной работы**

#### Задание:

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

2. Предлагается модернизировать задачу из предыдущей практической работы (см. методические указания по выполнению практических работ №1-8) – задача сортировки студентов по среднему баллу. Необходимо разработать пользовательский интерфейс для задачи поиска и сортировки (использовать массив интерфейсных ссылок- пример в лекции 5). Дополнить ее поиском студента по фιο – в случае отсутствия такого студента необходимо выдавать собственное исключение. Схема классов программы приведена на рисунке 17.1.

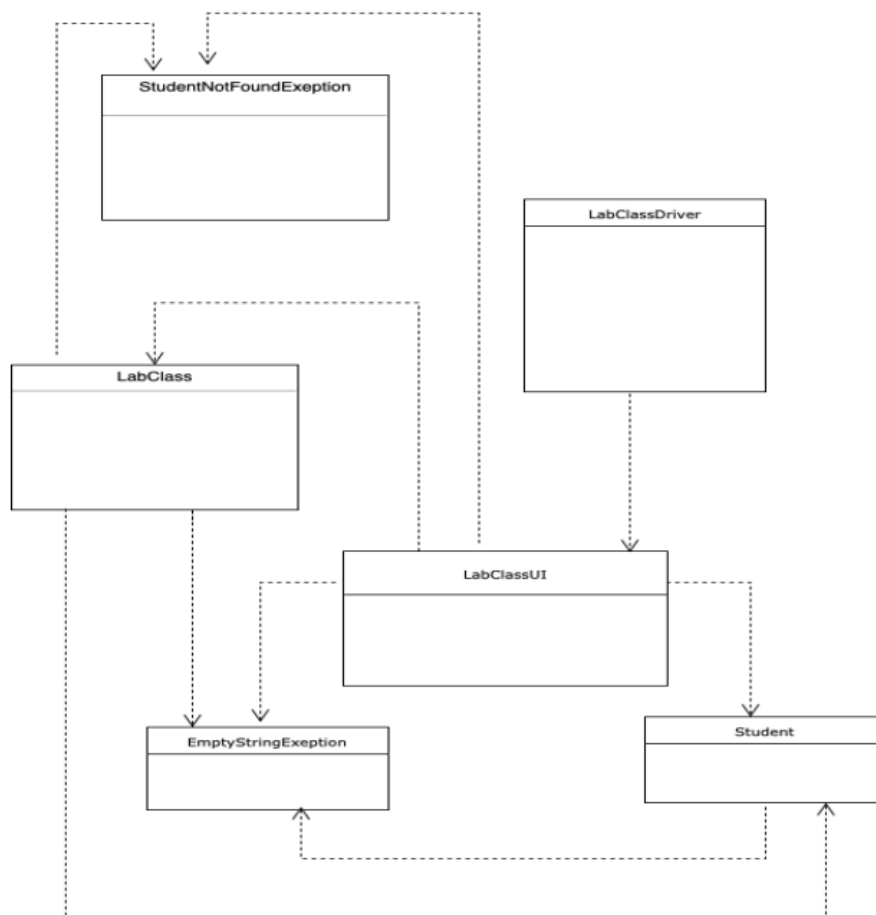


Рисунок 17.1 – UML диаграмма проекта LabClass

### Решение:

1. Программы, реализующие работу классов INNException (листинг 17.1) и MainINN (листинг 17.2.). Результаты работы программы представлены на рисунке 17.2 – 17.4.

### Листинг 17.1 – Код программы класса INNException

```

public class INNException extends Exception {
    public INNException(String message) {
        super(message);
    }
}

```

### Листинг 17.2 – Код программы класса MainINN

```

import java.util.Scanner;

public class MainINN {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите ФИО");
        String fio = scanner.nextLine();
        System.out.println("Введите ИНН");
        String inn = scanner.nextLine();

        try {
            validateINN(inn);
        }
    }
}

```

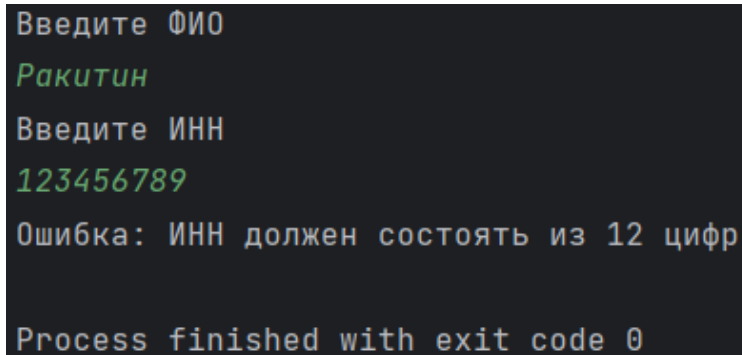


## Продолжение листинга 17.2

```
        System.out.println("Заказ оформлен успешно");
    } catch (INNException e) {
        System.out.println("Ошибка: " + e.getMessage());
    }
}

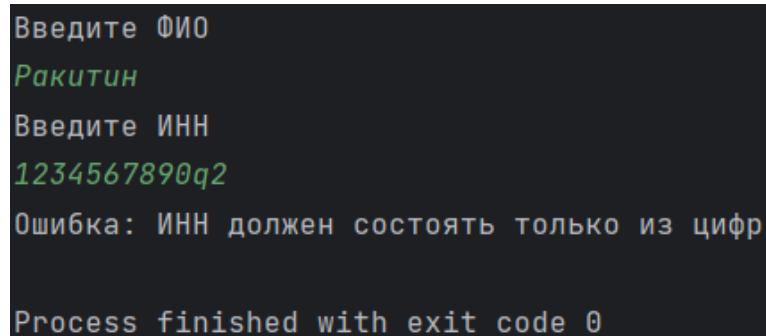
public static void validateINN(String inn) throws INNException {
    if (inn.length() != 12) {
        throw new INNException("ИНН должен состоять из 12 цифр");
    }

    try {
        Long.parseLong(inn);
    } catch (NumberFormatException e) {
        throw new INNException("ИНН должен состоять только из цифр");
    }
}
}
```



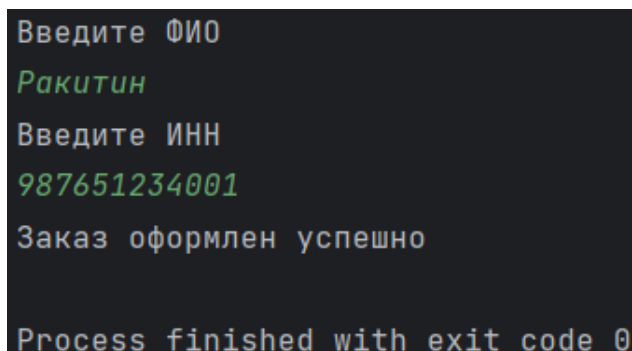
```
Введите ФИО
Ракитин
Введите ИНН
123456789
Ошибка: ИНН должен состоять из 12 цифр
Process finished with exit code 0
```

Рисунок 17.2 – Результат работы класса MainINN



```
Введите ФИО
Ракитин
Введите ИНН
1234567890q2
Ошибка: ИНН должен состоять только из цифр
Process finished with exit code 0
```

Рисунок 17.3 – Результат работы класса MainINN



```
Введите ФИО
Ракитин
Введите ИНН
987651234001
Заказ оформлен успешно
Process finished with exit code 0
```

Рисунок 17.4 – Результат работы класса MainINN

2. Программы, реализующие работу классов Student (листинг 17.3), StudentNotFoundException (листинг 17.4), EmptyStringException (листинг 17.5), LabClass (листинг 17.6), LabClassDriver (листинг 17.7), LabClassGUI (листинг 17.8). Результаты работы программы представлены на рисунках 17.5-17.6.

#### Листинг 17.3 – Код программы класса Student

```
public class Student {
    private String surname;
    private int id;
    private double GPA;
    public Student(String surname, int id, double GPA) {
        this.surname = surname;
        this.id = id;
        this.GPA = GPA;
    }
    public double getGPA() {
        return GPA;
    }
    public void setGPA(double GPA) {
        this.GPA = GPA;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getSurname() {
        return surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }

    @Override
    public String toString() {
        return "surname - "+getSurname()+" , ID - "+getId()+" , GPA = "+getGPA();
    }
}
```

#### Листинг 17.4 – Код программы класса StudentNotFoundException

```
import javax.swing.*;

public class StudentNotFoundException extends Exception{
    public StudentNotFoundException(String fio, LabClassGUI gui){
        JOptionPane.showMessageDialog(gui, "Студент "+fio+" не найден!");
    }
}
```

#### Листинг 17.5 – Код программы класса EmptyStringException

```
import javax.swing.*;

public class EmptyStringException extends IllegalArgumentException{
    public EmptyStringException(LabClassGUI gui){
        JOptionPane.showMessageDialog(gui, "Пустая строка!");
    }
}
```

### Листинг 17.6 – Код программы класса LabClass

```
import java.util.ArrayList;

public class LabClass {
    public static void main(String[] args) {
        ArrayList<Student> student = new ArrayList<>();
        student.add(new Student("Комаров П.В.", 11111, 4.4));
        new LabClassGUI(student);
    }
}
```

### Листинг 17.7 – Код программы класса LabClassDriver

```
import java.util.Comparator;

public class LabClassDriver implements Comparator<Student> {

    @Override
    public int compare(Student s1, Student s2) {
        return Double.compare(s1.getGPA(), s2.getGPA());
    }
    public void QuickSort(Student[] arr, int begin, int end){
        if (begin < end){
            int partindex = part(arr, begin, end);
            QuickSort(arr, begin, partindex-1);
            QuickSort(arr, partindex+1, end);
        }
    }
    public int part(Student[] arr, int begin, int end){
        Student pt = arr[end];
        int i = begin-1;
        for (int j = begin; j < end; j++) {
            if (compare(arr[j], pt) >= 0) {
                i++;
                Student s = arr[i];
                arr[i] = arr[j];
                arr[j] = s;
            }
        }
        Student s = arr[i+1];
        arr[i+1] = arr[end];
        arr[end] = s;
        return i+1;
    }
}
```

### Листинг 17.7 – Код программы класса LabClassGUI

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Objects;

public class LabClassGUI extends JFrame {
    private ArrayList<Student> studentArrayList;
    private JTable table;
    private DefaultTableModel tableModel;
    public LabClassGUI(ArrayList<Student> studentArrayList) {
        this.studentArrayList = studentArrayList;
        setTitle("Студенты");
        JButton addButton = new JButton("Добавить");
    }
}
```

```

JButton delButton = new JButton("Удалить");
JButton GPAButton = new JButton("Отсортировать по GPA");
JButton searchButton = new JButton("Найти");
JPanel panel = new JPanel(new FlowLayout());
setSize(500, 500);
panel.add(addButton);
panel.add(delButton);
panel.add(GPAButton);
panel.add(searchButton);
setDefaultCloseOperation(EXIT_ON_CLOSE);
add(panel, BorderLayout.NORTH);
setVisible(true);

Object [][] tableStudent = new Object[studentArrayList.size()][3];
for(int i = 0; i < studentArrayList.size(); i++){
    tableStudent[i][0] = studentArrayList.get(i).getSurname();
    tableStudent[i][1] = studentArrayList.get(i).getId();
    tableStudent[i][2] = studentArrayList.get(i).getGPA();
}
table = new JTable(new DefaultTableModel(tableStudent, new String[]
{"ФИО", "ID", "GPA"}));
JScrollPane scrollPane = new JScrollPane(table);
add(scrollPane, BorderLayout.CENTER);
updateTable();

addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        addStudent();
    }
});
delButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        deleteStudent();
    }
});
GPAButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        sortingStudentByGPA();
    }
});
searchButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        searchStudentBySurname();
    }
});
}
private void addStudent(){
    String surname = JOptionPane.showInputDialog("ФИО");
    if(surname.isEmpty()){
        throw new EmptyStringException(this);
    }
    String id = JOptionPane.showInputDialog("ID");
    if(id.isEmpty()){
        throw new EmptyStringException(this);
    }
    String gpa = JOptionPane.showInputDialog("GPA");
    if(gpa.isEmpty()){
        throw new EmptyStringException(this);
    }
}

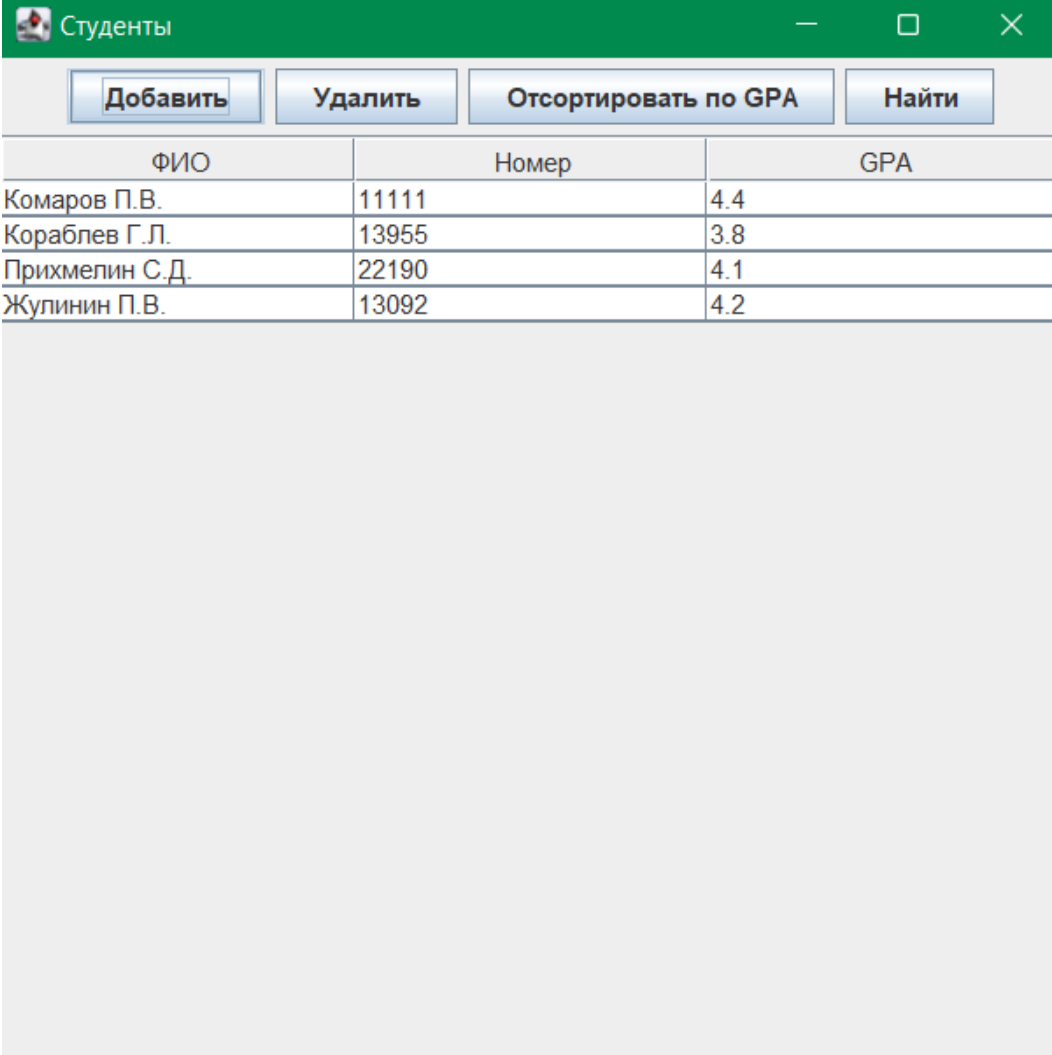
```

## Продолжение листинга 17.7

```
        Student addStudent = new Student(surname, Integer.parseInt(id),
Double.parseDouble(gpa));
        studentArrayList.add(addStudent);
        updateTable();
    }
    private void deleteStudent(){
        String surname = JOptionPane.showInputDialog("Введите ФИО
студента");
        if(surname.isEmpty()){
            throw new EmptyStringException(this);
        }
        for(int i = 0; i < studentArrayList.size(); i++){
            if(Objects.equals(studentArrayList.get(i).getSurname(),
surname)){
                studentArrayList.remove(i);
                updateTable();
                return;
            }
        }
        new StudentNotFoundException(surname, this);
    }
    private void sortingStudentByGPA(){
        studentArrayList.sort((s1, s2) ->Double.compare(s2.getGPA(),
s1.getGPA()));
        updateTable();
    }
    private void searchStudentBySurname(){
        String surname = JOptionPane.showInputDialog("Введите ФИО
студента");
        if(surname.isEmpty()){
            throw new EmptyStringException(this);
        }
        for(int i = 0; i < studentArrayList.size(); i++){
            if(Objects.equals(studentArrayList.get(i).getSurname(),
surname)){
                Student student = studentArrayList.get(i);
                Object[][] tableStudent = new Object[1][3];
                tableStudent[0][0] = student.getSurname();
                tableStudent[0][1] = student.getId();
                tableStudent[0][2] = student.getGPA();
                String[] names = {"ФИО", "ID", "GPA"};
                if(tableModel == null){
                    tableModel = new DefaultTableModel(tableStudent, names);
                    table = new JTable(tableModel);
                    JScrollPane scrollPane = new JScrollPane(table);
                    add(scrollPane, BorderLayout.CENTER);
                }
                else{
                    tableModel.setDataVector(tableStudent, names);
                }
                revalidate();
                repaint();
                return;
            }
        }
        new StudentNotFoundException(surname, this);
    }
    private void updateTable(){
        Object [][] tableStudent = new Object[studentArrayList.size()][3];
        for(int i = 0; i < studentArrayList.size(); i++){
            Student student = studentArrayList.get(i);
            tableStudent[i][0] = student.getSurname();
```

### Продолжение листинга 17.7

```
        tableStudent[i][1] = student.getId();
        tableStudent[i][2] = student.getGPA();
    }
    String[] names = {"ФИО", "Номер", "GPA"};
    if(tableModel == null) {
        tableModel = new DefaultTableModel(tableStudent, names);
        table = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(table);
        add(scrollPane, BorderLayout.CENTER);
    }
    else{
        tableModel.setDataVector(tableStudent, names);
    }
    revalidate();
    repaint();
}
```



ФИО	Номер	GPA
Комаров П.В.	11111	4.4
Кораблев Г.Л.	13955	3.8
Прихмелин С.Д.	22190	4.1
Жулинин П.В.	13092	4.2

Рисунок 17.5 – Результат работы класса LabClass

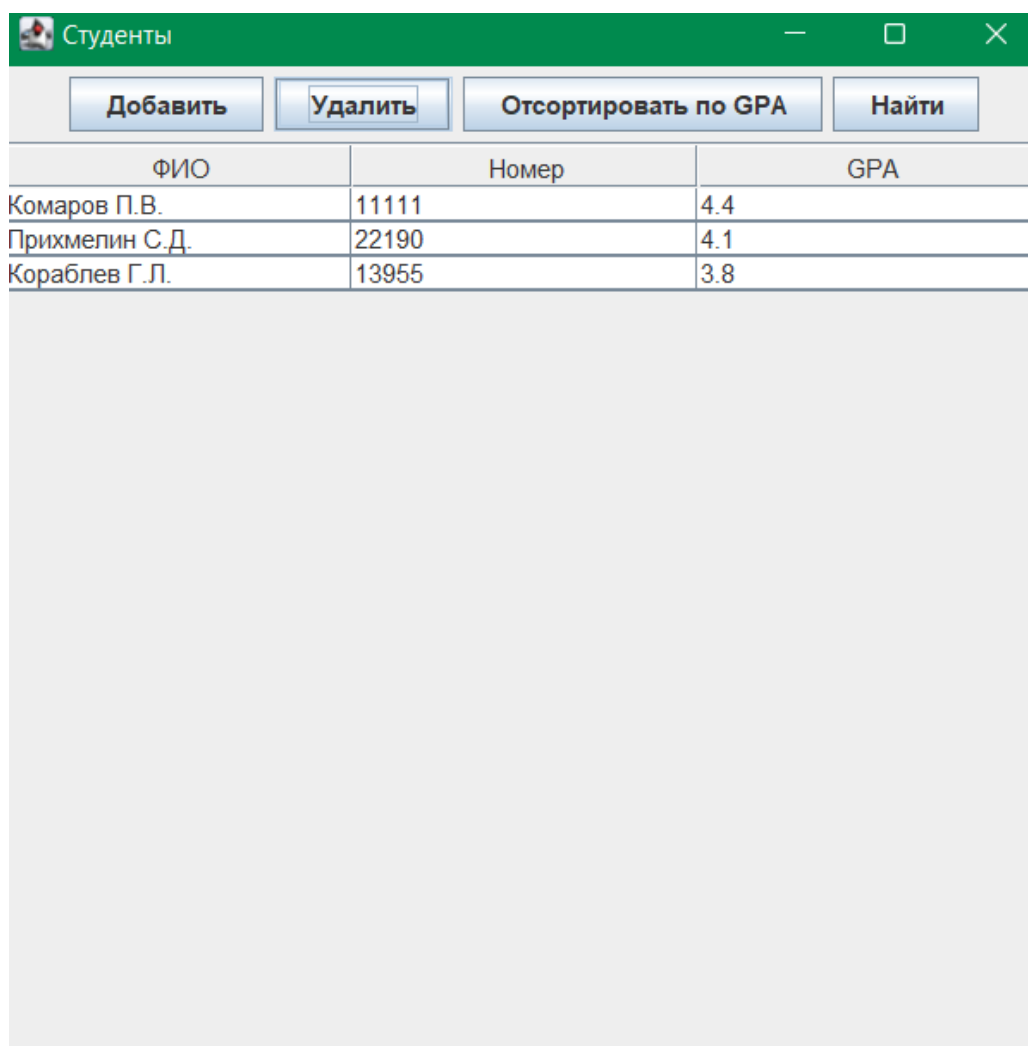


Рисунок 17.6 – Результат работы класса LabClass

### **Выводы по работе:**

Во время выполнения работы были получены навыки создания пользовательских исключений.

## **Практическая работа №18**

### **Цель работы**

Научиться работать с обобщенными типами в Java и применять их в программах.

### **Теоретическое введение**

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.

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»).

### **Выполнение лабораторной работы**

#### **Задание:**

1. Создать обобщенный класс с тремя параметрами (T, V, K). Класс содержит три переменные типа (T, V, K), конструктор, принимающий на вход параметры типа (T, V, K), методы возвращающие значения трех переменных. Создать метод, выводящий на консоль имена классов для трех переменных класса. Наложить ограничения на параметры типа: T должен реализовать интерфейс Comparable (классы оболочки, String), V должен реализовать интерфейс Serializable и расширять класс Animal, K – класс Number.
2. 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.

3. Написать класс Калькулятор (необобщенный), который содержит обобщенные статические методы - sum, multiply, divide, subtraction. Параметры этих методов - два числа разного типа, над которыми должна быть произведена операция.

4. Написать класс Matrix, на основе обобщенного типа, реализовать операции с матрицами.

#### Решение:

1. Программа, реализующая работу класса TVKClass, представлена в листинге 18.1. Результат работы программы представлен на рисунке 18.1.

Листинг 18.1 – Код программы класса TVKClass

```
public class TVKClass<T extends String, V extends Animal, K extends Integer>
{
    private T t;
    private V v;
    private K k;
    public TVKClass(T t, V v, K k) {
        this.t = t;
        this.v = v;
        this.k = k;
    }
    public T getT() {
        return t;
    }
    public V getV() {
        return v;
    }
    public K getK() {
        return k;
    }

    @Override
    public String toString() {
        return "T - "+getT()+"\nV - "+getV()+"\nK - "+getK()+"\n";
    }

    public static void main(String[] args) {
        TVKClass<String, Animal, Integer> tvk = new TVKClass<>("qwerty", new
        Animal("cat"), 13);
        System.out.println(tvk.toString());
    }
}
```

```

T - qwerty, class java.lang.String
V - JavaLabs.Animal@54a097cc, class JavaLabs.Animal
K - 13, class java.lang.Integer

Process finished with exit code 0

```

Рисунок 18.1 – Результат работы класса TVKClass

2. Программа, реализующая работу класса MinMax, представлена в листинге 18.2. Результат работы программы представлен на рисунке 18.2.

Листинг 18.2 – Код программы класса MinMax

```

public class MinMax<T extends Integer> {
    private T[] arr;
    public MinMax(T[] arr){
        this.arr = arr;
    }
    public int minElement() {
        T min = arr[0];
        for(int i = 1; i < arr.length; i++){
            if(arr[i].compareTo(min) < 0){
                min = arr[i];
            }
        }
        return min;
    }
    public int maxElement() {
        T max = arr[0];
        for(int i = 1; i < arr.length; i++){
            if(arr[i].compareTo(max) > 0){
                max = arr[i];
            }
        }
        return max;
    }

    public static void main(String[] args) {
        MinMax<Integer> minMax = new MinMax<>(new Integer[]{1, 3, 2, -4, 7,
-1, 4});
        System.out.println("minElement: "+minMax.minElement());
        System.out.println("maxElement: "+minMax.maxElement());
    }
}

```

```

minElement: -4
maxElement: 7

Process finished with exit code 0

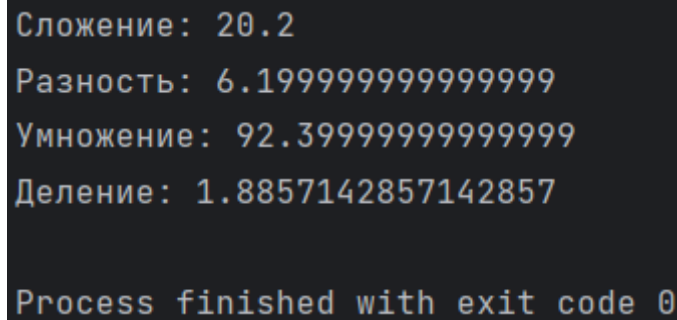
```

Рисунок 18.2 – Результат работы класса MinMax

3. Программа, реализующая работу класса Calculator, представлена в листинге 18.3. Результат работы программы представлен на рисунке 18.3.

### Листинг 18.3 – Код программы класса Calculator

```
public class Calculator {
    public static <T extends Number> double sum(T a, T b){
        return a.doubleValue() + b.doubleValue();
    }
    public static <T extends Number> double multiply(T a, T b){
        return a.doubleValue() * b.doubleValue();
    }
    public static <T extends Number> double divide(T a, T b){
        return a.doubleValue() / b.doubleValue();
    }
    public static <T extends Number> double subtraction(T a, T b){
        return a.doubleValue() - b.doubleValue();
    }
    public static void main(String[] args) {
        System.out.println("Сложение: "+sum(13.2, 7));
        System.out.println("Разность: "+subtraction(13.2, 7));
        System.out.println("Умножение: "+multiply(13.2, 7));
        System.out.println("Деление: "+divide(13.2, 7));
    }
}
```



Сложение: 20.2  
Разность: 6.1999999999999999  
Умножение: 92.3999999999999999  
Деление: 1.8857142857142857  
  
Process finished with exit code 0

Рисунок 18.3 – Результат работы класса Calculator

4. Программа, реализующая работу класса Matrix, представлена в листинге 18.4. Результат работы программы представлен на рисунке 18.4.

### Листинг 18.4 – Код программы класса Matrix

```
public class Matrix<T extends Integer> {
    private T [][] matrix1;
    private T [][] matrix2;
    public Matrix(T [][] matrix1, T [][] matrix2){
        this.matrix1 = matrix1;
        this.matrix2 = matrix2;
    }
    public void sum(){
        System.out.println("Сумма матриц:");
        for(int i = 0; i < matrix1.length; i++){
            for(int j = 0; j < matrix1[0].length; j++){
                System.out.print(matrix1[i][j].intValue() +
matrix2[i][j].intValue()+" ");
            }
            System.out.println();
        }
    }
    public void multiply(){
        int [][] multiplyMatrix = new
int[matrix1[0].length][matrix1.length];
        System.out.println("Произведение матриц:");
    }
}
```

## Продолжение листинга 18.4

```

        for(int i = 0; i < matrix1.length; i++){
            for(int j = 0; j < matrix1[0].length; j++){
                for(int k = 0; k < matrix1.length; k++){
                    multiplyMatrix[i][j] += matrix1[i][k].intValue() *
matrix2[k][j].intValue();
                }
            }
        }
        for (int i = 0; i < multiplyMatrix.length; i++){
            for (int j = 0; j < multiplyMatrix[0].length; j++){
                System.out.print(multiplyMatrix[i][j]+ " ");
            }
            System.out.println();
        }
    }
    public void transp(){
        Integer [][] tMatrix1 = new
Integer[matrix1[0].length][matrix1.length];
        System.out.println("Транспонированная первая матрица:");

        for(int i = 0; i < matrix1.length; i++){
            for(int j = 0; j < matrix1[0].length; j++){
                tMatrix1[j][i] = matrix1[i][j];
            }
        }
        for(int i = 0; i < tMatrix1.length; i++){
            for(int j = 0; j < tMatrix1[0].length; j++){
                System.out.print(tMatrix1[i][j]+" ");
            }
            System.out.println();
        }
        System.out.println("Транспонированная вторая матрица:");
        Integer[][] tMatrix2 = new
Integer[matrix2[0].length][matrix2.length];
        for (int i = 0; i < matrix2.length; i++){
            for(int j = 0; j < matrix2[0].length; j++){
                tMatrix2[j][i] = matrix2[i][j];
            }
        }
        for(int i = 0; i < tMatrix2.length; i++){
            for (int j = 0; j < tMatrix2[0].length; j++){
                System.out.print(tMatrix2[i][j]+" ");
            }
            System.out.println();
        }
    }
    public void print(){
        System.out.println("Вывод первой матрицы:");
        for(int i = 0; i < matrix1.length; i++){
            for(int j = 0; j < matrix1[0].length; j++){
                System.out.print(matrix1[i][j]+" ");
            }
            System.out.println();
        }
        System.out.println("Вывод второй матрицы: ");
        for(int i = 0; i < matrix2.length; i++){
            for(int j = 0; j < matrix2[0].length; j++){
                System.out.print(matrix2[i][j]+" ");
            }
            System.out.println();
        }
    }
}

```

#### Продолжение листинга 18.4

```
public static void main(String[] args) {  
    Integer[][] m1 = new Integer[][] {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  
    Integer[][] m2 = new Integer[][] {{5, 6, 7}, {2, 4, 1}, {0, 3, 8}};  
    Matrix<Integer> matrix = new Matrix<>(m1, m2);  
    matrix.print();  
    matrix.sum();  
    matrix.transp();  
    matrix.multiply();  
}
```

```
Вывод первой матрицы:  
1 2 3  
4 5 6  
7 8 9  
Вывод второй матрицы:  
5 6 7  
2 4 1  
0 3 8  
Сумма матриц:  
6 8 10  
6 9 7  
7 11 17  
Транспонированная первая матрица:  
1 4 7  
2 5 8  
3 6 9  
Транспонированная вторая матрица:  
5 2 0  
6 4 3  
7 1 8  
Произведение матриц:  
9 23 33  
30 62 81  
51 101 129  
Process finished with exit code 0
```

Рисунок 18.4 – Результат работы класса Matrix

#### Выводы по работе:

Во время выполнения работы были получены навыки работы с обобщенными типами и применения их в разработке программ.

## **Практическая работа №19**

### **Цель работы**

Научиться работать с обобщенными типами в Java и применять прием стирание типов в разработке программ на Джаве.

### **Теоретическое введение**

Из предыдущего примера может создаться видимость того, что компилятор заменяет параметризованный тип 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.

С помощью дженериков компилятор может выполнять проверку типов во время компиляции и обеспечивать безопасность использования типов во время выполнения.

В отличие от «шаблона» в C ++, который создает новый тип для каждого определенного параметризованного типа, в Java класс generics компилируется только один раз, и есть только один файл класса, который используется для создания экземпляров для всех конкретных типов.

### **Выполнение лабораторной работы**

#### **Задание:**

1. Написать метод для конвертации массива строк/чисел в список.
2. Написать класс, который умеет хранить в себе массив любых типов данных (int, long etc.). Реализовать метод, который возвращает любой элемент массива по индексу.

3. Написать функцию, которая сохранит содержимое каталога в список и выведет первые 5 элементов на экран.

Решение:

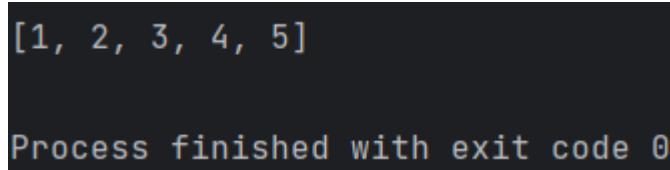
1. Программа, реализующая работу класса Convert, представлена в листинге 19.1. Результат работы программы представлен на рисунке 19.1.

**Листинг 19.1 – Код программы класса Convert**

```
import java.util.ArrayList;

public class Convert<T> {
    public static<T> ArrayList<T> convertArrToList(T[] arr){
        ArrayList<T> arrayList = new ArrayList<>();
        for(int i = 0; i < arr.length; i++){
            arrayList.add((T) arr[i]);
        }
        return arrayList;
    }

    public static void main(String[] args) {
        Integer[] arr = {1, 2, 3, 4, 5};
        System.out.println(convertArrToList(arr));
    }
}
```



```
[1, 2, 3, 4, 5]
Process finished with exit code 0
```

**Рисунок 19.1 – Результат работы класса Convert**

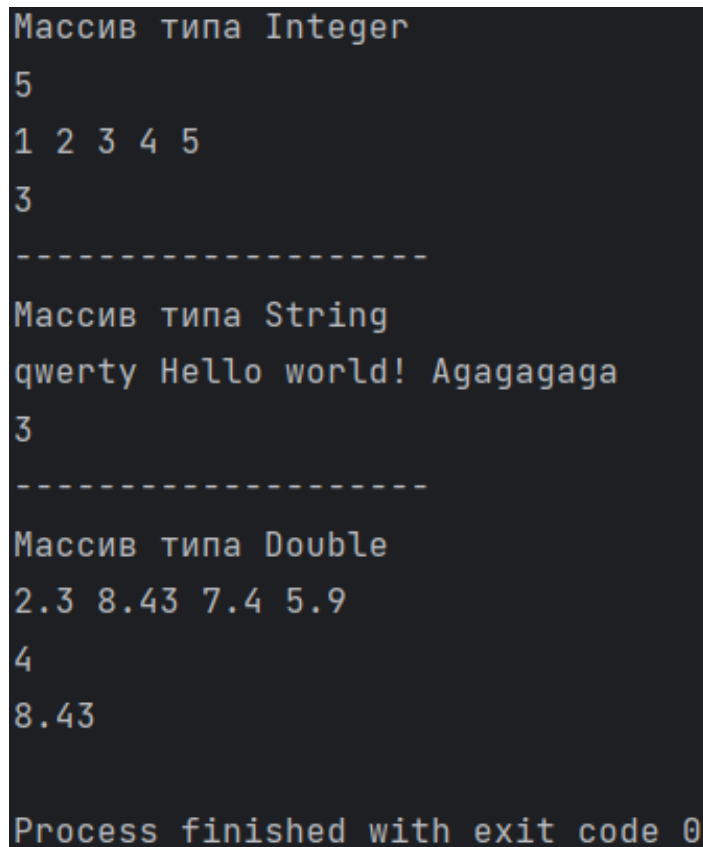
2. Программа, реализующая работу класса ArrayOfAnyTypes, представлена в листинге 19.2. Результат работы программы представлен на рисунке 19.2.

**Листинг 19.2 – Код программы класса ArrayOfAnyTypes**

```
public class ArrayOfAnyTypes<T> {
    private T[] arr;
    public ArrayOfAnyTypes(T[] arr){
        this.arr = arr;
    }
    public T getElement(int index){
        return arr[index];
    }
    public int getSize(){
        return arr.length;
    }
    public void getArr(){
        String str;
        for(int i = 0; i < arr.length; i++){
            System.out.print(arr[i]+" ");
        }
        System.out.println();
    }
}
```

## Продолжение листинга 19.2

```
public static void main(String[] args) {
    Integer[] intArr = {1, 2, 3, 4, 5};
    ArrayOfAnyTypes<Integer> integetArr = new ArrayOfAnyTypes<>(intArr);
    System.out.println("Массив типа Integer");
    System.out.println(+integetArr.getSize());
    integetArr.getArr();
    System.out.println(integetArr.getElement(2));
    System.out.println("-----");
    String[] strArr = {"qwerty", "Hello world!", "Agagagaga"};
    ArrayOfAnyTypes<String> stringArr = new ArrayOfAnyTypes<>(strArr);
    System.out.println("Массив типа String");
    stringArr.getArr();
    System.out.println(stringArr.getSize());
    System.out.println("-----");
    Double[] dblArr = {2.3, 8.43, 7.4, 5.9};
    ArrayOfAnyTypes<Double> doubleArr = new ArrayOfAnyTypes<>(dblArr);
    System.out.println("Массив типа Double");
    doubleArr.getArr();
    System.out.println(doubleArr.getSize());
    System.out.println(doubleArr.getElement(1));
}
}
```



```
Массив типа Integer
5
1 2 3 4 5
3
-----
Массив типа String
qwerty Hello world! Agagagaga
3
-----
Массив типа Double
2.3 8.43 7.4 5.9
4
8.43
Process finished with exit code 0
```

Рисунок 19.2 – Результат работы класса ArrayOfAnyTypes

3. Программа, реализующая работу класса ContentsDirectoryToList, представлена в листинге 19.3. Результат работы программы представлен на рисунке 19.3.

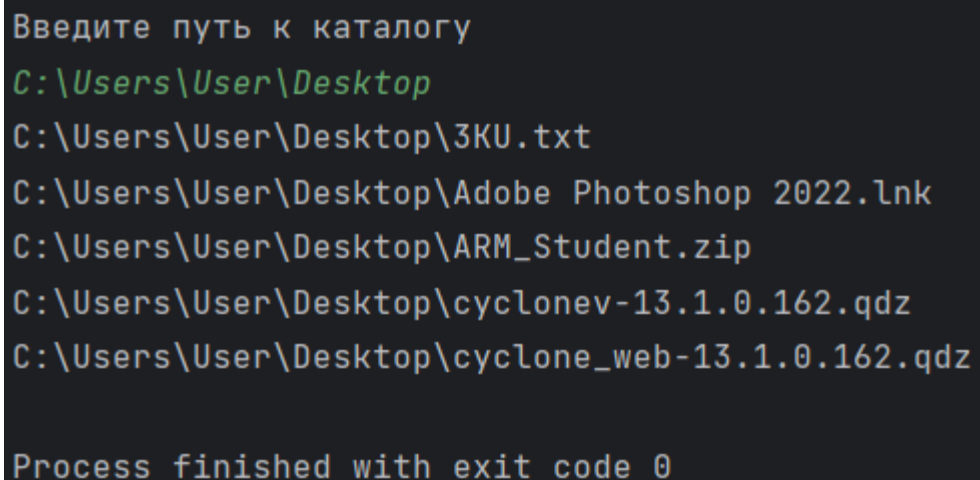


### Листинг 19.3 – Код программы класса ContentsDirectoryToList

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ContentsDirectoryToList {
    public static List<String> directoryPathMethod(String path) {
        List<String> fileList = new ArrayList<>(100);
        File file = new File(path);
        if(file.exists() && file.isDirectory()){
            for(int i = 0; i < file.listFiles().length; i++){
                fileList.add(String.valueOf(file.listFiles()[i]));
            }
        }
        return fileList;
    }
    //C:\Users\User\Desktop

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Введите путь к каталогу");
        String path = scanner.next();
        List<String> fileList = directoryPathMethod(path);
        for(int i = 0; i < 5; i++){
            System.out.println(fileList.get(i));
        }
    }
}
```



```
Введите путь к каталогу
C:\Users\User\Desktop
C:\Users\User\Desktop\3KU.txt
C:\Users\User\Desktop\Adobe Photoshop 2022.lnk
C:\Users\User\Desktop\ARM_Student.zip
C:\Users\User\Desktop\cyclonev-13.1.0.162.qdz
C:\Users\User\Desktop\cyclone_web-13.1.0.162.qdz
Process finished with exit code 0
```

Рисунок 19.3 – Результат работы класса ContentsDirectoryToList

#### Выводы по работе:

Во время выполнения работы были получены навыки работы с обобщенными типами в Java и применения приема стирания типов в разработке программ на Джаве.

## Практическая работа №20

### Цель работы

Научиться разрабатывать программы с абстрактными типами данных на языке Джава.

### 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.

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.

### Выполнение лабораторной работы

#### Задание:

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

#### Решение:

Программа, реализующая работу класса `StackRPN`, представлен в листинге 20.1. Результаты работы программы представлены на рисунках 20.1

#### Листинг 20.1 – Код программы класса `StackRPN`

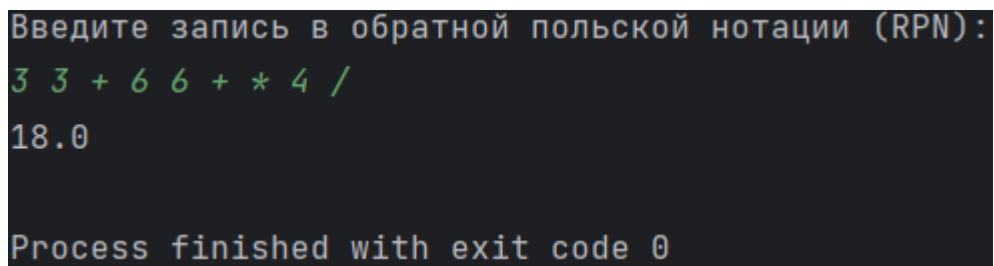
```
import java.util.Scanner;
import java.util.Stack;

public class StackRPN {

    public static void main(String[] args) throws Exception {
```

## Продолжение листинга 20.1

```
Stack <Double> stack = new Stack<>();
System.out.println("Введите запись в обратной польской нотации
(RPN): ");
Scanner scanner = new Scanner(System.in);
String expression = scanner.nextLine();
String[] token = expression.split(" ");
for (int i = 0; i < token.length; i++) {
    Double number2;
    Double number1;
    switch (token[i]) {
        case "+":
            number2 = stack.pop();
            number1 = stack.pop();
            stack.push(number2 + number1);
            break;
        case "-":
            number2 = stack.pop();
            number1 = stack.pop();
            stack.push(number1 - number2);
            break;
        case "*":
            number2 = stack.pop();
            number1 = stack.pop();
            stack.push(number1 * number2);
            break;
        case "/":
            number2 = stack.pop();
            number1 = stack.pop();
            if (number2 != 0) {
                stack.push(number1 / number2);
            }
            else {
                throw new ArithmeticException("Деление на ноль!");
            }
            break;
        default:
            stack.push(Double.parseDouble(token[i]));
    }
}
System.out.println(stack.pop());
}
```



```
Введите запись в обратной польской нотации (RPN):
3 3 + 6 6 + * 4 /
18.0
Process finished with exit code 0
```

Рисунок 20.1 – Результат работы программы

### Выводы по работе:

Во время выполнения работы были получены навыки работы со стеком и RPN-нотацией.

## **Практическая работа №21**

### **Цель работы**

Научиться разрабатывать программы с абстрактными типами данных на языке Джава.

### **Теоретическое введение**

Очередь – это структура данных, в которой элемент, который приходит в очередь первым, удаляется первым, то есть работа очереди строго соответствует политике FIFO (First-In-First-Out). Очередь также можно определить как список или коллекцию, в которой вставка выполняется с одного конца, известного как конец очереди, тогда как удаление выполняется с другого конца, известного как начало очереди. Реальным примером очереди является очередь за билетами возле кинозала, где человек, входящий в очередь первым, получает билет первым, а последний человек, входящий в очередь, получает билет последним. Аналогичный подход используется в очереди как в структуре данных.

### **Выполнение лабораторной работы**

#### Задание:

1. Реализовать очередь на массиве. 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.

Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях. Напишите тесты реализованным классам.

2. Реализовать очередь на связанном списке. Определите интерфейс очереди Queue и опишите его контракт. Реализуйте класс LinkedList – очередь на связанном списке. Выделите общие части классов LinkedList и ArrayQueue в базовый класс AbstractQueue.

### Решение:

1. Программы, реализующие работу классов ArrayQueueModule (листинг 21.1), ArrayQueueADT (листинг 21.2), ArrayQueue (листинг 21.3) и тестового класса TestArray (листинг 21.4). Результат тестирования представлен на рисунке 21.1.

Листинг 21.1 – Код программы класса ArrayQueueModule

```
public class ArrayQueueModule {
    int SIZE = 5;
    Object[] items;
    int front, rear;
    private static ArrayQueueModule obj;
    public ArrayQueueModule() {
        items = new Object[SIZE];
    }
    public void enqueue(Object elem) {
        items[rear] = elem;
        rear++;
        if (rear == items.length) {
            rear = 0;
        }
        if (rear == front) {
            throw new ArrayStoreException("Очередь заполнена!");
        }
    }
    public Object element() {
        if (isEmpty()) {
            throw new IndexOutOfBoundsException("Очередь пуста!");
        }
        else {
            return items[front];
        }
    }
    public Object dequeue() {
        if (isEmpty()) {
            throw new IndexOutOfBoundsException("Queue is empty!");
        }
        if (front == items.length) {
            front = 0;
        }
        return items[front++];
    }
    public boolean isEmpty() {
```

## Продолжение листинга 21.1

```
        return front == rear;
    }
    public int size() {
        return items.length - 1;
    }
    public void clear() {
        front = 0;
        rear = 0;
    }
    public static ArrayQueueModule getObj() {
        if (obj == null) {
            obj = new ArrayQueueModule();
        }
        return obj;
    }
}
```

## Листинг 21.2 – Код программы класса ArrayQueueADT

```
public class ArrayQueueADT {
    public ArrayQueueADT(ArrayQueueModule queue) {
        this.queue = queue;
    }
    private ArrayQueueModule queue;
    public void enqueue(Object v) {
        queue.enqueue(v);
    }
    public Object dequeue() {
        return queue.dequeue();
    }
    public Object element() {
        return queue.element();
    }
    public boolean isEmpty() {
        return queue.isEmpty();
    }
    public void clear() {
        queue.clear();
    }
    public int size() {
        return queue.size();
    }
}
```

## Листинг 21.3 – Код программы класса ArrayQueue

```
public class ArrayQueue {
    private ArrayQueueModule queue;

    public ArrayQueue() {
        this.queue = ArrayQueueModule.getObj();
    }
    public void enqueue(Object v) {
        queue.enqueue(v);
    }
    public Object dequeue() {
        return queue.dequeue();
    }
    public Object element() {
        return queue.element();
    }
    public boolean isEmpty() {
        return queue.isEmpty();
    }
}
```

### Продолжение листинга 21.3

```
}  
public void clear() {  
    queue.clear();  
}  
public int size() {  
    return queue.size();  
}  
}
```

### Листинг 21.4 – Код программы класса TestArray

```
public class TestArray {  
    public static void main(String[] args) {  
        ArrayQueueModule queue1 = ArrayQueueModule.getObj();  
        queue1.enqueue(11);  
        queue1.enqueue(22);  
        queue1.enqueue(33);  
        queue1.enqueue(44);  
        System.out.println(queue1.dequeue());  
        System.out.println("Очередь пуста? " + queue1.isEmpty());  
  
        ArrayQueueADT queue2 = new ArrayQueueADT(queue1);  
        System.out.println(queue2.dequeue());  
        System.out.println(queue2.element());  
        System.out.println("Очередь пуста? " + queue2.isEmpty());  
  
        ArrayQueue queue3 = new ArrayQueue();  
        System.out.println(queue3.dequeue());  
        System.out.println(queue3.element());  
        System.out.println("Очередь пуста? " + queue3.isEmpty());  
        System.out.println(queue3.dequeue());  
        System.out.println("Очередь пуста? " + queue3.isEmpty());  
    }  
}
```

```
11  
Очередь пуста? false  
22  
33  
Очередь пуста? false  
33  
44  
Очередь пуста? false  
44  
Очередь пуста? true  
  
Process finished with exit code 0
```

Рисунок 21.1 – Результат работы тестового класса TestArray

2. Программы, реализующие работу интерфейса Queue (листинг 21.5), классов AbstractQueue (листинг 21.6), LinkedList (листинг 21.7) и

тестового класса TestLinked (листинг 21.8). Результат работы тестового класса представлен на рисунке 21.2.

#### Листинг 21.5 – Код программы интерфейса Queue

```
public interface Queue {  
    void enqueue(Object o);  
    Object element();  
    Object dequeue();  
    boolean isEmpty();  
    boolean clear();  
}
```

#### Листинг 21.6 – Код программы класса AbstractQueue

```
public class AbstractQueue {  
    protected int rear, front;  
}
```

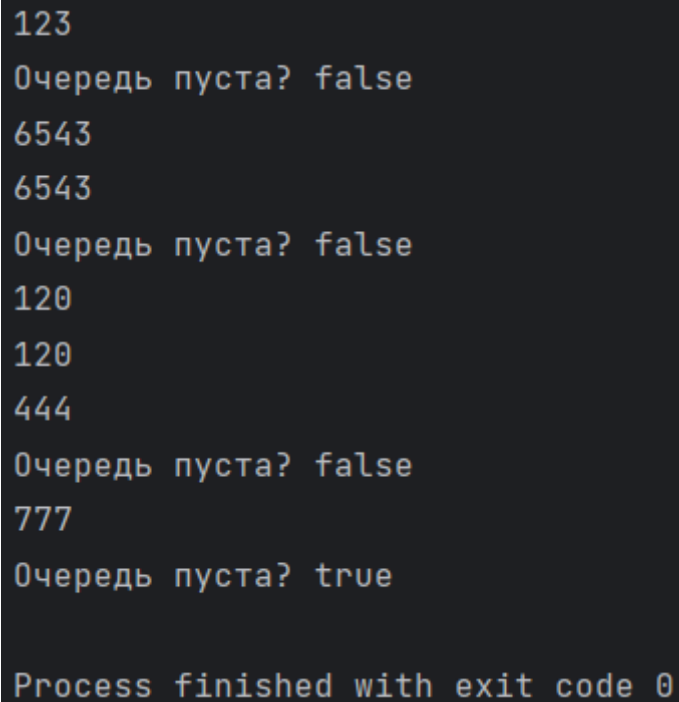
#### Листинг 21.7 – Код программы класса LinkedQueue

```
import java.util.LinkedList;  
  
public class LinkedQueue extends AbstractQueue implements Queue {  
    private LinkedList<Object> queue;  
    public LinkedQueue() {  
        queue = new LinkedList<>();  
        front = 0;  
        rear = 0;  
    }  
    @Override  
    public void enqueue(Object o) {  
        queue.add(o);  
        rear++;  
    }  
    @Override  
    public Object element() {  
        if (isEmpty()) {  
            throw new IndexOutOfBoundsException("Очередь пуста!");  
        }  
        return queue.get(front);  
    }  
    @Override  
    public Object dequeue() {  
        if (isEmpty()) {  
            throw new IndexOutOfBoundsException("Очередь пуста!");  
        }  
        return queue.remove(front);  
    }  
    @Override  
    public boolean isEmpty() {  
        return queue.isEmpty();  
    }  
    @Override  
    public boolean clear() {  
        queue.clear();  
        front = 0;  
        rear = 0;  
        return false;  
    }  
}
```



## Листинг 21.8 – Код программы тестового класса TestLinked

```
public class TestLinked {  
    public static void main(String[] args){  
        LinkedList queue = new LinkedList();  
        queue.enqueue(123);  
        queue.enqueue(6543);  
        queue.enqueue(120);  
        queue.enqueue(444);  
        queue.enqueue(777);  
        System.out.println(queue.dequeue());  
        System.out.println("Очередь пуста? "+queue.isEmpty());  
        System.out.println(queue.dequeue());  
        System.out.println("Очередь пуста? "+queue.isEmpty());  
        System.out.println(queue.dequeue());  
        System.out.println("Очередь пуста? "+queue.isEmpty());  
        System.out.println(queue.dequeue());  
        System.out.println("Очередь пуста? "+queue.isEmpty());  
        System.out.println(queue.dequeue());  
        System.out.println("Очередь пуста? "+queue.isEmpty());  
    }  
}
```



```
123  
Очередь пуста? false  
6543  
6543  
Очередь пуста? false  
120  
120  
444  
Очередь пуста? false  
777  
Очередь пуста? true  
  
Process finished with exit code 0
```

Рисунок 21.2 – Результат работы тестового класса TestLinked

### Выводы по работе:

Во время выполнения работы были получены навыки работы с очередями.

## **Практическая работа №22**

### **Цель работы**

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 программным продуктом;
- 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.

## Выполнение лабораторной работы

### Задание:

1. Разработать программную реализацию по UML диаграмме, представленной на рисунке 22.1 с использованием изучаемых паттернов;

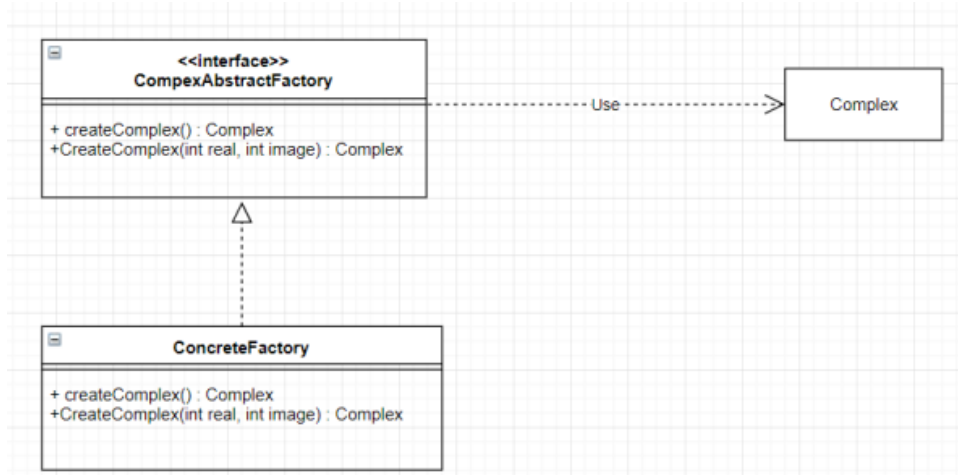


Рисунок 22.1 – UML диаграмма проекта ComplexNumber с обработкой исключений

2. Реализовать класс Абстрактная фабрика (рисунок 22.2) 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).

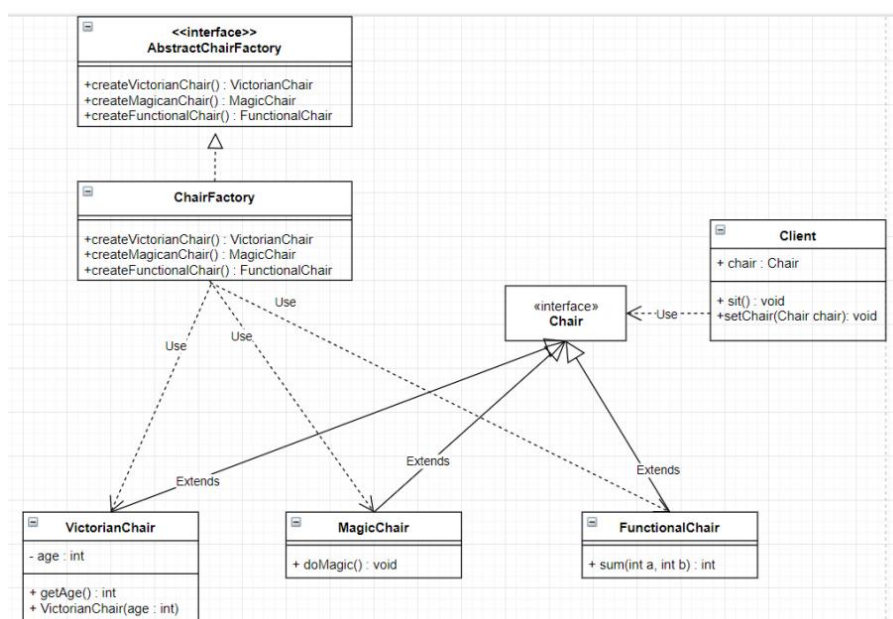


Рисунок 22.2 – UML диаграмма проекта для Фабрики стульев

3. Легенда “Компании XXX нужно написать редактор текста, редактор изображений и редактор музыки.

В трёх приложениях будет очень много общего: главное окно, панель инструментов, команды меню будут весьма схожими. Чтобы не писать повторяющуюся основу трижды, вам поручили разработать основу (каркас) приложения, которую можно использовать во всех трёх редакторах.” На совещании в компании была принята следующая архитектура проекта:

Исходные данные. Есть некий базовый интерфейс `IDocument`, представляющий документ неопределённого рода. От него впоследствии будут унаследованы конкретные документы: `TextDocument`, `ImageDocument`, `MusicDocument` и т.п. Интерфейс `IDocument` перечисляет общие свойства и операции для всех документов.

При нажатии пунктов меню `File -> New` и `File -> Open` требуется создать новый экземпляр документа (конкретного подкласса). Однако каркас не должен быть привязан ни к какому конкретному виду документов. Нужно создать фабричный интерфейс `ICreateDocument`. Этот интерфейс содержит два абстрактных фабричных метода: `CreateNew` и `CreateOpen`, оба возвращают экземпляр `IDocument`. Каркас оперирует одним экземпляром `IDocument` и одним экземпляром `ICreateDocument`. Какие конкретные классы будут подставлены сюда, определяется во время запуска приложения.

Требуется:

- создать перечисленные классы. Создать каркас приложения — окно редактора с меню `File`. В меню `File` реализовать пункты `New`, `Open`, `Save`, `Exit`;
- продемонстрировать работу каркаса на примере текстового редактора.

Потребуется создать конкретный унаследованный класс `TextDocument` и фабрику для него — `CreateTextDocument`.

Решение:

1. Программы, реализующие работу интерфейса ComplexAbstractFactory (листинг 22.1), классов Complex (листинг 22.2), ConcreteFactory (листинг 22.3), тестового класса TestComplex (листинг 22.4). Результат работы тестового класса представлен на рисунке 22.3.

**Листинг 22.1 – Код программы интерфейса ComplexAbstractFactory**

```
public interface ComplexAbstractFactory {  
    Complex createComplex();  
  
    Complex CreateComplex(int real, int image);  
}
```

**Листинг 22.2 – Код программы класса Complex**

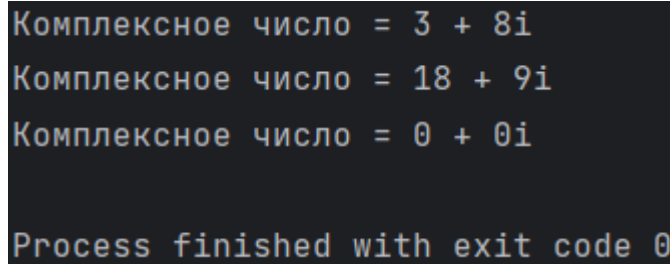
```
public class Complex {  
    private int real;  
    private int image;  
    public Complex() {  
        this.real = 0;  
        this.image = 0;  
    }  
    public Complex(int real, int image) {  
        this.real = real;  
        this.image = image;  
    }  
    public int getReal() {  
        return real;  
    }  
    public void setReal(int real) {  
        this.real = real;  
    }  
    public int getImage() {  
        return image;  
    }  
    public void setImage(int image) {  
        this.image = image;  
    }  
    @Override  
    public String toString() {  
        return "Комплексное число = " + real + " + " + image + "i";  
    }  
}
```

**Листинг 22.3 – Код программы класса ConcreteFactory**

```
public class ConcreteFactory implements ComplexAbstractFactory {  
    @Override  
    public Complex createComplex() {  
        return new Complex();  
    }  
  
    @Override  
    public Complex CreateComplex(int real, int image) {  
        return new Complex(real, image);  
    }  
}
```

#### Листинг 22.4 – Код программы тестового класса TestComplex

```
public class TestComplex {  
    public static void main(String[] args){  
        ComplexAbstractFactory complex = new ConcreteFactory();  
        System.out.println(complex.CreateComplex(3,8));  
        System.out.println(complex.CreateComplex(18, 9));  
        System.out.println(complex.createComplex());  
    }  
}
```



```
Комплексное число = 3 + 8i  
Комплексное число = 18 + 9i  
Комплексное число = 0 + 0i  
  
Process finished with exit code 0
```

Рисунок 22.3 – Результат работы тестового класса TestComplex

2. Программы, реализующие работу интерфейсов Chair (листинг 22.5), AbstractChairFactory (листинг 22.6), классов Client (листинг 22.7), ChairFactory (листинг 22.8), FunctionalChair (листинг 22.9), MagicChair (листинг 22.10), VictorianChair (листинг 22.11), тестового класса TestChair (листинг 22.12). Результат работы тестового класса представлен на рисунке 22.4.

#### Листинг 22.5 – Код программы интерфейса Chair

```
public interface Chair {  
}
```

#### Листинг 22.6 – Код программы интерфейса AbstractChairFactory

```
public interface AbstractChairFactory {  
  
    VictorianChair createVictorianChair();  
    MagicChair createMagicanChair();  
    FunctionalChair createFunctionalChair();  
}
```

#### Листинг 22.7 – Код программы класса Client

```
public class Client implements Chair {  
    Chair chair;  
  
    public void sit(){  
        System.out.println("Клиент сел на стул");  
    }  
  
    public void setChair(Chair chair){  
        this.chair = chair;  
    }  
}
```

### Листинг 22.8 – Код программы класса ChairFactory

```
public class ChairFactory implements AbstractChairFactory {
    @Override
    public VictorianChair createVictorianChair() {
        return new VictorianChair(50);
    }
    @Override
    public MagicChair createMagicanChair() {
        return new MagicChair();
    }
    @Override
    public FunctionalChair createFunctionalChair() {
        return new FunctionalChair();
    }
}
```

### Листинг 22.9 – Код программы класса FunctionalChair

```
public class FunctionalChair implements Chair {
    public int sum(int a, int b){
        return a+b;
    }
}
```

### Листинг 22.10 – Код программы класса MagicChair

```
public class MagicChair implements Chair {
    public void doMagic(){
        System.out.println("Магический стул");
    }
}
```

### Листинг 22.11 – Код программы класса VictorianChair

```
public class VictorianChair implements Chair {
    private int age;
    public VictorianChair(int age) {
        this.age = age;
    }
    public int getAge() {
        return age;
    }
}
```

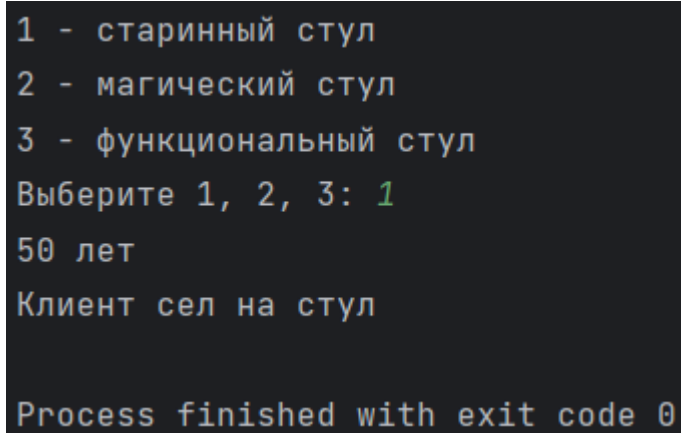
### Листинг 22.12 – Код программы тестового класса TestChair

```
import java.util.Scanner;

public class TestChair {
    public static void main(String[] args){
        AbstractChairFactory abstractChairFactory = new ChairFactory();
        Scanner scanner = new Scanner(System.in);
        Client client = new Client();
        System.out.println(("1 - старинный стул \n2 - магический стул \n3 - функциональный стул"));
        System.out.print("Выберите 1, 2, 3: ");
        int choice = scanner.nextInt();
        switch (choice){
            case 1:
                VictorianChair victorianChair =
abstractChairFactory.createVictorianChair();
                System.out.println(victorianChair.getAge()+" лет");
                break;
            case 2:
```

## Продолжение листинга 22.12

```
        MagicChair magicChair =
abstractChairFactory.createMagicanChair();
        magicChair.doMagic();
        break;
    case 3:
        FunctionalChair functionalChair =
abstractChairFactory.createFunctionalChair();
        System.out.println("Сумма = " + functionalChair.sum(44,17));
        break;
    default:
        System.out.println("Неверный ввод!");
        return;
    }
    client.sit();
}
```



```
1 - старинный стул
2 - магический стул
3 - функциональный стул
Выберите 1, 2, 3: 1
50 лет
Клиент сел на стул

Process finished with exit code 0
```

Рисунок 22.4 – Результат работы тестового класса TestChair

3. Программы, реализующие работу интерфейсов ICreateDocument (листинг 22.13), IDocument (листинг 22.14), классов CarcasModel (листинг 22.15), CarcasView (листинг 22.16), CarcasController (листинг 22.17), CreateTestDoc (листинг 22.18), TextDocument (листинг 22.19), тестового класса TestWindow (листинг 22.20). Результат работы тестового класса представлен на рисунках 22.5 – 22.6.

## Листинг 22.13 – Код программы интерфейса ICreateDocument

```
import java.io.*;
public interface ICreateDocument <T>{
    IDocument<T> CreateNew();
    IDocument<T> CreateOpen(String path) throws FileNotFoundException;
}
```

## Листинг 22.14 – Код программы интерфейса IDocument

```
import java.io.IOException;

public interface IDocument <T>{
    String getName();
}
```



## Продолжение листинга 22.14

```
void setName(String name);
String getContent();
void setContent(String content);
void save() throws IOException;
}
```

## Листинг 22.15 – Код программы класса CarcasModel

```
public class CarcasModel {
    private IDocument<?> document;
    public CarcasModel() {
    }
    public CarcasModel(IDocument<?> document) {
        this.document = document;
    }
    public IDocument<?> getDocument() {
        return document;
    }
    public void setDocument(IDocument<?> document) {
        this.document = document;
    }
}
```

## Листинг 22.16 – Код программы класса CarcasView

```
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class CarcasView<T> extends JFrame {
    IDocument<T> doc;
    class newDocA extends ActionEvent {
        public newDocA(Object obj) {
            super(obj, 0, "");
        }
    }
    class openDocA extends ActionEvent {
        private String path;
        public String getPath() {
            return path;
        }
        public openDocA(Object source, String path) {
            super(source, 0, "");
            this.path = path;
        }
    }
    JMenuBar menu = new JMenuBar();
    ArrayList<ActionListener> saveListener = new ArrayList<>();
    ArrayList<ActionListener> newListener = new ArrayList<>();
    ArrayList<ActionListener> openListener = new ArrayList<>();
    class saveDocA extends ActionEvent {
        private IDocument<T> doc;
        public saveDocA(Object obj, IDocument<T> doc) {
            super(obj, 0, "");
            this.doc = doc;
        }
        public IDocument<T> getDocument() {
            return doc;
        }
    }
}
```

## Продолжение листинга 22.16

```
void addNewListener(ActionListener actionListener) {
    newListener.add(actionListener);
}
void addOpenListener(ActionListener actionListener) {
    openListener.add(actionListener);
}
void addSaveListener(ActionListener actionListener) {
    saveListener.add(actionListener);
}
void setOpenDoc(String path) {
    for (ActionListener actionListener: openListener) {
        actionListener.actionPerformed(new openDocA(this, path));
    }
}

void setNewDoc() {
    for (ActionListener actionListener: newListener) {
        actionListener.actionPerformed(new newDocA(this));
    }
}

void setSaveDoc() {
    for (ActionListener actionListener: saveListener) {
        actionListener.actionPerformed(new saveDocA(this, doc));
    }
}

CarcasView() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(400, 200);
    JMenu fileItem = new JMenu("File");

    JMenuItem newPosition = new JMenuItem("Create");
    newPosition.addActionListener(e -> {
        setNewDoc();
    });

    JMenuItem openPosition = new JMenuItem("Open");
    openPosition.addActionListener(e -> {
        JFileChooser file_choose = new JFileChooser();
        if (file_choose.showDialog(getContentPane(), "OK") ==
JFileChooser.APPROVE_OPTION) {
            setOpenDoc(file_choose.getSelectedFile().getAbsolutePath());
        }
    });

    JMenuItem savePosition = new JMenuItem("Save");
    savePosition.addActionListener(e -> {
        setSaveDoc();
    });

    JMenuItem exitPosition = new JMenuItem("Exit");
    exitPosition.addActionListener(e -> {
        setVisible(false);
    });
    fileItem.add(newPosition);
    fileItem.add(openPosition);
    fileItem.add(savePosition);
    fileItem.add(exitPosition);
    menu.add(fileItem);
    setJMenuBar(menu);
    setBounds(0, 0, 390, 200);
}
```

## Продолжение листинга 22.16

```
    }

    void showDocText(TextDocument textDocument) {
        this.doc = (IDocument<T>) textDocument;
        setTitle(textDocument.getName());
        getContentPane().removeAll();
        repaint();
        JTextArea textFieldBig = new JTextArea(textDocument.getContent());
        textFieldBig.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent documentEvent) {
                ((TextDocument) doc).setContent(textFieldBig.getText());
            }
            @Override
            public void removeUpdate(DocumentEvent documentEvent) {
                ((TextDocument) doc).setContent(textFieldBig.getText());
            }
            @Override
            public void changedUpdate(DocumentEvent documentEvent) {
                ((TextDocument) doc).setContent(textFieldBig.getText());
            }
        });
        add(textFieldBig);
        setBounds(0, 0, 450, 200);
    }
}
```

## Листинг 22.17 – Код программы класса CarcasController

```
import javax.swing.*;
import java.io.FileNotFoundException;
import java.io.IOException;
public class CarcasController {
    private final CarcasModel model;
    private final CarcasView view;
    public CarcasController() {
        this.model = new CarcasModel();
        this.view = new CarcasView<>();

        view.addNewListener(e -> {
            model.setDocument(new CreateTextDoc().CreateNew());
            view.showDocText((TextDocument) model.getDocument());
        });
        view.addOpenListener(e -> {
            try {
                model.setDocument(new
CreateTextDoc().CreateOpen(((CarcasView.openDocA) e).getPath()));
                view.showDocText((TextDocument) model.getDocument());
            } catch (FileNotFoundException exc) {
                JOptionPane.showMessageDialog(view, "File does not
exist");
            }
        });
        view.addSaveListener(e -> {
            var document = ((CarcasView.saveDocA) e).getDocument();
            System.out.println("Saved");
            try {
                document.save();
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
        });
    }
}
```

## Продолжение листинга 22.17

```
        });  
    }  
  
    void showWindow() {  
        this.view.setVisible(true);  
    }  
}
```

## Листинг 22.18 – Код программы класса CreateTestDoc

```
import java.io.*;  
import java.util.stream.Collectors;  
  
public class CreateTextDoc<T> implements ICreateDocument{  
    public IDocument<T> CreateNew() {  
        return (IDocument<T>) new TextDocument("New document");  
    }  
  
    @Override  
    public IDocument<T> CreateOpen(String path) throws FileNotFoundException  
    {  
        BufferedReader file = new BufferedReader(new FileReader(path));  
        return (IDocument<T>) new TextDocument(path,  
file.lines().collect(Collectors.joining("\n")));  
    }  
}
```

## Листинг 22.19 – Код программы класса TextDocument

```
import java.io.*;  
public class TextDocument implements IDocument{  
    private String text;  
    private String name;  
    TextDocument(String name) {  
        this.name = name;  
    }  
    TextDocument(String name, String text) {  
        this.name = name;  
        this.text = text;  
    }  
    @Override  
    public String getName() {  
        return name;  
    }  
    @Override  
    public void setName(String name) {  
        this.name = name;  
    }  
    @Override  
    public String getContent() {  
        return text;  
    }  
    @Override  
    public void setContent(String text) {  
        this.text = text;  
    }  
    @Override  
    public void save() throws IOException {  
        BufferedWriter writer = new BufferedWriter(new FileWriter(name));  
        writer.write(text);  
        writer.close();  
    }  
}
```

## Листинг 22.20 – Код программы тестового класса TestWindow

```
import javax.swing.*;

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

            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored){}

        CarcasController controller = new CarcasController();
        controller.showWindow();
    }
}
```

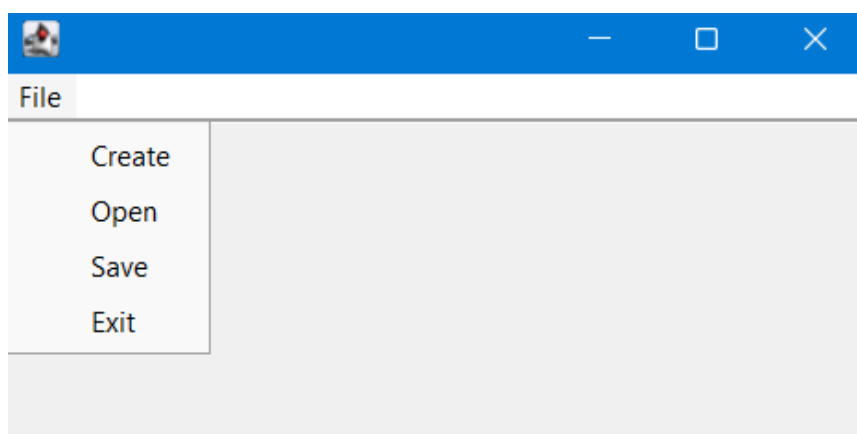


Рисунок 22.5 – Результат работы тестового класса TestWindow

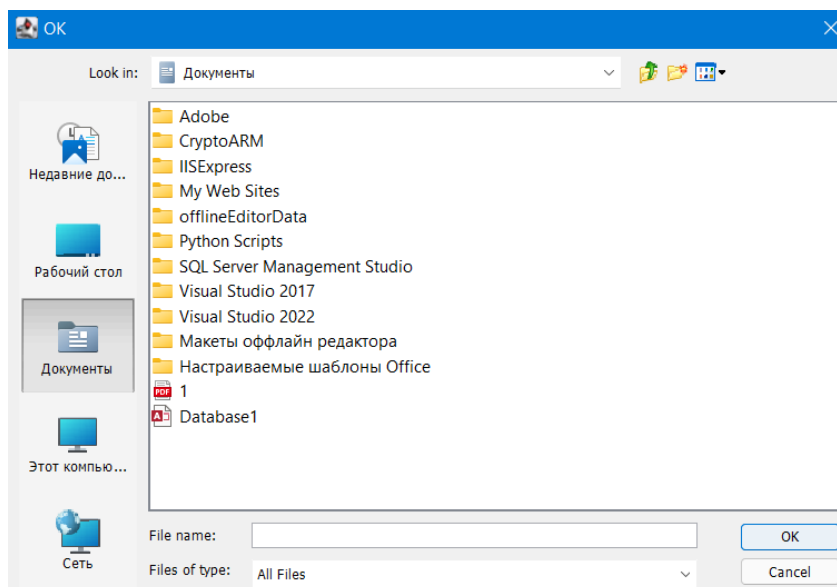


Рисунок 22.6 – Результат работы тестового класса TestWindow

### Выводы по работе:

Во время выполнения работы были получены навыки применения порождающих паттернов при разработке программ на Java.

## Практическая работа №23

### Цель работы

Закрепить навыки разработки программ на языке Java.

### Выполнение лабораторной работы

#### Задание:

1. Создайте класс Drink – напиток. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием. Класс должен быть определен как неизменяемый (Immutable class).

#### Конструкторы:

- принимающий два параметра – название и описание. Стоимость при этом инициализируется значением 0;
- принимающий три параметра – стоимость, название и описание.

#### Методы:

- возвращающий стоимость;
- возвращающий название;
- возвращающий описание.

#### Дополнительные требования:

Вместо литералов в коде (магических констант) необходимо использовать константы класса, содержащие эти значения. Пояснение: в этом случае вы локализуете изменения этих значений в одном месте, а имя константы скажет нам о сути литерала. Этот класс должен быть неизменяемым (правила проектирования таких классов приводятся в лекциях).

2. Создайте интерфейс Item для работы с позициями заказа. Интерфейс определяет 3 метода:

- возвращает стоимость;
- возвращает название;
- возвращает описание позиции.

Класс Drink и Dish должны реализовывать этот интерфейс. Класс Dish сделайте неизменяемым (аналогично Drink). Order должен хранить (удалять и добавлять) не только экземпляры класса Dish, но и Drink (Для этого разработайте классы Order и TablesOrderManager).

3. Создайте класс InternetOrder, который моделирует сущность интернет-заказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки. Внимание: список реализуется самостоятельно.

Конструкторы:

- Не принимающий параметров (для списка создается только головной элемент, сам список пуст);
- Принимающий массив позиций заказа (создаем список из n позиций).

Методы:

- Добавляющий позицию в заказ (принимает ссылку типа item). Пока этот метод возвращает истину после выполнения операции добавления элемента;
- Удаляющий позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Если позиций с заданным названием несколько, всегда удаляются последние. Возвращает логическое значение (true, если элемент был удален);
- Удаляющий все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.
- Возвращающий общее число позиций заказа (повторяющиеся тоже считаются) в заказе;
- Возвращающий массив заказанных блюд и напитков (значений null в массиве быть не должно);
- Возвращающий общую стоимость заказа;

- Возвращающий число заказанных блюд или напитков (принимает название блюда или напитка в качестве параметра);
- Возвращающий массив названий заказанных блюд и напитков (без повторов);
- Возвращающий массив позиций заказа, отсортированный по убыванию цены.

Дополнительные требования продемонстрированы на рисунке 23.1.

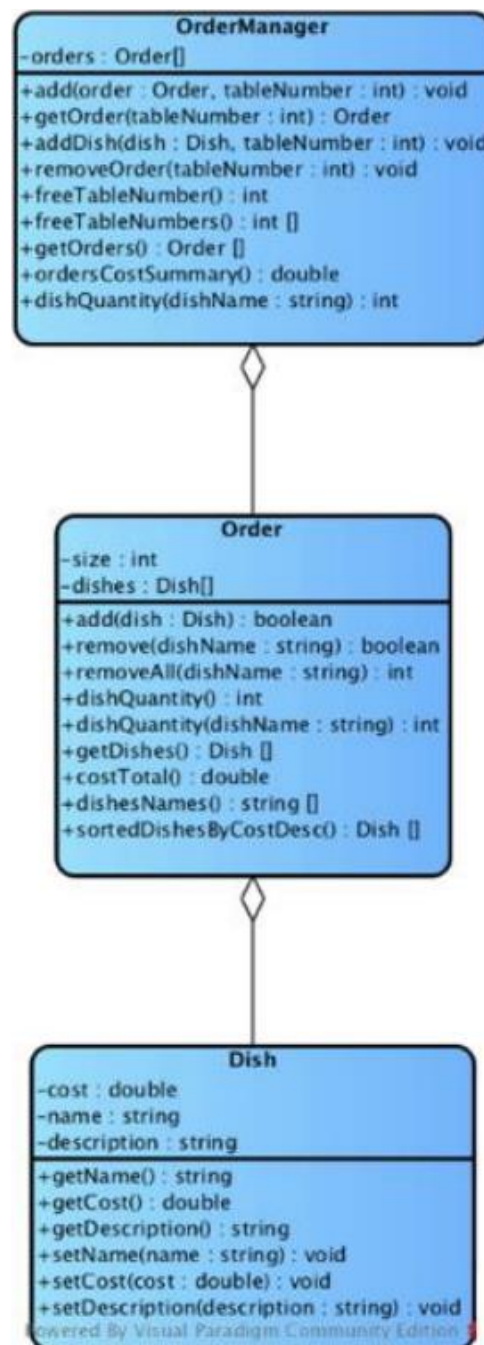


Рисунок 23.1 – Дополнительные требования к реализации практической работы



Решение:

1. Программа, реализующая работу класса Drink, представлена в листинге 23.1.

**Листинг 23.1 – Код программы класса Drink**

```
public final class Drink implements Item{
    private int coast;
    private String name;
    private String description;
    Drink(String name, String description){
        this.coast = 0;
        this.name = name;
        this.description = description;
    }
    Drink(int coast, String name, String description){
        this.coast = coast;
        this.name = name;
        this.description = description;
    }
    @Override
    public int getCoast() {
        return coast;
    }
    @Override
    public String getName() {
        return name;
    }
    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void setCoast(int coast) {
        this.coast = coast;
    }

    @Override
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void setDescription(String description) {
        this.description = description;
    }
}
```

2. Программы, реализующие работу интерфейса Item (листинг 23.2), классов Dish (листинг 23.3), Order (листинг 23.4).

**Листинг 23.2 – Код программы интерфейса Item**

```
public interface Item {
    public int getCoast();

    public String getName();
}
```

## Продолжение листинга 23.2

```
public String getDescription();  
public void setCoast(int coast);  
  
public void setName(String name);  
  
public void setDescription(String description);  
}
```

## Листинг 23.3 – Код программы класса Dish

```
public final class Dish implements Item{  
    private int coast;  
    private String name;  
    private String description;  
    Dish(String name,String description){  
        this.coast = 0;  
        this.name = name;  
        this.description = description;  
    }  
    Dish(int coast,String name,String description){  
        this.coast = coast;  
        this.name = name;  
        this.description = description;  
    }  
    @Override  
    public int getCoast() {  
        return coast;  
    }  
    @Override  
    public String getName() {  
        return name;  
    }  
    @Override  
    public String getDescription() {  
        return description;  
    }  
  
    @Override  
    public void setCoast(int coast) {  
        this.coast = coast;  
    }  
  
    @Override  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public void setDescription(String description) {  
        this.description = description;  
    }  
    public String toString(){  
        return "Coast = " + getCoast() + ", name = " + getName() + ",  
description = " + getDescription() + "\n";  
    }  
}
```

## Листинг 23.4 – Код программы класса Order

```
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.stream.Collectors;
```

## Продолжение листинга 23.4

```
public class Order {
    private List<Item> items;
    private int size;
    private Dish[] dishes;

    Order(int size){
        this.items = new ArrayList<>();
    }
    public boolean add(Dish dish){
        return items.add(dish);
    }
    public boolean remove(String Name){
        for(int i = 0; i < items.size();i++){
            if(items.get(i).getName().equals(Name)){
                items.remove(i);
                return true;
            }
        }
        return false;
    }
    public int removeAll(String Name){
        int count = 0;
        for (int i = 0; i < items.size();i++){
            Item temp = items.get(i);
            if (temp.getName().equals(Name)){
                count++;
                items.remove(i);
            }
        }
        return count;
    }
    public int dishQuantity(){
        return items.size();
    }
    public int dishQuantity(String Name){
        return (int) items.stream().filter(n->n.getName().equals(Name)).count();
    }

    public Dish[] getDishes() {
        return items.toArray(new Dish[0]);
    }
    public double costTotal(){
        return items.stream().mapToDouble(Item::getCoast).sum();
    }
    public String[] dishesName(){
        return new String[]{items.stream().filter(n -> n.getName() !=
        "").toArray().toString()};
    }
    public Dish[] sortedDishesByCostDesc(){
        return
        items.stream().sorted(Comparator.comparingInt(Item::getCoast)).collect(Collectors.toList()).toArray(new Dish[0]);
    }
}
```

3. Программы, реализующие работу класса `InternetOrder` (листинг 23.5) и тестового класса `Test` (листинг 23.6). Результат работы тестового класса представлен на рисунке 23.1.

#### Листинг 23.5 – Код программы класса `InternetOrder`

```
public class InternetOrder {
    private Order[] orders;
    public Order getOrder(String name){
        return new Order(1);
    }
    public void addDish(Dish dish,String name){
        return;
    }
    public boolean removeOrder(String name){
        return true;
    }
    public int freeTableNumber(){
        return 0;
    }
    public int[] freeTableNumbers(){
        return new int[]{};
    }
    public Order[] getOrders(){
        return new Order[]{};
    }
    public double orderCoastSummary(){
        return 0.0;
    }
    public int dishQuantity(String dishName){
        return 0;
    }
}
```

#### Листинг 23.6 – Код программы тестового класса `Test`

```
import java.util.Arrays;
import java.util.stream.Collectors;

public class Test {
    public static void main(String[] args) {
        Order order = new Order(5);
        order.add(new Dish(230, "Харчо", "Горячее блюдо"));
        order.add(new Dish(230, "Котлеты с пюре", "Горячее блюдо"));
        order.add(new Dish(150, "Салат овощной", "Холодное блюдо"));
        order.add(new Dish(50, "Чай", "Напиток"));
        order.add(new Dish(200, "Суп-пюре", "Горячее блюдо"));
        System.out.println(order.remove("Шашлык из баранины"));
        System.out.println(order.dishQuantity());
        System.out.println(order.remove("Котлеты с пюре"));
        System.out.println(order.dishQuantity());
        System.out.println(order.costTotal());
        System.out.println(order.dishQuantity("Чай"));

        System.out.println(Arrays.stream(order.sortedDishesByCostDesc()).collect(Collectors.toList()).toString());
    }
}
```

```
false
5
true
4
630.0
1
[Coast = 50, name = Чай, description = Напиток
, Coast = 150, name = Салат овощной, description = Холодное блюдо
, Coast = 200, name = Суп-пюре, description = Горячее блюдо
, Coast = 230, name = Харчо, description = Горячее блюдо
]
Process finished with exit code 0
```

Рисунок 23.1 – Результат работы тестового класса

### **Выводы по работе:**

Во время выполнения работы были закреплены навыки разработки программа на языке Java.

## Практическая работа №24

### Цель работы

Закрепить навыки разработки программ на языке Java.

### Выполнение лабораторной работы

#### Задание:

1. Переименуйте класс `Order` из предыдущего задания в `RestaurantOrder`. Создайте интерфейс `Order` – позиции заказа. Интерфейс должен определять следующие методы:

- добавления позиции в заказ (принимает ссылку типа `Item`), при этом возвращает логическое значение;
- удаляет позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Возвращает логическое значение;
- удаляет все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов;
- возвращает общее число позиций заказа в заказе;
- возвращает массив позиций заказа;
- возвращает общую стоимость заказа;
- возвращает число заказанных блюд или напитков (принимает название в качестве параметра);
- возвращает массив названий заказанных блюд и напитков (без повторов);
- возвращает массив позиций заказа, отсортированный по убыванию. цены.

Замечание: классы `InternetOrder` и `RestaurantOrder` должны реализовывать интерфейс `Order`.

2. Переименуйте класс `TablesOrderManager` в `OrderManager`. Добавьте ему еще одно поле типа `java.util.HashMap`, которое содержит пары адрес-заказ, и методы (работающие с этим полем). Методы класса:

- перегрузка метода добавления заказа. В качестве параметров принимает строку – адрес и ссылку на заказ.
- перегрузка метода получения заказа. В качестве параметра принимает строку – адрес;
- перегрузка метода удаления заказа. В качестве параметра принимает строку – адрес заказа;
- перегрузка метода добавления позиции к заказу. В качестве параметра принимает адрес и Item.
- возвращающий массив имеющихся на данный момент интернетзаказов.
- возвращающий суммарную сумму имеющихся на данный момент интернет-заказов.
- возвращающий общее среди всех интернет-заказов количество заказанных порций заданного блюда по его имени. Принимает имя блюда в качестве параметра.

Методы должны работать с интерфейсными ссылками Order и Item.

3. Создайте объявляемое исключение `OrderAlreadyAddedException`, выбрасываемое при попытке добавить заказ столику или по адресу, если со столиком или адресатом уже связан заказ.

Конструктор классов `Drink` и `Dish` должен выбрасывать исключение `java.lang.IllegalArgumentException` при попытке создать блюдо или напиток со стоимостью меньше 0, без имени или описания (если параметры имя и описание - пустые строки).

Создайте не объявляемое исключение `IllegalTableNumber`, выбрасываемое в методах, принимающих номер столика в качестве параметра, если столика с таким номером не существует. На рисунке 24.1 представлена диаграмма с примером реализации функционала ресторана.

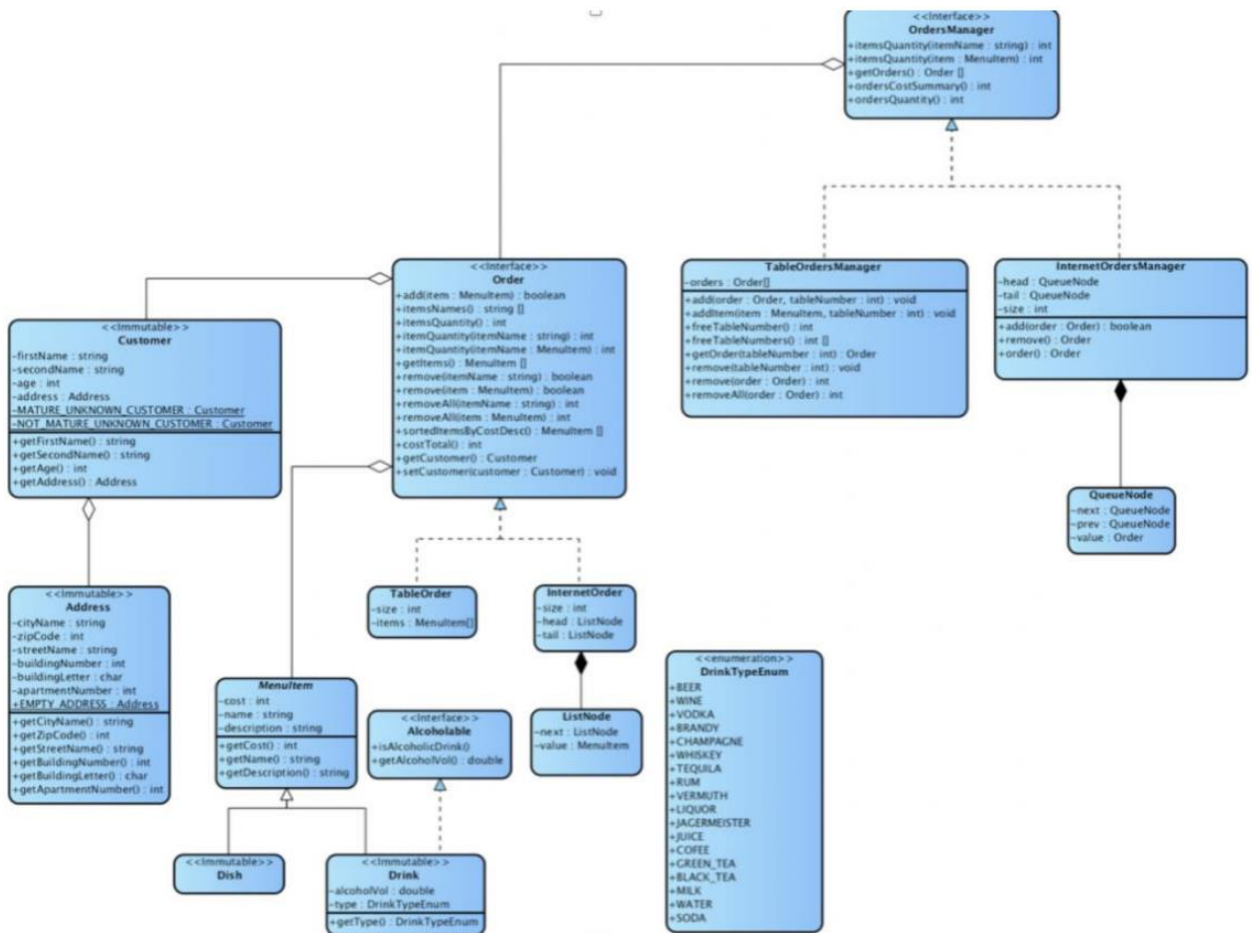


Рисунок 24.1 – Пример реализации функционала ресторана

### Решение:

Программы, реализующие работу интерфейсов AlchohTable (листинг 24.1), Order (листинг 24.2), OrdersManager (листинг 24.3), классов Address (листинг 24.4), Customer (листинг 24.5), Dish (листинг 24.6), Drink (листинг 24.7), DrinkTypeEnum (листинг 24.8), InternetOrder (листинг 24.9), InternetOrdersManager (листинг 24.10), QueueNode (листинг 24.11), MenuItem (листинг 24.12), TableOrder (листинг 24.13), TableOrdersManager (листинг 24.14), тестового класса Test (листинг 24.15). Результат работы программы представлен на рисунке 24.2.

### Листинг 24.1 – Код программы интерфейса AlchohTable

```

interface AlchohTable {

    public boolean isAlcoholicDrink();
    public double getAlcoholVol();
}

```



## Листинг 24.2 – Код программы интерфейса Order

```
public interface Order extends OrdersManager{
    public boolean add(MenuItem item);
    public String[] itemsNames();
    public int itemsQuantity();
    public int itemsQuantity(String itemName);
    public int itemsQuantity(MenuItem itemName);
    public MenuItem[] getItems();
    public boolean remove(MenuItem item);
    public int removeAll(String itemName);
    public int removeAll(MenuItem item);
    public boolean remove(String itemName);
    public int costTotal();
    public Customer getCustomer();
    public Customer setCustomer(Customer customer);
}
```

## Листинг 24.3 – Код программы интерфейса OrdersManager

```
import java.util.HashMap;

interface OrdersManager{
    HashMap<String,Order> orderManager = new HashMap<>();
    public int itemsQuantity(String itemName);
    public int itemsQuantity(MenuItem item);
    public Order[] getOrders();
    public int ordersCostSummary();
    public int ordersQuantity();
}
```

## Листинг 24.4 – Код программы класса Address

```
public final class Address extends Customer{
    private String cityName;
    private int zipCode;
    private String streetName;
    private int buildingNumber;
    private int buildingLetter;
    private int apartmentNumber;
    public Address EMPTY_ADDRESS;

    public String getCityName() {
        return cityName;
    }

    public int getZipCode() {
        return zipCode;
    }

    public String getStreetName() {
        return streetName;
    }

    public int getBuildingNumber() {
        return buildingNumber;
    }

    public int getBuildingLetter() {
        return buildingLetter;
    }

    public int getApartmentNumber() {
        return apartmentNumber;
    }
}
```

## Листинг 24.5 – Код программы класса Customer

```
public class Customer implements Order {
    private String firstName;
    private String secondName;
    private int age;
    private Address address;
    private Customer MATURE_UNKNOWN_CUSTOMER;
    private Customer NOT_MATURE_UNKNOWN_CUSTOMER;

    public String getFirstName() {
        return firstName;
    }
    public String getSecondName() {
        return secondName;
    }
    public int getAge() {
        return age;
    }
    public Address getAddress() {
        return address;
    }
    @Override
    public boolean add(MenuItem item) {
        return false;
    }
    @Override
    public String[] itemsNames() {
        return new String[0];
    }
    @Override
    public int itemsQuantity() {
        return 0;
    }
    @Override
    public int itemsQuantity(String itemName) {
        return 0;
    }
    @Override
    public int itemsQuantity(MenuItem itemName) {
        return 0;
    }
    @Override
    public Order[] getOrders() {
        return new Order[0];
    }
    @Override
    public int ordersCostSummary() {
        return 0;
    }
    @Override
    public int ordersQuantity() {
        return 0;
    }
    @Override
    public MenuItem[] getItems() {
        return new MenuItem[0];
    }
    @Override
    public boolean remove(MenuItem item) {
        return false;
    }
    @Override
    public int removeAll(String itemName) {
```

## Продолжение листинга 24.5

```
        return 0;
    }
    @Override
    public int removeAll(MenuItem item) {
        return 0;
    }
    @Override
    public boolean remove(String itemName) {
        return false;
    }
    @Override
    public int costTotal() {
        return 0;
    }
    @Override
    public Customer getCustomer() {
        return null;
    }
    @Override
    public Customer setCustomer(Customer customer) {
        return null;
    }
}
```

## Листинг 24.6 – Код программы класса Dish

```
import java.util.Scanner;

public final class Dish extends MenuItem {
    private int coast;
    private String name;
    private String description;
    Dish(String name,String description){
        super();
        if (name.isEmpty()){
            throw new IllegalArgumentException("Название блюда должно быть не пустым");
        }
        else if (description.isEmpty()){
            throw new IllegalArgumentException("Описание должно быть не пустым");
        }
        else {
            this.coast = 0;
            this.name = name;
            this.description = description;
        }
    }
    Dish(int coast,String name,String description){
        super();
        if (name.isEmpty()){
            throw new IllegalArgumentException("Название блюда должно быть не пустым");
        }
        else if (description.isEmpty()){
            throw new IllegalArgumentException("Описание должно быть не пустым");
        }
        else if ( coast < 0){
            throw new IllegalArgumentException("Стоимость должны быть больше 0");
        }
    }
}
```

## Продолжение листинга 24.6

```
        }else {
            this.coast = coast;
            this.name = name;
            this.description = description;
        }
    }
    public Dish(double cost, Scanner name, Scanner opis) {
        super();
    }

    @Override
    public int getCost() {
        return coast;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public String getDescription() {
        return description;
    }
}
```

## Листинг 24.7 – Код программы класса Drink

```
final class Drink extends MenuItem implements AlchohTable {
    private double alcoholVol;
    private DrinkTypeEnum type;
    private int coast;
    private String name;
    private String description;

    public DrinkTypeEnum getType() {
        return type;
    }
    Drink(String name,String description){
        if (name.isEmpty()){
            throw new IllegalArgumentException("Название блюда должно быть не пустым");
        }
        else if (description.isEmpty()){
            throw new IllegalArgumentException("Описание должно быть не пустым");
        }
        }else {
            this.coast = 0;
            this.name = name;
            this.description = description;
        }
    }
    Drink(int coast,String name,String description){
        if (name.isEmpty()){
            throw new IllegalArgumentException("Название блюда должно быть не пустым");
        }
        else if (description.isEmpty()){
            throw new IllegalArgumentException("Описание должно быть не пустым");
        }
    }
}
```

## Продолжение листинга 24.7

```
    }
    else if ( coast < 0){
        throw new IllegalArgumentException("Стоимость должны быть больше
0");

    }else {
        this.coast = coast;
        this.name = name;
        this.description = description;
    }
}
Drink(int coast,String name,String description, int
alcoholVol,DrinkTypeEnum type){
    if (name.isEmpty()){
        throw new IllegalArgumentException("Название блюда должно быть
не пустым");
    }
    else if (description.isEmpty()){
        throw new IllegalArgumentException("Описание должно быть не
пустым");
    }
    else if ( coast < 0){
        throw new IllegalArgumentException("Стоимость должны быть больше
0");

    }else {
        this.coast = coast;
        this.name = name;
        this.description = description;
        this.alcoholVol = alcoholVol;
        this.type = type;
    }
}

@Override
public boolean isAlcoholicDrink() {
    return false;
}

@Override
public double getAlcoholVol() {
    return 0;
}

@Override
public int getCost() {
    return coast;
}

@Override
public String getName() {
    return name;
}

@Override
public String getDescription() {
    return description;
}
}
```

## Листинг 24.8 – Код программы класса DrinkTypeEnum

```
import java.util.Objects;
public class DrinkTypeEnum {
    public static String Proverb(String n){
        String t = "";
        if (Objects.equals(n, "AL")){
            t = "Предъявите паспорт";
        }
        return t;
    }
}
```

## Листинг 24.9 – Код программы класса InternetOrder

```
import java.util.stream.Collectors;

public class InternetOrder implements Order{
    private int size;
    private ListNode head;
    private ListNode tail;
    class ListNode{
        private ListNode next;
        private ListNode prev;
        private MenuItem value;
        ListNode(MenuItem value){
            this.next = null;
            this.prev = null;
            this.value = value;
        }
    }
    InternetOrder(int size){
        this.tail = null;
        this.head = null;
        this.size = size;
    }
    @Override
    public boolean add(MenuItem item) {
        ListNode node = new ListNode(item);
        if(head == null){
            head = node;
            tail = node;
            return true;
        }else{
            node.next = head;
            head.prev = node;
            head = node;
            return true;
        }
    }
    @Override
    public String[] itemsNames() {
        String str[] = new String[size];
        ListNode temp = head;
        int i = 0;
        while (temp != null & i < size){
            str[i] = temp.value.getName();
            i++;
            temp = temp.next;
        }
        return str;
    }
    @Override
```

## Продолжение листинга 24.9

```
public int itemsQuantity() {
    ListNode temp = head;
    int count = 0;
    while (temp != null) {
        count++;
        temp = temp.next;
    }
    return count;
}

@Override
public int itemsQuantity(String itemName) {
    ListNode temp = head;
    int count = 0;
    while (temp != null) {
        if (temp.value.getName().equals(itemName)) {
            count++;
        }
        temp = temp.next;
    }
    return count;
}

@Override
public int itemsQuantity(MenuItem itemName) {
    ListNode temp = head;
    int count = 0;
    while (temp != null) {
        if (temp.value.equals(itemName)) {
            count++;
        }
        temp = temp.next;
    }
    return count;
}

@Override
public Order[] getOrders() {
    return orderManager.values().toArray(new Order[0]);
}

@Override
public int ordersCostSummary() {
    int sum = 0;
    ListNode temp = head;
    while (temp != null) {
        sum += temp.value.getCost();
        temp = temp.next;
    }
    return sum;
}

@Override
public int ordersQuantity() {
    return orderManager.size();
}

@Override
public MenuItem[] getItems() {
    return orderManager.values().toArray(new MenuItem[0]);
}

@Override
```

## Продолжение листинга 24.9

```
public boolean remove(MenuItem item) {
    if (head == null){
        return false;
    }
    if (head == tail && head.value.equals(item)){
        head = null;
        tail = null;
        return true;
    }
    ListNode temp = head;
    while (temp != null & temp.value.equals(item)){
        temp = temp.next;
    }
    temp.prev.next = temp.next;
    temp.next.prev = temp.prev;
    temp.prev = null;
    temp.next = null;
    return true;
}

@Override
public int removeAll(String itemName) {
    if (head == null){
        return 0;
    }
    if (head == tail & head.value.getName().equals(itemName)){
        head = null;
        tail = null;
        return 1;
    }
    int count = 0;
    ListNode temp = head;
    while (temp != null){
        if(temp.value.getName().equals(itemName)){
            count++;
            temp.prev.next = temp.next;
            temp.next.prev = temp.prev;
            temp.prev = null;
            temp.next = null;
        }
        temp = temp.next;
    }
    return count;
}

@Override
public int removeAll(MenuItem item) {
    if (head == null){
        return 0;
    }
    if (head == tail & head.value.equals(item)){
        head = null;
        tail = null;
        return 1;
    }
    int count = 0;
    ListNode temp = head;
    while (temp != null){
        if(temp.value.equals(item)){
            count++;
            temp.prev.next = temp.next;
            temp.next.prev = temp.prev;
        }
    }
    return count;
}
```



## Продолжение листинга 24.9

```
        temp.prev = null;
        temp.next = null;
    }
    temp = temp.next;
}
return count;
}

@Override
public boolean remove(String itemName) {
    if (head == null) {
        return false;
    }
    if (head == tail && head.value.getName().equals(itemName)) {
        head = null;
        tail = null;
        return true;
    }
    ListNode temp = head;
    while (temp != null & temp.value.getName().equals(itemName)) {
        temp = temp.next;
    }
    temp.prev.next = temp.next;
    temp.next.prev = temp.prev;
    temp.prev = null;
    temp.next = null;
    return true;
}

@Override
public int costTotal() {
    orderManager.values().stream().filter(value ->
orderManager.containsValue(value)).collect(Collectors.toList());
    return 0;
}

@Override
public Customer getCustomer() {
    return null;
}

@Override
public Customer setCustomer(Customer customer) {
    return null;
}
}
```

## Листинг 24.10 – Код программы класса InternetOrdersManager

```
public class InternetOrdersManager implements OrdersManager {
    private QueueNode head;
    private QueueNode tail;
    private int size;
    public boolean add(Order order) {
        return true;
    }
    public Order remove() {
        return null;
    }
    public Order order() {
        return null;
    }
}
```

## Продолжение листинга 24.10

```
}  
@Override  
public int itemsQuantity(String itemName) {  
    return 0;  
}  
  
@Override  
public int itemsQuantity(MenuItem item) {  
    return 0;  
}  
  
@Override  
public Order[] getOrders() {  
    return new Order[0];  
}  
  
@Override  
public int ordersCostSummary() {  
    return 0;  
}  
  
@Override  
public int ordersQuantity() {  
    return 0;  
}  
}
```

## Листинг 24.11 – Код программы класса QueueNode

```
class QueueNode{  
    private QueueNode next;  
    private QueueNode prev;  
    private Order value;  
}
```

## Листинг 24.12 – Код программы класса MenuItem

```
import java.util.Map;  
  
public class MenuItem implements Order{  
    private int cost;  
    private String name;  
    private String description;  
  
    public int getCost(){  
        return cost;  
    }  
  
    public String getName(){  
        return name;  
    }  
  
    public String getDescription(){  
        return description;  
    }  
  
    @Override  
    public boolean add(MenuItem item) {  
        orderManager.put(item.getName(), item);  
        return true;  
    }  
  
    @Override
```

## Продолжение листинга 24.12

```
public String[] itemsNames() {
    return orderManager.keySet().stream().filter(key ->
orderManager.get(key) != null).toArray(String[]::new);
}

@Override
public int itemsQuantity() {
    return orderManager.keySet().size();
}

@Override
public int itemsQuantity(String itemName) {
    return (int) orderManager.keySet().stream().filter(key ->
key.equals(itemName)).count();
}

@Override
public int itemsQuantity(MenuItem itemName) {
    int count = 0;
    for (Map.Entry<String, Order> entry : orderManager.entrySet()) {
        Order order = entry.getValue();
        if (order.equals(itemName)) {
            count++;
        }
    }
    return count;
}

@Override
public Order[] getOrders() {
    return orderManager.values().toArray(new Order[0]);
}

@Override
public int ordersCostSummary() {
    return 0;
}

@Override
public int ordersQuantity() {
    return orderManager.size();
}

@Override
public MenuItem[] getItems() {
    return orderManager.values().toArray(new MenuItem[0]);
}

@Override
public boolean remove(MenuItem item) {
    return orderManager.remove(item.getName(), item);
}

@Override
public int removeAll(String itemName) {
    int initialSize = orderManager.size(); // Запоминаем начальный
размер HashMap
    orderManager.entrySet().removeIf(entry ->
entry.getKey().equals(itemName));
    int countDel = initialSize - orderManager.size(); // Вычисляем
количество удаленных элементов
}
```

## Продолжение листинга 24.12

```
        return countDel;
    }

    @Override
    public int removeAll(MenuItem item) {
        int count = 0;
        for(int i = 0 ; i < orderManager.size();i++){
            if(orderManager.remove(item.name,item)) {
                count++;
            }
        }
        return count;
    }

    @Override
    public boolean remove(String itemName) {
        return orderManager.remove(itemName) != null;
    }

    @Override
    public int costTotal() {
        return 0;
    }

    @Override
    public Customer getCustomer() {
        return null;
    }

    @Override
    public Customer setCustomer(Customer customer) {
        return null;
    }

    @Override
    public String toString() {
        return "cost = " + cost + ", name = " + name + ", description = " +
description;
    }
    public String prints(){
        return orderManager.toString();
    }
}
```

## Листинг 24.13 – Код программы класса TableOrder

```
import java.util.Map;

public class TableOrder implements Order{
    private int size;
    private MenuItem[] items;

    @Override
    public boolean add(MenuItem item) {
        orderManager.put(item.getName(),item);
        return true;
    }

    @Override
    public String[] itemsNames() {
        return orderManager.keySet().stream().filter(key ->
orderManager.get(key) != null).toArray(String[]::new);
    }

    @Override
```

## Продолжение листинга 24.13

```
public int itemsQuantity() {
    return orderManager.keySet().size();
}

@Override
public int itemsQuantity(String itemName) {
    return (int) orderManager.keySet().stream().filter(key ->
key.equals(itemName)).count();
}

@Override
public int itemsQuantity(MenuItem itemName) {
    int count = 0;
    for (Map.Entry<String, Order> entry : orderManager.entrySet()) {
        Order order = entry.getValue();
        if (order.equals(itemName)) {
            count++;
        }
    }
    return count;
}

@Override
public Order[] getOrders() {
    return orderManager.values().toArray(new Order[0]);
}

@Override
public int ordersCostSummary() {
    return 0;
}

@Override
public int ordersQuantity() {
    return orderManager.size();
}

@Override
public MenuItem[] getItems() {
    return orderManager.values().toArray(new MenuItem[0]);
}

@Override
public boolean remove(MenuItem item) {
    return orderManager.remove(item.getName(), item);
}

@Override
public int removeAll(String itemName) {
    int initialSize = orderManager.size(); // Запоминаем начальный
размер HashMap
    orderManager.entrySet().removeIf(entry ->
entry.getKey().equals(itemName));
    int countDel = initialSize - orderManager.size(); // Вычисляем
количество удаленных элементов
    return countDel;
}

@Override
public int removeAll(MenuItem item) {
    int count = 0;
    for (int i = 0 ; i < orderManager.size(); i++) {
        if (orderManager.remove(item.getName(), item)) {
            count++;
        }
    }
}
```

## Продолжение листинга 24.13

```
    }  
    }  
    return count;  
}  
  
@Override  
public boolean remove(String itemName) {  
    return orderManager.remove(itemName) != null;  
}  
  
@Override  
public int costTotal() {  
    return 0;  
}  
  
@Override  
public Customer getCustomer() {  
    return null;  
}  
  
@Override  
public Customer setCustomer(Customer customer) {  
    return null;  
}  
}
```

## Листинг 24.14 – Код программы класса TableOrdersManager

```
public class TableOrdersManager implements OrdersManager{  
    private Order[] orders;  
    public void add(Order order, int tableNumber){}  
    public void addItem(MenuItem item, int tableNumber){}  
    public int freeTableNumber(Order order, int tableNumber){ return 0;}  
    public int[] freeTableNumbers(Order order, int tableNumber){ return  
null;}  
  
    public Order getOrder(int tableNumber){ return null;}  
    public void remove(int tableNumber){}  
    public int remove(Order order){ return 0;}  
    public int removeAll(Order order){return 0;}  
  
    @Override  
    public int itemsQuantity(String itemName) {  
        return 0;  
    }  
    @Override  
    public int itemsQuantity(MenuItem item) {  
        return 0;  
    }  
    @Override  
    public Order[] getOrders() {  
        return new Order[0];  
    }  
    @Override  
    public int ordersCostSummary() {  
        return 0;  
    }  
    @Override  
    public int ordersQuantity() {  
        return 0;  
    }  
}
```

## Листинг 24.15 – Код программы тестового класса Test

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Test {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        Scanner r = new Scanner(System.in);
        Scanner w = new Scanner(System.in);
        MenuItem menuItem = new MenuItem();
        double cost;
        boolean bool = true;
        String name, opis;
        Scanner nomder = new Scanner(System.in);
        System.out.println("Введите 1, если вы делаете заказ в ресторане:\n"
+
            "Введите 2, если делаете заказ на сайте:\n" +
            "Введите 3, если хотите популярные блюда ресторана:");
        switch (nomder.nextInt()) {
            case 1:
                while (bool) {
                    System.out.println("Что хотите добавить в заказ?
\nВведите 1, если еду: \nВведите 2, если напиток:");
                    switch (nomder.nextInt()) {
                        case 1:
                            System.out.print("Введите название блюда: ");
                            name = r.nextLine();
                            System.out.print("Введите цену блюда: ");
                            cost = s.nextDouble();
                            System.out.print("Введите описание блюда: ");
                            opis = w.nextLine();
                            Dish d = new Dish((int) cost, name, opis);
                            menuItem.add(d);
                            System.out.println("Введите 1, если вы хотите
заказать еще, или 0, если вы закончили с заказом");
                            if (nomder.nextInt() == 1){
                                bool = true;
                                break;
                            }else {
                                bool=false;
                                break;
                            }
                        case 2:
                            System.out.print("Введите название напитка: ");
                            name = r.nextLine();
                            System.out.print("Введите цену напитка: ");
                            cost = s.nextDouble();
                            System.out.print("Введите описание напитка если
он алкагольный введите AL: ");
                            opis = w.nextLine();
                            Drink dr = new Drink((int) cost, name, opis);
                            menuItem.add(dr);
                            System.out.println(DrinkTypeEnum.Proverb(opis));
                            System.out.println("Ввелите 1 если вы хотите
заказать еще или 0 если вы закончили с заказом");
                            if (nomder.nextInt() == 1){
                                bool = true;
                                break;
                            }else {
                                bool=false;
                                break;
                            }
                    }
                }
            case 3:
                // ... (code for case 3 is not fully visible in the image)
        }
    }
}
```

## Продолжение листинга 24.15

```

    }
    }
    break;
case 2:
    while (bool) {
        System.out.println("Что хотите добавить в заказ?
\nВведите 1, если еду: \nВведите 2, если напиток:");
        switch (nomder.nextInt()) {
            case 1:
                System.out.print("Введите название блюда: ");
                name = r.nextLine();
                System.out.print("Введите цену блюда: ");
                cost = s.nextDouble();
                System.out.print("Введите описание блюда: ");
                opis = w.nextLine();
                Dish d = new Dish((int) cost, name, opis);
                menuItem.add(d);
                System.out.println("Введите 1, если вы хотите
заказать еще, или 0, если вы закончили с заказом");
                if (nomder.nextInt() == 1){
                    bool = true;
                    break;
                }else {
                    bool=false;
                    break;
                }
            case 2:
                System.out.print("Введите название напитка: ");
                name = r.nextLine();
                System.out.print("Введите цену напитка: ");
                cost = s.nextDouble();
                System.out.print("Введите описание напитка если
он алкагольный введите AL: ");
                opis = w.nextLine();
                Drink dr = new Drink((int) cost, name, opis);
                menuItem.add(dr);
                System.out.println(DrinkTypeEnum.Proverc(opis));
                System.out.println("Введите 1, если вы хотите
заказать еще, или 0, если вы закончили с заказом");
                if (nomder.nextInt() == 1){
                    bool = true;
                    break;
                }else {
                    bool=false;
                    break;
                }
            }
        }
    }
    break;
case 3:
    Dish dishOne = new Dish(280,"Манты","Горячее блюдо");
    Dish dishTwo = new Dish(250,"Борщ","Горячее блюдо");
    Dish dishFree = new Dish(350,"Лазанья","Горячее блюдо");
    Drink drink = new Drink(250,"Салат Цезарь","Холодное
блюдо");

    menuItem.add(dishOne);
    menuItem.add(dishTwo);
    menuItem.add(dishFree);
    menuItem.add(drink);
}

```



## Продолжение листинга 24.15

```
System.out.println(Arrays.stream(menuItem.itemsNames()).collect(Collectors.toList()));
    }
}
```

```
Введите 1, если вы делаете заказ в ресторане:
Введите 2, если делаете заказ на сайте:
Введите 3, если хотите популярные блюда ресторана:
1
Что хотите добавить в заказ?
Введите 1, если еду:
Введите 2, если напиток:
1
Введите название блюда: Борщ
Введите цену блюда: 250
Введите описание блюда: Горячее блюдо
Введите 1, если вы хотите заказать еще, или 0, если вы закончили с заказом
1
Что хотите добавить в заказ?
Введите 1, если еду:
Введите 2, если напиток:
2
Введите название напитка: Водка
Введите цену напитка: 150
Введите описание напитка если он алкогольный введите AL: AL
Предъявите паспорт
Введите 1 если вы хотите заказать еще или 0 если вы закончили с заказом
0
[Водка, Борщ]

Process finished with exit code 0
```

Рисунок 24.2 – Результат работы тестового класса Test

### Выводы по работе:

Во время выполнения работы были закреплены навыки разработки программ.

## ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. Конспект лекций по дисциплине «Программирование на языке Джава», РТУ МИРЭА, лектор – старший преподаватель Зорина Н.В.
2. Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. –СПб.: Питер, 2013. – 240 с.
3. Фрэйз Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств – Питер: 2016г. Режим доступа свободный: [https://www.htbook.ru/kompjutery\\_i\\_seti/setevye\\_tekhnologii/html5-i-css3-razrabotka-sajtov-dlja-ljubyx-brauzerov-i-ustrojstv](https://www.htbook.ru/kompjutery_i_seti/setevye_tekhnologii/html5-i-css3-razrabotka-sajtov-dlja-ljubyx-brauzerov-i-ustrojstv);
4. Справочник по языку PHP [Электронный ресурс]:php.su— Режим доступа свободный: <http://www.php.su>;