



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий

КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиПО)

ПРАКТИЧЕСКИЕ РАБОТЫ
ПО ДИСЦИПЛИНЕ «Программирование на языке Джава»

Выполнил студент группы ИКБО-13-22

Руденко А. Д.

Принял старший преподаватель

Матчин В. Т.

Практические работы работа выполнены «__»____ 2023г.

«Зачтено» «__»____ 2023г.

Москва 2023

Практическая работа № 1

Цель работы

Освоить на практике работу с классами языка программирования Java.

Теоретическое введение

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

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

Выполнение лабораторной работы

Задание

- 1) реализуйте простейший класс «Собака»;
- 2) реализуйте простейший класс «Мяч»;
- 3) реализуйте простейший класс «Книга».

Решение

Реализация класса «Собака» представлена на рисунке 1.1

```

6 usages  ± mrfels1 *
1 public class Dog {
    // usages
2     private String name; private int age;
    // 2 usages new *
3     public Dog(String n, int a) {
4         name = n;
5         age = a;
        new *
6     }
7     public Dog(String n) {
8         name = n;
9         age = 0;
        new *
10    }
11    public Dog() {
12        name = "Pup";
13        age = 0;
        new *
14    }
15    public void setAge(int age) {
16        this.age = age;
        new *
17    }
18    public void setName(String name) {
19        this.name = name;
        new *
20    }
21    public String getName(String name) {
22        return name;
        new *
23    }
24    public int getAge() {
25        return age;
        new *
26    }
27    public String toString() {
28        return this.name + ", age " + this.age;
        new *
29    }
30    public void intoHumanAge() {
31        System.out.println(name + "'s age in human years is " + age * 7 + " years");
32    }
33 }

```

Рисунок 1.1 – Код класса «Собака»

Реализация класса «Мяч» представлена на рисунке 1.2

```

4 usages  ± mrfels1 *
1 public class Ball {
    no usages new *
2     public String getColor() {
3         return color;
        new *
4     }
5     public void setColor(String color) {
6         this.color = color;
7     }
8     private String color; private double radius;
    // 2 usages new *
9     public Ball(String color, double radius) {
10        this.color = color;
11        this.radius = radius;
        new *
12    }
13    public double getRadius() {
14        return radius;
        new *
15    }
16    public void setRadius(double radius) {
17        this.radius = radius;
        new *
18    }
19    public double calculateVolume() {
20        return (4.0 / 3.0) * Math.PI * Math.pow(radius, 3);
        new *
21    }
22    public String toString() {
23        return "A " + color + " ball with a radius of " + radius + " units.";
24    }
25 }

```

Рисунок 1.2 – Код класса «Мяч»

Реализация класса «Книга» представлена на рисунке 1.3



```
1 public class Book {  
2     private String title; private String author; private int pages;  
3     public Book(String title, String author, int pages) {  
4         this.title = title;  
5         this.author = author;  
6         this.pages = pages;  
7     }  
8     public String getTitle() {  
9         return title;  
10    }  
11    public void setTitle(String title) {  
12        this.title = title;  
13    }  
14    public String getAuthor() {  
15        return author;  
16    }  
17    public void setAuthor(String author) {  
18        this.author = author;  
19    }  
20    public int getPages() {  
21        return pages;  
22    }  
23    public void setPages(int pages) {  
24        this.pages = pages;  
25    }  
26    public String toString() {  
27        return "Title: " + title + ", Author: " + author + ", Pages: " + pages;  
28    }  
29 }
```

Рисунок 1.3 – Код класса «Книга»

Выводы по работе:

В ходе работы были изучены основы языка Java, работа с классами и их методами.

Практическая работа № 2

Цель работы

Введение в разработку программ на языке программирования Java.

Теоретическое введение

Язык Джава — это объектно-ориентированный язык программирования, со строгой инкапсуляцией и типизацией. Программы, написанные на языке, Джава могут выполняться под управлением различных операционных системах при наличии необходимого ПО – Java Runtime Environment.

Для того чтобы создать и запускать программы на языке Джава необходимо следующее ПО:

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

Выполнение лабораторной работы

Задание:

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.

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.

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

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

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

Результаты выполнения практической работы залить через IDE в свой репозиторий и продемонстрировать преподавателю.

Решение:

1. Написание программы

```

public class Main {
    public static void main(String[] args) {
        int[] arr = {1,3,5,7,9};
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }
        float mean = (float)sum/arr.length;
    }
}

```

Рисунок 2.1 - Код программы

2. Написание программы

```

Scanner sc = new Scanner(System.in);
System.out.println("Введите размер массива: ");
int arr2length = 0;
if(sc.hasNextInt()) {
    arr2length = sc.nextInt();
}
int[] arr2 = new int[arr2length];
for(int i = 0; i < arr2.length; i++){
    System.out.println("Введите число: ");
    arr2[i] = sc.nextInt();
}
int sum2 = 0;
int i = 0;
int maxnum = Integer.MIN_VALUE;
int minnum = Integer.MAX_VALUE;
while(i < arr2.length) {
    sum2 += arr2[i];
    if (arr2[i] > maxnum){
        maxnum = arr2[i];
    }
    if(arr2[i] < minnum){
        minnum = arr2[i];
    }
    i++;
}
System.out.println("Sum = " + sum2 + " Max = " + maxnum + " Min = " + minnum);

```

Рисунок 2.2 - Код программы

3. Написание программы

```
for (int k = 0; k < args.length; k++) {  
    System.out.println("Аргумент " + k + ": " + args[k]); //Аргументов нет, ничего не выведется  
}
```

Рисунок 2.3 - Код программы

4. Написание программы

```
for (i = 1; i <= 10; i++) {  
    System.out.printf("1/" + i + " = " + (float) 1/i + "\n");  
}
```

Рисунок 2.4 - Код программы

5. Написание программы

```
System.out.printf("Factorial of 4 equals: " + factroialCounter.countFactorial( n: 4));
```

Рисунок 2.5 - Код программы

```
public class factroialCounter {  
    2 usages  ± mrfels1  
    public static int countFactorial(int n){  
        if (n <= 2) {  
            return n;  
        }  
        return n * countFactorial( n: n - 1);  
    }  
}
```

Рисунок 2.6 - Код метода класса

Выводы по работе:

В ходе работы были изучены основы языка Java, работа с классами и их методами.

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).

Имя (или сущность): определяет класс.

Выполнения лабораторной работы

Задание:

1. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 3.1

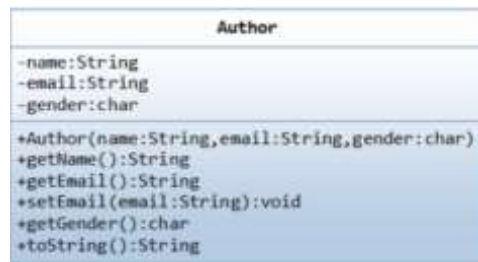


Рисунок 3.1 - Диаграмма класса Author

2. По UML диаграмме класса, представленной на рисунке 3.2. 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.

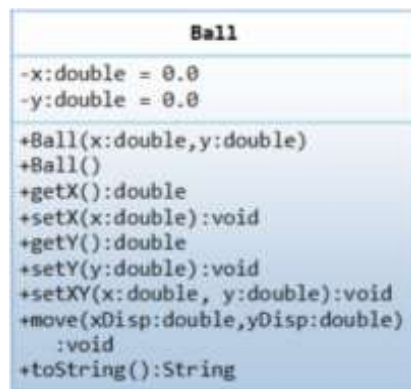


Рисунок 3.2 - Диаграмма класса Ball

Решение:

1. Разработка программы

Код класса Author представлен на рисунке 3.3:

```

package Test;

2 usages
public class Author {
    2 usages
    private String name;
    3 usages
    private String email;
    2 usages
    private char gender;
    1 usage
    public Author(String name, String email, char gender){
        this.email = email;
        this.name = name;
        this.gender = gender;
    }
    2 usages
    public String getName(){
        return this.name;
    }
    2 usages
    public String getEmail(){
        return this.email;
    }

    2 usages
    public char getGender() {
        return gender;
    }
    1 usage
    public void setEmail(String newEmail){
        this.email = newEmail;
    }
    @Override
    public String toString(){
        return ("Name = " + getName() +
            "; Email = " + getEmail() +
            "; Gender = " + getGender());
    }
}

```

Рисунок 3.3 – Код класса Author

На рисунке 3.4 представлен код класса TestAuthor:

```

package Test;

public class TestAuthor {
    public static void main(String[] args) {
        Author Pattinson = new Author( name: "Robert", email: "rob@gmail.com", gender: 'M');
        System.out.println("Author name: " + Pattinson.getName() +
            " Email: " + Pattinson.getEmail() +
            " Gender: " + Pattinson.getGender() + '\n');
        //Or use this
        Pattinson.setEmail("pattinson@gmail.com");
        System.out.println(Pattinson.toString());
    }
}

```

Рисунок 3.4 – Код класса TestAuthor

Результат выполнения программы задания 1 представлена на рисунке 3.5:

```

"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent
Author name: Robert Email: rob@gmail.com Gender: M

Name = Robert; Email = pattinson@gmail.com; Gender = M

Process finished with exit code 0

```

Рисунок 3.5 – Результат выполнения программы задания 1

2. Разработка программы

Код класса Ball представлен на рисунке 3.6:

```

package Test;

no usages
class Ball {
    6 usages
    private double x = 0.0;
    6 usages
    private double y = 0.0;
    no usages
    public Ball() {
    }
    no usages
    public Ball(double x, double y) {
        this.x = x;
        this.y = y;
    }
    no usages
    public double getX() {
        return x;
    }
    no usages
    public double getY() {
        return y;
    }
    no usages
    public void setX(double x) {
        this.x = x;
    }
    no usages
    public void setY(double y) {
        this.y = y;
    }
    no usages
    public void setXY(double x, double y) {
        this.x = x;
        this.y = y;
    }
    no usages
    public void move(double xDisp, double yDisp) {
        x += xDisp;
        y += yDisp;
    }
    @Override
    public String toString() {
        return "Ball @ (" + this.x + ", " + this.y + ").";
    }
}

```

Рисунок 3.6 – Код класса Ball

Код класса TestBall представлен на рисунке 3.7:

```
package Test;

public class TestBall {
    public static void main(String[] args) {
        Ball ball = new Ball( x: 100, y: 100);
        System.out.println(ball);
        ball.move( xDisp: 30, yDisp: 15);
        System.out.println(ball);
    }
}
```

Рисунок 3.7 – Код класса TestBall

Результат выполнения программы задания 2 представлена на рисунке 3.8:

```
"C:\Program Files\Java\jdk-20\bin\java.exe"
Ball @ (100.0, 100.0).
Ball @ (130.0, 115.0).

Process finished with exit code 0
```

Рисунок 3.8 – Результат выполнения программы задания 2

Выводы по работе:

В ходе работы были изучены основы работы с классами и их методами.

Практическая работа № 4

Цель работы

Цель данной работы - изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

Теоретическое введение

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

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:

Разработка класса Circle

На рисунке 4.1 представлен код класса Circle

```
1 public class Circle {
2     7 usages
3     private double radius; private String colour; private boolean isFilled;
4     no usages new *
5     Circle(){
6         this.radius = 0;
7         this.setColour("Blank");
8         this.setFilled(false);
9     }
10    new *
11    }Circle(double radius, String colour, boolean isFilled){
12        this.radius = radius;
13        this.setColour(colour);
14        this.setFilled(isFilled);
15    }
16    new *
17    }public double getRadius() { return radius; }
18    no usages new *
19    public void setRadius(double radius) { this.radius = radius; }
20    2 usages new *
21    public void setColour(String colour) {this.colour = colour; }
22    2 usages new *
23    public void setFilled(boolean filled) {isFilled = filled;}
24    1 usage new *
25    public double getArea() { return 3.14*pow(radius,2.0); }
26    1 usage new *
27    public double getPerimeter() { return 2*3.14*radius; }
28    new *
29    @Override
30    public String toString() {
31        return "Circle{" +
32            "radius=" + radius +
33            ", colour='" + colour + '\'' +
34            ", isFilled=" + isFilled +
35            ", perimeter=" + getPerimeter()+
36            ", area=" + getArea() +
37            '}';
38    }
39 }
```

Рисунок 4.1 – Код класса Circle

Разработка класса Human

На рисунках 4.2 и 4.3 представлен код класса Human

```
3 public class Human {
4     private Head head;
5     private Leg leftLeg; private Leg rightLeg;
6     private Hand leftHand; private Hand rightHand;
7     private boolean isAlive = false;
8     public Human(){
9         this.head = new Head();
10        this.leftHand = new Hand();
11        this.rightHand = new Hand();
12        this.leftLeg = new Leg();
13        this.rightLeg = new Leg();
14        isAlive = true;
15    }
16    public void doDamage(int bodyPart){
17        if(isAlive){
18            switch (bodyPart){
19                case 1:
20                    head.setHealth(head.getHealth() - 10);
21                    if (head.getHealth() <= 0){
22                        isAlive = false;
23                        System.out.println("Human got killed");
24                        return;
25                    }
26                    break;
27                case 2:
28                    if (leftLeg.getHealth() <= 0){
29                        System.out.println("Left leg is destroyed");
30                    }
31                    else {
32                        leftLeg.setHealth(leftLeg.getHealth() - 10);
33                    }
34                    break;
35                case 3:
36                    if (rightLeg.getHealth() <= 0){
```

Рисунок 4.2 – Код части класса Human


```

39         else {
40             rightLeg.setHealth(rightLeg.getHealth() - 10);
41         }
42         break;
43     case 4:
44         if (rightHand.getHealth() <= 0){
45             System.out.println("right hand is destroyed");
46         }
47         else {
48             rightHand.setHealth(rightHand.getHealth() - 10);
49         }
50         break;
51     case 5:
52         if (leftHand.getHealth() <= 0){
53             System.out.println("left hand is destroyed");
54         }
55         else {
56             leftHand.setHealth(leftHand.getHealth() - 10);
57         }
58         break;
59     }
60 }
61 else {
62     System.out.println("HE IS ALREADY DEAD STOP!");
63     return;
64 }
65 }
66 new *
67 @Override
68 public String toString() {
69     return "Human{" +
70         "head=" + head +
71         ", leftLeg=" + leftLeg +
72         ", rightLeg=" + rightLeg +
73         ", leftHand=" + leftHand +
74         ", rightHand=" + rightHand +
75         ", isAlive=" + isAlive +
76         "}";
77 }

```

Рисунок 4.3 – Код второй части класса Human

Разработка класса Book

На рисунке 4.4 представлен код класса Book

```
1 public class Book {  
    4 usages  
2     private String title; private String author; private int pages;  
    1 usage new *  
3     public Book(String title, String author, int pages) {  
4         this.title = title;  
5         this.author = author;  
6         this.pages = pages;  
    new *  
7     } public String getTitle() { return title; }  
    no usages new *  
10    public void setTitle(String title) { this.title = title; }  
    no usages new *  
13    public String getAuthor() { return author; }  
    no usages new *  
16    public void setAuthor(String author) { this.author = author; }  
    no usages new *  
19    public int getPages() { return pages; }  
    no usages new *  
22    public void setPages(int pages) { this.pages = pages; }  
    new *  
25    public String toString() { return "Title: " + title + ", Author: " + author + ", Pages: " + pages; }  
28 }
```

Рисунок 4.4 – Код класса Book

Выводы по работе:

В ходе работы были изучены основы работы с классами и их методами.

Практическая работа № 5

Цель работы

Освоить 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.

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе

Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Выполнения лабораторной работы

Задание

- 1) создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы);

- 2) перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square рисунка 3.1.

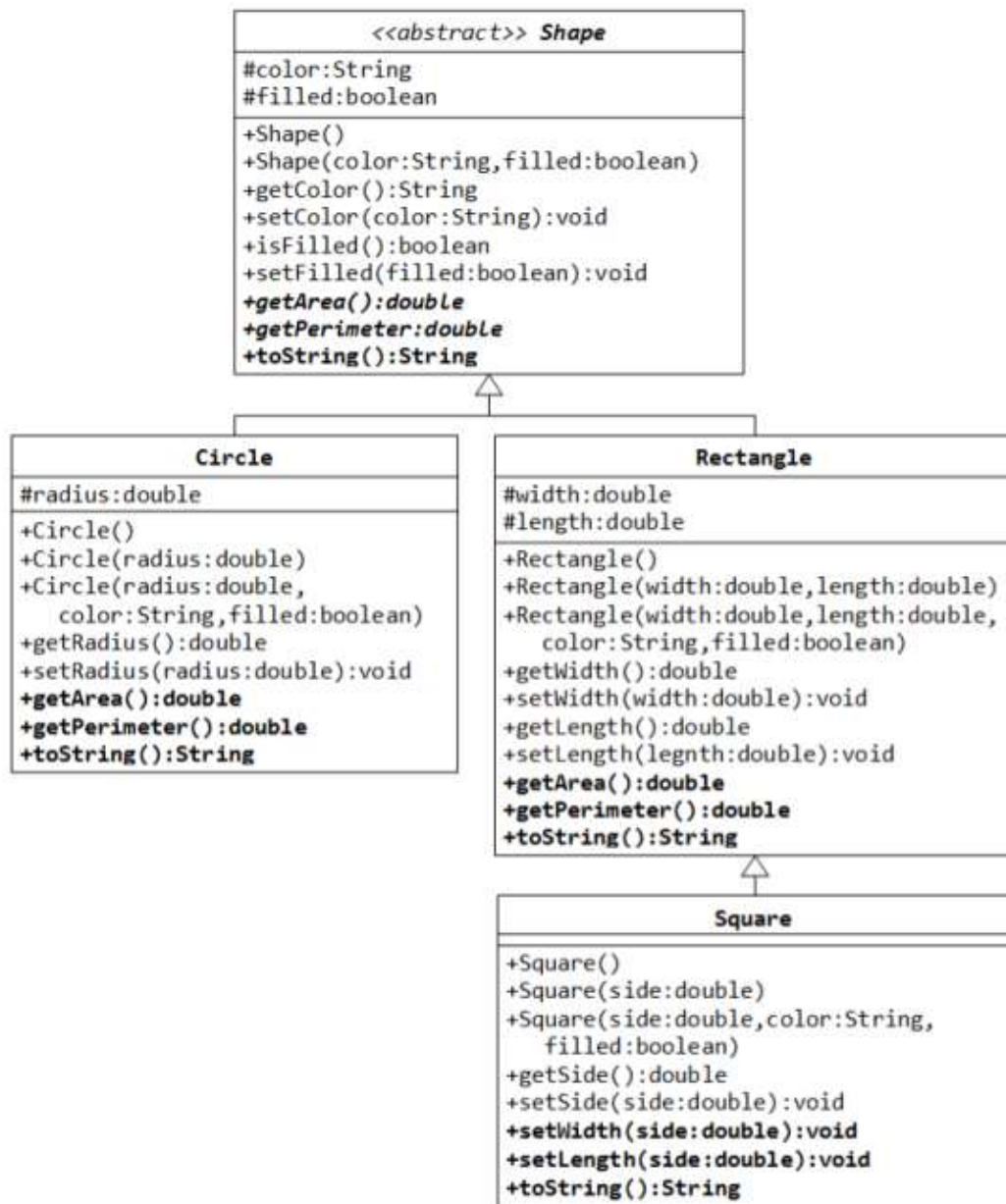


Рисунок 5.1 – Диаграмма суперкласса Shape

- 1) проверить возможность выполнить методы класса при приведении типов (полиморфизм класса);
- 2) напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable;
- 3) напишите новый класс MovableRectangle.

Решение

1. Разработка программы

На рисунке 5.2 представлен код класса Shape

```
package Test;

5 usages 3 inheritors
abstract public class Shape {
    8 usages
    protected String color;
    7 usages
    protected boolean filled;
    3 usages
    public String getColor() {return color;}
    no usages
    public void setColor(String color) {this.color = color;}
    no usages
    public void setFilled(boolean filled) {this.filled = filled;}
    no usages
    public Shape(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
    }
    5 usages
    public Shape(){}
    3 usages
    public boolean isFilled() { return filled; }
    3 usages 2 implementations
    abstract public double getArea();
    2 usages 2 implementations
    abstract public double getPerimeter();

    3 overrides
    @Override
    public String toString() {
        return "Shape{" +
            "color=" + color + '\'' +
            ", filled=" + filled +
            '}';
    }
}
```

Рисунок 5.2 - Код класса Shape

2. Разработка программы

На рисунке 5.3 представлен код класса Circle

```
package Test;
```

⚠ 2 ^

2 usages

```
public class Circle extends Shape{
```

8 usages

```
    private double radius;
```

no usages

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

1 usage

```
3 public Circle(double radius, String color, boolean filled){
```

```
    this.color = color;
```

```
    this.filled = filled;
```

```
    this.radius = radius;
```

```
1 }
```

1 usage

```
1 public double getRadius() { return radius; }
```

no usages

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

3 usages

```
1 public double getArea() { return (Math.PI*radius)*(Math.PI*radius); }
```

2 usages

```
1 public double getPerimeter() { return 2*Math.PI*radius; }
```

@Override

```
1 public String toString() { return String.format("Shape:Circle,radius:%s",this.radius); }
```

```
}
```

Рисунок 5.3 - Код класса Circle

На рисунке 5.4 изображен код класса Rectangle

```
package Test;
```

4

3 usages 1 inheritor

```
public class Rectangle extends Shape {
```

13 usages

```
    protected double width;
```

11 usages

```
    protected double length;
```

3 usages

```
    public Rectangle(){} 
```

no usages

```
    public double getWidth() {return width;} 
```

no usages 1 override

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

1 usage

```
    public double getLength() {return length;} 
```

no usages 1 override

```
    public void setLength(double length) {this.length = length;} 
```

no usages

```
1    public Rectangle(double width, double length) {
```

```
        this.width = width;
```

```
        this.length = length;
```

```
1    }
```

1 usage

```
1    public Rectangle(double width, double length, String color, boolean filled){
```

```
        this.color = color;
```

```
        this.filled = filled;
```

```
        this.width = width;
```

```
        this.length = length;
```

```
1    }
```

3 usages

```
1    public double getArea() { return (length*width); }
```

2 usages

```
1    public double getPerimeter() { return ((length*2)+(width*2)); }
```

1 override

```
@Override
```

```
1    public String toString() { return String.format("Shape:Rectangle, width:%s, length:%s",this.width,this.length); }
```

```
1 }
```

Рисунок 5.4 - Код класса Rectangle

На рисунке 5.5 изображен код класса Square

```

package Test;

2 usages
public class Square extends Rectangle{

    no usages
    public Square(){ }

    no usages
    public Square(double side){
        this.length = side;
        this.width = side;
    }

    1 usage
    public Square(double side, String color, boolean filled){
        this.length = side;
        this.width = side;
        this.color = color;
        this.filled = filled;
    }

    1 usage
    public double getSide() { return width; }

    no usages
    public void setSide(double side) {
        this.width = side;
        this.length = side;
    }

    no usages
    public void setWidth(double side) { this.width = side; }

    no usages
    public void setLength(double side) { this.length = side; }

    @Override
    public String toString() { return String.format("Shape: square, side:%s, color:%s",this.width,this.color); }
}

```

Рисунок 5.5 - Код класса Square

3. Разработка программы

Часть кода задания представлена на рисунках 5.6 и 5.7


```

package Test;

public class Test {
    public static void main(String[] args) {
        Shape s1 = new Circle( radius: 5.5, color: "RED", filled: 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());

        System.out.println("\n");

        Shape s3 = new Rectangle( width: 1.0, length: 2.0, color: "BLACK", filled: false);
        System.out.println(s3);
        System.out.println(s3.getArea());
        System.out.println(s3.getPerimeter());
        System.out.println(s3.getColor());
        System.out.println(s3.isFilled());
        System.out.println(((Rectangle)s3).getLength());

        System.out.println("\n");

        Shape s4 = new Square( side: 6, color: "BLUE", filled: true);
        System.out.println(s4);
        System.out.println(s4.getArea());
        System.out.println(s4.getColor());
        System.out.println(s4.isFilled());
        System.out.println(((Square)s4).getSide());
    }
}

```

Рисунок 5.6 – Часть кода задания

```

package Test;
public class Test_5 {
    public static void main(String[] args) {
        MovableRectangle movableRectangle = new MovableRectangle( x1: 3, x2: 6, y1: 2, y2: -7, xSpeed: 10, ySpeed: 10);
        System.out.println(movableRectangle.toString());
        movableRectangle.moveUp();
        movableRectangle.moveRight();
        System.out.println(movableRectangle.toString());
    }
}

```

Рисунок 5.7 – Вторая часть кода задания

4. Разработка программы

На рисунке 3.8 представлен код интерфейса Movable, а на рисунках 3.9 и 3.10 код классов MovablePoint, MovableCircle которые его используют.

```
package Test;

3 usages 3 implementations
public interface Movable {
    5 usages 3 implementations
    public void moveUp();
    3 usages 3 implementations
    public void moveDown();
    4 usages 3 implementations
    public void moveLeft();
    4 usages 3 implementations
    public void moveRight();
}
```

Рисунок 5.8 – Код интерфейса Movable

```

package Test;

6 usages
public class MovablePoint implements Movable{

    6 usages
    private int x;

    6 usages
    private int y;

    4 usages
    private int xSpeed;

    4 usages
    private int ySpeed;

    3 usages
    public MovablePoint(int x, int y, int xSpeed, int ySp
        this.x = x;
        this.y=y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
    }

    @Override
    3 usages
    public String toString(){
        return ("[point] coords:[" + x + ", "+y+
            "], speed:["+xSpeed +", "+ ySpeed+"]");
    }

    5 usages
    @Override
    3 usages
    public void moveUp() { this.y = y + ySpeed; }

    @Override
    3 usages
    public void moveDown() { this.y = y - ySpeed; }

    4 usages
    @Override
    3 usages
    public void moveLeft() { this.x = x - xSpeed; }

    4 usages
    @Override
    3 usages
    public void moveRight() { this.x = x + xSpeed; }
}

```

Рисунок 5.9 – Код класса MovablePoint

```

package Test;

2 usages
public class MovableCircle implements Movable{
    2 usages
    private int radius;
    6 usages
    private MovablePoint center;

    1 usage
    public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius)
    {
        this.radius = radius;
        this.center = new MovablePoint(x,y,xSpeed,ySpeed);
    }
    @Override
    public String toString(){
        return ("Figure: [circle]; Radius["+radius+
            "]; Center: [" + center.toString() +"]");
    }
    5 usages
    @Override
    public void moveUp() { this.center.moveUp(); }
    3 usages
    @Override
    public void moveDown() { this.center.moveDown(); }
    4 usages
    @Override
    public void moveLeft() { this.center.moveLeft(); }
    4 usages
    @Override
    public void moveRight(){this.center.moveRight();}
}

```

Рисунок 5.10 – Код класса MovableCircle

5. Разработка программы

На рисунке 5.11 представлен код класса MovableRectangle

```

package Test;

2 usages
public class MovableRectangle implements Movable{
    6 usages
    private MovablePoint topLeft; private MovablePoint bottomRight;
    1 usage
    public MovableRectangle(int x1, int x2, int y1, int y2, int xSpeed, int ySpeed){
        this.topLeft = new MovablePoint(x1,y1,xSpeed,ySpeed);
        this.bottomRight = new MovablePoint(x2,y2,xSpeed,ySpeed);
    }
    @Override
    public String toString(){
        return ("Figure: [rectangle]; topLeft: ["+topLeft.toString()+
            "]; bottomRight: [" + bottomRight.toString() +"]");
    }
    5 usages
    @Override
    public void moveUp(){
        this.bottomRight.moveUp(); this.topLeft.moveUp();
    }
    3 usages
    @Override
    public void moveDown(){
        this.topLeft.moveDown(); this.bottomRight.moveUp();
    }
    4 usages
    @Override
    public void moveLeft(){
        this.topLeft.moveLeft(); this.bottomRight.moveLeft();
    }
    4 usages
    @Override
    public void moveRight(){
        this.topLeft.moveRight(); this.bottomRight.moveRight();
    }
}

```

Рисунок 5.11 – Код класса MovableRectangle

Выводы по работе

В ходе работы были изучены основы работы с абстрактными классами и их методами.

Практическая работа № 6

Цель работы

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

Теоретическое введение

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

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

Разработка программы

На рисунке 6.1 представлен код задания 1.

```

public class Circle extends Shape{
    8 usages
    private double radius;
    no usages
    public Circle(double radius) { this.radius = radius; }
    1 usage
    public Circle(double radius, String color, boolean filled){
        this.color = color;
        this.filled = filled;
        this.radius = radius;
    }
    1 usage
    public double getRadius() { return radius; }
    no usages
    public void setRadius(double radius) { this.radius = radius; }
    3 usages
    public double getArea() { return (Math.PI*radius)*(Math.PI*radius); }
    2 usages
    public double getPerimeter() { return 2*Math.PI*radius; }
    @Override
    public String toString() { return String.format("Shape:Circle,radius:%s",this.radius); }
}

```

Рисунок 6.1 – Код задания 1

На рисунках 6.2 и 6.3 представлен код задания 2.

```

abstract class Dish {
    2 usages
    private String material;
    2 usages
    private double weight;

    2 usages
    public Dish(String material, double weight) {
        this.material = material;
        this.weight = weight;
    }

    2 usages 2 implementations
    public abstract void use();

    no usages
    public String getMaterial() {
        return material;
    }

    no usages
    public double getWeight() {
        return weight;
    }
}

```

Рисунок 6.2 – Код задания 2

```
class Plate extends Dish {  
    2 usages  
    private int capacity;  
  
    1 usage  
    public Plate(String material, double weight, int capacity) {  
        super(material, weight);  
        this.capacity = capacity;  
    }  
  
    2 usages  
    @Override  
    public void use() {  
        System.out.println("Using the plate to serve food.");  
    }  
  
    no usages  
    public int getCapacity() {  
        return capacity;  
    }  
}  
  
2 usages  
class Cup extends Dish {  
    2 usages  
    private boolean handle;  
  
    1 usage  
    public Cup(String material, double weight, boolean handle) {  
        super(material, weight);  
        this.handle = handle;  
    }  
  
    2 usages  
    @Override  
    public void use() {  
        System.out.println("Using the cup to drink.");  
    }  
  
    no usages  
    public boolean hasHandle() {  
        return handle;  
    }  
}
```

Рисунок 6.3 – Код задания 2

На рисунках 6.4 и 6.5 представлен код задания 3.

```
abstract class Furniture {  
    2 usages  
    private String name;  
    2 usages  
    private double price;  
  
    2 usages  
    public Furniture(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    2 usages  
    public String getName() { return name; }  
  
    2 usages  
    public double getPrice() { return price; }  
  
    1 usage 2 implementations  
    public abstract void displayInfo();  
}  
  
2 usages  
class Chair extends Furniture {  
    1 usage  
    public Chair(String name, double price) { super(name, price); }  
  
    1 usage  
    @Override  
    public void displayInfo() { System.out.println("Chair: " + getName() + ", Price: $" + getPrice()); }  
}
```

Рисунок 6.4 – Код задания 3

```

class Table extends Furniture {
    1 usage
    public Table(String name, double price) { super(name, price); }

    1 usage
    @Override
    public void displayInfo() { System.out.println("Table: " + getName() + ", Price: $" + getPrice()); }
}

2 usages
class FurnitureShop {
    3 usages
    private List<Furniture> inventory;

    1 usage
    public FurnitureShop() { inventory = new ArrayList<>(); }

    2 usages
    public void addFurniture(Furniture furniture) { inventory.add(furniture); }

    1 usage
    public void displayInventory() {
        System.out.println("Furniture Inventory:");
        for (Furniture item : inventory) {
            item.displayInfo();
        }
    }
}

```

Рисунок 6.5 – Код задания 3

На рисунке 6.6 представлен результат выполнения программы.

```

Using the plate to serve food.
Using the cup to drink.
Labrador is barking.
Poodle is barking.
Furniture Inventory:
Chair: Wooden Chair, Price: $50.0
Table: Dining Table, Price: $150.0

```

Рисунок 6.6 – Результат выполнения программы

Выводы по работе:

В ходе работы были изучено понятие наследование, а также были получены навыки его реализации на Java.

Практическая работа № 7

Цель работы

Введение в событийное программирование на языке Java.

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.

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.

Выполнение лабораторной работы

Задание

Написать интерактивную программу с использованием GUI, которая имитирует таблицу результатов матчей между командами Милан и Мадрид.

Решение

1. Разработка программы

На рисунках 4.1 и 4.2 представлены фрагменты кода программы.

```

package Test;

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

class CHETIRE extends JFrame {
    8 usages
    static int milanScore = 0; static int madridScore = 0; static String lastScorerTeam = "N/A";
    9 usages
    static String winner = "DRAW"; JLabel ResultJLabel = new JLabel( text: "Result: " + madridScore + " X " + milanScore );
    7 usages
    JLabel LastScorerJLabel = new JLabel( text: "Last Scorer: " + lastScorerTeam); Label WinnerLabel = new Label( text: "Winner: " + winner);
    2 usages
    JButton MilanButton = new JButton( text: "AC Milan"); JButton RealButton = new JButton( text: "Real Madrid");
    1 usage
    public CHETIRE()
    {
        super( title: "SCORING");
        setSize( width: 400, height: 400);
        setLayout(new BorderLayout());
        MilanButton.addActionListener(a -> {
            milanScore++;
            lastScorerTeam = "AC Milan";
            ResultJLabel.setText("Result: " + madridScore + " X " + milanScore);
            LastScorerJLabel.setText("Last Scorer: " + lastScorerTeam);
            if(milanScore > madridScore){
                winner = "AC Milan";
                WinnerLabel.setText("Winner: " + winner);
                getContentPane().setBackground(Color.GREEN);
                ResultJLabel.setForeground(Color.BLACK); // весь цвет не нужен!
                LastScorerJLabel.setForeground(Color.BLACK);
            } else if(milanScore == madridScore) {
                winner = "DRAW";
                WinnerLabel.setText("Winner: " + winner);
                getContentPane().setBackground(Color.GRAY);
                ResultJLabel.setForeground(Color.WHITE);
                LastScorerJLabel.setForeground(Color.WHITE);
            }
        });
    }
}

```

Рисунок 7.1 – Часть кода программы

```

RealButton.addActionListener(b -> {
    madridScore++;
    lastScorerTeam = "Real Madrid";
    ResultJLabel.setText("Result: " + madridScore + " X " + milanScore );
    LastScorerJLabel.setText("Last Scorer: " + lastScorerTeam);
    if(madridScore > milanScore){
        winner = "Real Madrid";
        WinnerLabel.setText("Winner: " + winner);
        getContentPane().setBackground(Color.MAGENTA);
        ResultJLabel.setForeground(Color.WHITE);
        LastScorerJLabel.setForeground(Color.WHITE);
    } else if(milanScore == madridScore) {
        winner = "DRAW";
        WinnerLabel.setText("Winner: " + winner);
        getContentPane().setBackground(Color.GRAY);
        ResultJLabel.setForeground(Color.WHITE);
        LastScorerJLabel.setForeground(Color.WHITE);
    }
});

ResultJLabel.setHorizontalAlignment(0);
add(ResultJLabel, BorderLayout.NORTH);
add(LastScorerJLabel, BorderLayout.CENTER);
add(WinnerLabel, BorderLayout.SOUTH);
add(MilanButton, BorderLayout.EAST);
add(RealButton, BorderLayout.WEST);
}

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

```

Рисунок 7.2 – Вторая часть кода программы

На рисунке 7.3 изображено окно программы во время исполнения.

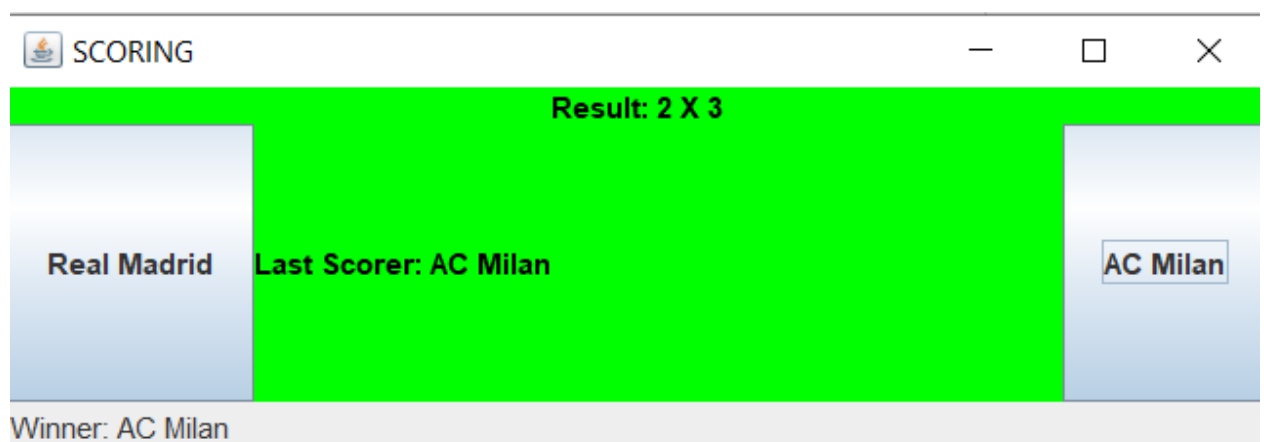


Рисунок 7.3 – Окно программы

Выводы по работе

В ходе работы были изучены основы работы с интерфейсами GUI и слушателями.

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

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.

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.

Выполнение лабораторной работы

Задание

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. Затем создать окно, отобразить в нем картинку, путь к которой указан в аргументах командной строки. Далее

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

Решение

Разработка программы

На рисунке 8.1 представлен фрагмент кода программы.

```
package Test2;
import java.awt.*;
1 usage
class MyRectangle extends Shape {
    2 usages
    private int width;
    2 usages
    private int height;

    1 usage
    public MyRectangle(Color color, int x, int y, int width, int height) {
        super(color, x, y);
        this.width = width;
        this.height = height;
    }

    1 usage
    @Override
    void draw(Graphics g) {
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
}
```

Рисунок 8.1 – Часть кода программы

На рисунках 8.2, 8.3 и 8.4 представлены фрагменты кода программы.

```

package Test2;

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

5 usages 2 inheritors
abstract class Shape {
    3 usages
    protected Color color;
    3 usages
    protected int x;
    3 usages
    protected int y;

    2 usages
    public Shape(Color col, int x, int y) {
        this.color = col;
        this.x = x;
        this.y = y;
    }

    1 usage 2 implementations
    abstract void draw(Graphics g);
}

1 usage
class Circle extends Shape {
    5 usages
    private int radius;

    1 usage
    public Circle(Color color, int x, int y, int radius) {
        super(color, x, y);
        this.radius = radius;
    }

    1 usage
    @Override
    void draw(Graphics g) {
        g.setColor(color);
        g.fillOval(x - radius, y - radius, width: 2 * radius, height: 2 * radius);
    }
}

```

Рисунок 8.2 – Следующая часть кода программы


```

public class RandomShapes extends JPanel {
    4 usages
    private final Shape[] shapes;

    1 usage
    public RandomShapes(int width, int height, int numShapes) {
        shapes = new Shape[numShapes];
        Random random = new Random();

        for (int i = 0; i < numShapes; i++) {
            int shapeType = random.nextInt( bound: 2);
            Color col = new Color(random.nextInt( bound: 256), random.nextInt( bound: 256), random.nextInt( bound: 256));
            int x = random.nextInt(width);
            int y = random.nextInt(height);

            if (shapeType == 0) {
                int radius = random.nextInt( bound: 50) + 10;
                shapes[i] = new Circle(col, x, y, radius);
            } else {
                int widthRect = random.nextInt( bound: 100) + 20;
                int heightRect = random.nextInt( bound: 100) + 20;
                shapes[i] = new MyRectangle(col, x, y, widthRect, heightRect);
            }
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (Shape shape : shapes) {
            shape.draw(g);
        }
    }
}

```

Рисунок 8.3 – Следующая часть кода программы

```

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    for (Shape shape : shapes) {
        shape.draw(g);
    }
}

public static void main(String[] args) {
    int width = 800;
    int height = 600;
    int numShapes = 20;

    JFrame frame = new JFrame( title: "Random Shapes");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    RandomShapes randomShapes = new RandomShapes(width, height, numShapes);
    frame.add(randomShapes);
    frame.setSize(width, height);
    frame.setVisible(true);
}
}

```

Рисунок 8.4 – Следующая часть кода программы

На рисунке 8.5 изображено окно программы во время выполнения.

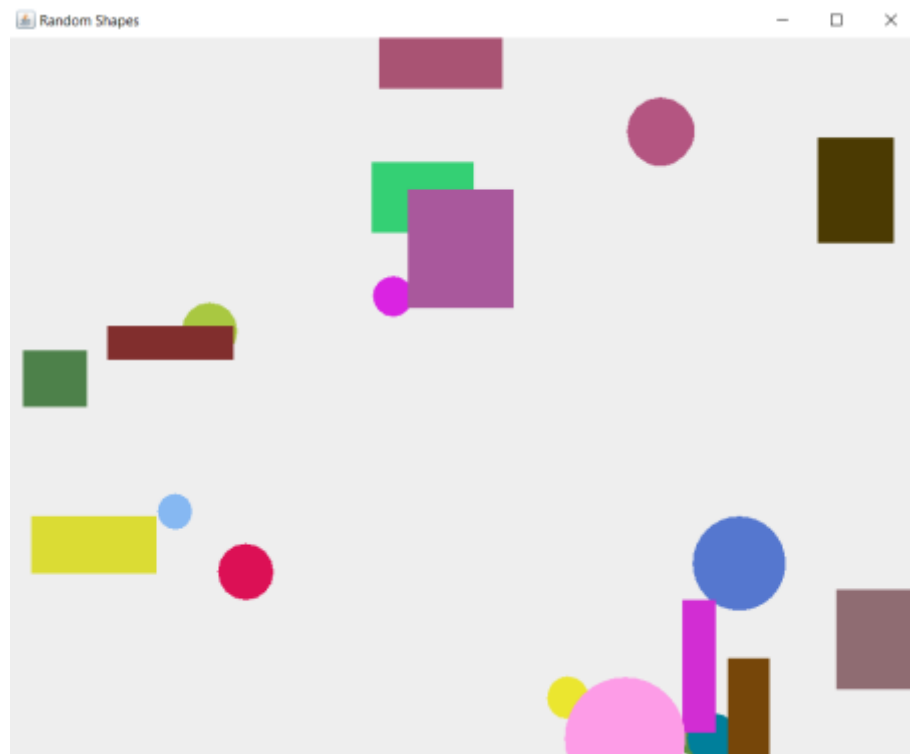


Рисунок 8.5 – Окно программы во время выполнения

На рисунках 8.6 и 8.7 представлен код следующего задания.

```
package Test2;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

public class AnimationFrame extends JPanel implements ActionListener {
    4 usages
    private List<ImageIcon> frames;
    6 usages
    private int currentFrame;

    1 usage
    public AnimationFrame(List<ImageIcon> frames) {
        this.frames = frames;
        currentFrame = 0;

        Timer timer = new Timer( delay: 100, listener: this); // Обновление каждые 100 миллисекунд
        timer.start();
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (currentFrame < frames.size()) {
            ImageIcon frame = frames.get(currentFrame);
            frame.paintIcon( c: this, g, x: 0, y: 0);
        }
    }
}
```

Рисунок 8.6 – Часть кода программы

```

@Override
public void actionPerformed(ActionEvent e) {
    currentFrame++;
    if (currentFrame >= frames.size()) {
        currentFrame = 0;
    }
    repaint();
}

public static void main(String[] args) {
    List<ImageIcon> frames = new ArrayList<>();
    frames.add(new ImageIcon( filename: "C:\\Users\\diono\\IdeaProjects\\Praktika1\\1.png"));
    frames.add(new ImageIcon( filename: "C:\\Users\\diono\\IdeaProjects\\Praktika1\\2.png"));

    JFrame frame = new JFrame( title: "Animation");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    AnimationFrame animationFrame = new AnimationFrame(frames);
    frame.add(animationFrame);
    frame.setSize(frames.get(0).getIconWidth(), frames.get(0).getIconHeight());
    frame.setVisible(true);
}
}

```

Рисунок 8.7 – Следующая часть кода программы

На рисунках 8.8 и 8.9 изображено окно программы во время выполнения.

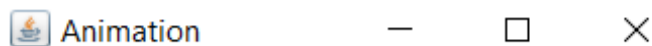


Рисунок 8.8 – Окно программы



Рисунок 8.9 – Окно программы

Выводы по работе

В ходе работы были изучены основы работы с интерфейсами GUI и слушателями.

Практическая работа № 9

Цель работы

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

Теоретическое введение

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

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

Выполнение лабораторной работы

Задание

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

Решение

Разработка программы

На рисунке 9.1 представлен код задания 1.

```
package Test2;

6 usages 3 implementations
public interface Nameable {
    3 usages 3 implementations
    String getName();
}

1 usage
class Planet implements Nameable {
    2 usages
    private String name;

    1 usage
    public Planet(String name) {
        this.name = name;
    }

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

1 usage
class Animal implements Nameable {
    2 usages
    private String name;

    1 usage
    public Animal(String name) {
        this.name = name;
    }

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

Рисунок 8.1 – Код задания 1

На рисунке 9.2 и 9.3 представлено продолжение кода задания 1.

```
1 usage
class Car implements Nameable {
    2 usages
    private String name;

    1 usage
    public Car(String name) {
        this.name = name;
    }

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

Рисунок 9.2 – Продолжение кода задания 1

```
package Test2;
```

```
public class Main {
    public static void main(String[] args) {
        Nameable planet = new Planet( name: "Earth");
        System.out.println("Planet name: " + planet.getName());

        Nameable animal = new Animal( name: "Lion");
        System.out.println("Animal name: " + animal.getName());

        Nameable car = new Car( name: "Toyota Camry");
        System.out.println("Car name: " + car.getName());
    }
}
```

Рисунок 9.3 – Продолжение кода задания 1

На рисунках 9.4, 9.5 и 9.6 представлена часть кода задания 2 и его продолжение.

```
package Test2;

6 usages 3 implementations
public interface Priceable {
    3 usages 3 implementations
    double getPrice();
}

1 usage
class Laptop implements Priceable {
    2 usages
    private double price;

    1 usage
    public Laptop(double price) {
        this.price = price;
    }

    3 usages
    @Override
    public double getPrice() {
        return price;
    }
}

1 usage
class Book implements Priceable {
    2 usages
    private double price;

    1 usage
    public Book(double price) {
        this.price = price;
    }

    3 usages
    @Override
    public double getPrice() {
        return price;
    }
}
```

Рисунок 9.4 – Часть кода задания 2

```

class Fruit implements Priceable {
    2 usages
    private double pricePerKg;
    2 usages
    private double weight;

    1 usage
    public Fruit(double pricePerKg, double weight) {
        this.pricePerKg = pricePerKg;
        this.weight = weight;
    }

    3 usages
    @Override
    public double getPrice() {
        return pricePerKg * weight;
    }
}

```

Рисунок 9.5 – Продолжение кода задания 2

```

Priceable laptop = new Laptop( price: 1000.0);
System.out.println("Laptop price: $" + laptop.getPrice());

Priceable book = new Book( price: 20.0);
System.out.println("Book price: $" + book.getPrice());

Priceable fruit = new Fruit( pricePerKg: 2.5, weight: 3.0);
System.out.println("Fruit price: $" + fruit.getPrice());
}
}

```

Рисунок 9.6 – Продолжение кода задания 2

На рисунке 9.7 представлен результат выполнения всех заданий.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Planet name: Earth  
Animal name: Lion  
Car name: Toyota Camry  
Laptop price: $1000.0  
Book price: $20.0  
Fruit price: $7.5  
  
Process finished with exit code 0
```

Рисунок 9.7 – Результат выполнения

Выводы по работе:

В ходе работы было изучено понятие интерфейса, получены навыки создания интерфейса в Java и применения их в программах.

Практическая работа № 10

Цель работы

Разработка и программирование рекурсивных алгоритмов на языке Java.

Теоретическое введение

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

Выполнение лабораторной работы

Задание

1) даны числа a и b . Определите, сколько существует последовательностей из a нулей и b единиц, в которых никакие два нуля не стоят рядом;

2) дано число n , десятичная запись которого не содержит нулей. Получите число, записанное теми же цифрами, но в противоположном порядке;

3) 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.

Решение

1. Разработка программы задания 1

На рисунке 10.1 представлен код требуемой функции.

```
package Test;

public class SumOfDigits {
    public static void main(String[] args) {
        int N = 62;
        int sum = calculateSumOfDigits(N);
        System.out.println("Сумма цифр числа " + N + " равна " + sum);
    }

    2 usages
    public static int calculateSumOfDigits(int N) {
        if (N < 10) {
            return N;
        } else {
            int lastDigit = N % 10;
            int remainingDigits = N / 10;
            return lastDigit + calculateSumOfDigits(remainingDigits);
        }
    }
}
```

Рисунок 10.1 – Код функции задания 1(5)

2. Разработка программы задания 2

На рисунке 10.2 представлен код требуемой функции.

```

package Test;

public class PrimeCheck {
    public static void main(String[] args) {
        int n = 17;
        boolean isPrime = isPrimeNumber(n);
        if (isPrime) {
            System.out.println("YES");
        } else {
            System.out.println("NO");
        }
    }
}

2 usages
public static boolean isPrimeNumber(int n, int divisor) {
    if (n <= 1) {
        return false;
    }
    if (divisor == 1) {
        return true;
    }
    if (n % divisor == 0) {
        return false;
    }
    return isPrimeNumber(n, divisor - 1);
}

1 usage
public static boolean isPrimeNumber(int n) {
    return isPrimeNumber(n, n - 1);
}
}

```

Рисунок 10.2 – Код функции задания 2(б)

3. Разработка программы задания 3

На рисунке 10.3 представлен код требуемой функции.

```

package Test;

public class PrimeFactors {
    public static void main(String[] args) {
        int n = 60;
        System.out.println("Простые множители числа " + n + ":");
        primeFactors(n, divisor: 2);
    }

    3 usages
    public static void primeFactors(int n, int divisor) {
        if (n <= 1) {
            return;
        }

        if (n % divisor == 0) {
            System.out.print(divisor + " ");
            primeFactors(n: n / divisor, divisor);
        } else {
            primeFactors(n, divisor: divisor + 1);
        }
    }
}

```

Рисунок 10.3 – Код функции задания 3(7)

На рисунках 10.4, 10.5 и 10.6 представлены результаты компиляции программ.

Сумма цифр числа 62 равна 8

Process finished with exit code 0

Рисунок 10.4 – Результат выполнения кода задания 1(5)

YES

Process finished with exit code 0

Рисунок 10.5 – Результат выполнения кода задания 2(6)

```
Простые множители числа 60:  
2 2 3 5  
Process finished with exit code 0
```

Рисунок 10.6 – Результат выполнения кода задания 3(7)

Выводы по работе:

В ходе работы были изучены основы работы с рекурсивными алгоритмами.

Практическая работа № 11

Цель работы

Цель данной лабораторной 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 или сортировка списка людей в алфавитном порядке по фамилии. Есть много алгоритмов для сортировки списка элементов, которые различаются по эффективности, такие как: вставками, быстрой сортировки, слиянием и так далее.

Выполнение лабораторной работы

Задание

Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber и сортирует его вставками. Напишите класс SortingStudentsByGPA который реализует интерфейс Comparator таким образом, чтобы сортировать список студентов по их итоговым баллам в порядке убывания с использованием алгоритма быстрой сортировки. Напишите программу, которая объединяет два списка данных о студентах в один отсортированный списках с использованием алгоритма сортировки слиянием.

Решение

Разработка программы

На рисунке 11.1 представлен код задания 1.

```
package Test;

7 usages
class Student3 {
    2 usages
    int iDNumber;
    2 usages
    String name;

    4 usages
    public Student3(int iDNumber, String name) {
        this.iDNumber = iDNumber;
        this.name = name;
    }

    3 usages
    public int getIDNumber() {
        return iDNumber;
    }

    1 usage
    public String getName() { return name; }
}

public class StudentSortTest {
    public static void main(String[] args) {
        Student3[] students = {
            new Student3( iDNumber: 102, name: "Platon"),
            new Student3( iDNumber: 101, name: "Alexey"),
            new Student3( iDNumber: 103, name: "Dmitriy"),
            new Student3( iDNumber: 100, name: "Roman")
        };

        for (int i = 1; i < students.length; i++) {
            Student3 currentStudent = students[i];
            int j = i - 1;
            while (j >= 0 && students[j].getIDNumber() > currentStudent.getIDNumber()) {
                students[j + 1] = students[j];
                j--;
            }
            students[j + 1] = currentStudent;
        }

        for (Student3 student : students) {
            System.out.println("ID: " + student.getIDNumber() + ", Name: " + student.getName());
        }
    }
}
```

Рисунок 11.1 – Код задания 1

На рисунках 11.2 и 11.3 представлен код задания 2 и его продолжение.

```
package Test;

import java.util.Arrays;
import java.util.Comparator;

9 usages
class Student2 {
    2 usages
    String name;
    2 usages
    int totalScore;

    4 usages
    public Student2(String name, int totalScore) {
        this.name = name;
        this.totalScore = totalScore;
    }

    1 usage
    public String getName() {
        return name;
    }

    3 usages
    public int getTotalScore() { return totalScore; }
}

2 usages
class SortingStudentsByGPA implements Comparator<Student2> {
    @Override
    public int compare(Student2 student1, Student2 student2) {
        return Integer.compare(student2.getTotalScore(), student1.getTotalScore());
    }
}
```

Рисунок 11.2 – Код задания 2

```

public class StudentGPASortTest {
    public static void main(String[] args) {
        Student2[] students = {
            new Student2( name: "Platon", totalScore: 90),
            new Student2( name: "Alexey", totalScore: 85),
            new Student2( name: "Dmitriy", totalScore: 92),
            new Student2( name: "Roman", totalScore: 88)
        };

        SortingStudentsByGPA sorter = new SortingStudentsByGPA();
        Arrays.sort(students, sorter);

        for (Student2 student : students) {
            System.out.println("Name: " + student.getName() + ", GPA: " + student.getTotalScore());
        }
    }
}

```

Рисунок 11.3 – продолжение кода задания 2

На рисунках 11.4, 11.5 и 11.6 представлен код задания 3 и его продолжение.

```
package Test;

import java.util.Arrays;

15 usages
class Student {
    2 usages
    String name;
    2 usages
    int id;

    6 usages
    public Student(String name, int id) {
        this.name = name;
        this.id = id;
    }

    1 usage
    public String getName() {
        return name;
    }

    3 usages
    public int getId() {
        return id;
    }
}
```

Рисунок 11.4 – Код задания 3

```

public class MergeSortedStudents {
    public static void main(String[] args) {
        Student[] students1 = {
            new Student( name: "Platon", id: 101),
            new Student( name: "Alexey", id: 102),
            new Student( name: "Dmitriy", id: 105)
        };

        Student[] students2 = {
            new Student( name: "Roman", id: 103),
            new Student( name: "Danya", id: 104),
            new Student( name: "Evokak", id: 106)
        };

        Student[] mergedStudents = mergeSortedArrays(students1, students2);

        // Вывод объединенного массива студентов
        for (Student student : mergedStudents) {
            System.out.println("Name: " + student.getName() + ", ID: " + student.getId());
        }
    }
}

```

Рисунок 11.5 – Продолжение кода задания 3

```

    public static Student[] mergeSortedArrays(Student[] arr1, Student[] arr2) {
        int len1 = arr1.length;
        int len2 = arr2.length;
        Student[] merged = new Student[len1 + len2];

        int i = 0, j = 0, k = 0;

        while (i < len1 && j < len2) {
            if (arr1[i].getId() < arr2[j].getId()) {
                merged[k++] = arr1[i++];
            } else {
                merged[k++] = arr2[j++];
            }
        }

        while (i < len1) {
            merged[k++] = arr1[i++];
        }

        while (j < len2) {
            merged[k++] = arr2[j++];
        }

        return merged;
    }
}

```

Рисунок 11.6 – Продолжение кода задания 3

На рисунках 11.7, 11.8 и 11.9 представлен результат выполнения всех заданий.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
ID: 100, Name: Roman  
ID: 101, Name: Alexey  
ID: 102, Name: Platon  
ID: 103, Name: Dmitriy  
  
Process finished with exit code 0
```

Рисунок 11.7 – Результат выполнения кода задания 1

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Name: Dmitriy, GPA: 92  
Name: Platon, GPA: 90  
Name: Roman, GPA: 88  
Name: Alexey, GPA: 85  
  
Process finished with exit code 0
```

Рисунок 11.8 – Результат выполнения кода задания 2

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Name: Platon, ID: 101  
Name: Alexey, ID: 102  
Name: Roman, ID: 103  
Name: Danya, ID: 104  
Name: Dmitriy, ID: 105  
Name: Evokak, ID: 106  
  
Process finished with exit code 0
```

Рисунок 11.9 – Результат выполнения кода задания 3

Выводы по работе:

В ходе работы были изучены алгоритмы сортировки на языке Java, а также были получены навыки их применения

Практическая работа № 12

Цель работы

Цель данной лабораторной работы - изучение на практике приемов работы со стандартными контейнерными классами Java Collection Framework.

Теоретическое введение

Java Collections Framework — это набор связанных классов и интерфейсов, реализующих широко используемые структуры данных — коллекции. На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие в фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари). Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out). Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

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

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

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

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).

Используйте для организации хранения структуру данных Stack. Используйте для организации хранения структуру данных Queue. Используйте для организации хранения структуру данных Dequeue. Используйте для организации хранения

структуру данных DoubleList. Реализуйте более усложненный вариант решения задачи из упражнения. Реализация должна иметь интерактивный интерфейс для взаимодействия с пользователем.

Решение

Разработка программы

На рисунках 12.1 и 12.2 представлен код задания 1 и его продолжение.

```
package Test;

import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Scanner;

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String firstPlayerCards = scanner.nextLine();
        String secondPlayerCards = scanner.nextLine();

        Queue<Integer> firstPlayerDeck = new ArrayDeque<>();
        Queue<Integer> secondPlayerDeck = new ArrayDeque<>();

        // Заполнение колод
        for (int i = 0; i < 5; i++) {
            firstPlayerDeck.add(Integer.parseInt(String.valueOf(firstPlayerCards.charAt(i))));
            secondPlayerDeck.add(Integer.parseInt(String.valueOf(secondPlayerCards.charAt(i))));
        }

        int moves = 0;
```

Рисунок 12.1 – Код задания 1


```

while (moves <= 106) {
    int firstCard = firstPlayerDeck.poll();
    int secondCard = secondPlayerDeck.poll();

    if (firstCard > secondCard) {
        firstPlayerDeck.add(firstCard);
        firstPlayerDeck.add(secondCard);
    } else {
        secondPlayerDeck.add(secondCard);
        secondPlayerDeck.add(firstCard);
    }

    moves++;

    if (firstPlayerDeck.isEmpty()) {
        System.out.println("first " + moves);
        break;
    } else if (secondPlayerDeck.isEmpty()) {
        System.out.println("second " + moves);
        break;
    }
}

if (moves > 106) {
    System.out.println("botva");
}

scanner.close();
}
}

```

Рисунок 12.2 – Продолжение кода задания 1

На рисунке 12.3 представлен код с использованием Stack.

```

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

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String firstPlayerCards = scanner.nextLine();
        String secondPlayerCards = scanner.nextLine();

        Stack<Integer> firstPlayerDeck = new Stack<>();
        Stack<Integer> secondPlayerDeck = new Stack<>();

        // Заполнение колод
        for (int i = 0; i < 5; i++) {
            firstPlayerDeck.add(Integer.parseInt(String.valueOf(firstPlayerCards.charAt(i))));
            secondPlayerDeck.add(Integer.parseInt(String.valueOf(secondPlayerCards.charAt(i))));
        }

        int moves = 0;

        while (moves <= 106) {
            int firstCard = firstPlayerDeck.pop();
            int secondCard = secondPlayerDeck.pop();

            if (firstCard > secondCard) {
                firstPlayerDeck.add(firstCard);
                firstPlayerDeck.add(secondCard);
            } else {
                secondPlayerDeck.add(secondCard);
                secondPlayerDeck.add(firstCard);
            }
        }
    }
}

```

Рисунок 12.3 – Код с использованием Stack

На рисунке 12.4 представлен код с использованием Queue

```

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

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String firstPlayerCards = scanner.nextLine();
        String secondPlayerCards = scanner.nextLine();

        Queue<Integer> firstPlayerDeck = new LinkedList<>();
        Queue<Integer> secondPlayerDeck = new LinkedList<>();

        // Заполнение колод
        for (int i = 0; i < 5; i++) {
            firstPlayerDeck.add(Integer.parseInt(String.valueOf(firstPlayerCards.charAt(i))));
            secondPlayerDeck.add(Integer.parseInt(String.valueOf(secondPlayerCards.charAt(i))));
        }

        int moves = 0;

        while (moves <= 106) {
            int firstCard = firstPlayerDeck.poll();
            int secondCard = secondPlayerDeck.poll();

            if (firstCard > secondCard) {
                firstPlayerDeck.add(firstCard);
                firstPlayerDeck.add(secondCard);
            } else {
                secondPlayerDeck.add(secondCard);
                secondPlayerDeck.add(firstCard);
            }
        }
    }
}

```

Рисунок 12.4 – Код с использованием Queue

На рисунке 12.5 представлен код с использованием Dequeue

```

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Scanner;

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String firstPlayerCards = scanner.nextLine();
        String secondPlayerCards = scanner.nextLine();

        Deque<Integer> firstPlayerDeck = new ArrayDeque<>();
        Deque<Integer> secondPlayerDeck = new ArrayDeque<>();

        // Заполнение колод
        for (int i = 0; i < 5; i++) {
            firstPlayerDeck.add(Integer.parseInt(String.valueOf(firstPlayerCards.charAt(i))));
            secondPlayerDeck.add(Integer.parseInt(String.valueOf(secondPlayerCards.charAt(i))));
        }

        int moves = 0;

        while (moves <= 106) {
            int firstCard = firstPlayerDeck.poll();
            int secondCard = secondPlayerDeck.poll();

            if (firstCard > secondCard) {
                firstPlayerDeck.add(firstCard);
                firstPlayerDeck.add(secondCard);
            } else {
                secondPlayerDeck.add(secondCard);
                secondPlayerDeck.add(firstCard);
            }
        }
    }
}

```

Рисунок 12.5 – Код с использованием Dequeue

На рисунке 12.6 представлен код с использованием LinkedList

```

import java.util.LinkedList;
import java.util.Scanner;

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String firstPlayerCards = scanner.nextLine();
        String secondPlayerCards = scanner.nextLine();

        LinkedList<Integer> firstPlayerDeck = new LinkedList<>();
        LinkedList<Integer> secondPlayerDeck = new LinkedList<>();

        // Заполнение колод
        for (int i = 0; i < 5; i++) {
            firstPlayerDeck.add(Integer.parseInt(String.valueOf(firstPlayerCards.charAt(i))));
            secondPlayerDeck.add(Integer.parseInt(String.valueOf(secondPlayerCards.charAt(i))));
        }

        int moves = 0;

        while (moves <= 106) {
            int firstCard = firstPlayerDeck.poll();
            int secondCard = secondPlayerDeck.poll();

            if (firstCard > secondCard) {
                firstPlayerDeck.add(firstCard);
                firstPlayerDeck.add(secondCard);
            } else {
                secondPlayerDeck.add(secondCard);
                secondPlayerDeck.add(firstCard);
            }
        }
    }
}

```

Рисунок 12.6 – Код с использованием LinkedList

На рисунках 12.7 и 12.8 представлен код усложненного варианта решения задачи (с интерфейсом) и его продолжение.

```

package Test;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Scanner;

public class Pyanitsa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Deque<Integer> firstPlayerDeck = new ArrayDeque<>();
        Deque<Integer> secondPlayerDeck = new ArrayDeque<>();

        //Ввод карт для игрока 1
        System.out.println("Введите 5 карт для первого игрока (0-9):");
        for (int i = 0; i < 5; i++) {
            int card = scanner.nextInt();
            firstPlayerDeck.add(card);
        }

        //Ввод карт для игрока 2
        System.out.println("Введите 5 карт для второго игрока (0-9):");
        for (int i = 0; i < 5; i++) {
            int card = scanner.nextInt();
            secondPlayerDeck.add(card);
        }

        int moves = 0;

        while (moves <= 106) {
            int firstCard = firstPlayerDeck.poll();
            int secondCard = secondPlayerDeck.poll();

            System.out.println("Ход " + (moves + 1));
            System.out.println("Первый игрок: " + firstCard);
            System.out.println("Второй игрок: " + secondCard);

            if (firstCard > secondCard) {
                firstPlayerDeck.addLast(firstCard);
                firstPlayerDeck.addLast(secondCard);
                System.out.println("Первый игрок выиграл ход.");
            } else {
                secondPlayerDeck.addLast(secondCard);
                secondPlayerDeck.addLast(firstCard);
                System.out.println("Второй игрок выиграл ход.");
            }
        }
    }
}

```

Рисунок 12.7 – Код усложненного варианта решения задачи

```

        moves++;

        if (firstPlayerDeck.isEmpty()) {
            System.out.println("Первый игрок выиграл всю игру после " + moves + " ходов!");
            break;
        } else if (secondPlayerDeck.isEmpty()) {
            System.out.println("Второй игрок выиграл всю игру после " + moves + " ходов!");
            break;
        }

        System.out.println("Нажмите Enter для продолжения...");
        scanner.nextLine();

    }

    if (moves > 106) {
        System.out.println("Игра закончилась вничью (botva) после 106 ходов.");
    }

    scanner.close();
}
}

```

Рисунок 12.8 – Продолжение кода усложненного варианта решения задачи

На рисунке 12.9 представлен результат выполнения программ всех заданий

<pre> "C:\Program Files\Java\jdk-20\bin\java.exe" 63457 91820 first 11 Process finished with exit code 0 </pre>	<pre> "C:\Program Files\Java\jdk-20\bin\java.exe" Введите 5 карт для первого игрока (0-9): 5 4 3 1 2 Введите 5 карт для второго игрока (0-9): 6 1 4 2 3 Ход 1 Первый игрок: 5 Второй игрок: 6 Второй игрок выиграл ход. Нажмите Enter для продолжения... Ход 2 Первый игрок: 4 Второй игрок: 1 Первый игрок выиграл ход. Нажмите Enter для продолжения... </pre>
--	--

Рисунок 12.9 – Результат выполнения программ

Выводы по работе:

В ходе работы были изучены стандартные контейнерные классы на языке Java, а также были получены навыки их применения.

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 один из следующих конструкторов: `FileWriter(File file)`, `FileWriter(File file, boolean append)`, `FileWriter(FileDescriptor fd)`, `FileWriter(String fileName)`, `FileWriter(String fileName, boolean append)`.

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)`.

Выполнение лабораторной работы

Задание

Реализовать запись в файл введенной с клавиатуры информации; реализовать вывод информации из файла на экран; заменить информацию в файле на информацию, введенную с клавиатуры, добавить в конец исходного файла текст, введенный с клавиатуры.

Решение

Разработка программы

На рисунке 13.1 представлен код задания 1. Посредством работы классов `FileWriter` и `BufferedWriter` происходит запись информации в файл.

```

FileWriter fileWriter = new FileWriter(fileName);
BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

System.out.print("Введите информацию для записи в файл: ");
String content = scanner.nextLine();

bufferedWriter.write(content);
bufferedWriter.close();

```

Рисунок 13.1 – Код задания 1 – работа классов FileWriter и BufferedWriter

На рисунке 13.2 представлен код задания 2. С помощью классов FileReader и BufferedReader, а также цикла, происходит считывание информации из файла.

```

FileReader fileReader = new FileReader(fileName);
BufferedReader bufferedReader = new BufferedReader(fileReader);

String line;
System.out.println("Содержимое файла:");
while ((line = bufferedReader.readLine()) != null) {
    System.out.println(line);
}
bufferedReader.close();

```

Рисунок 13.2 – Код задания 2 – работа классов FileReader и BufferedReader

На рисунке 13.3 представлен код задания 3. Благодаря работе новой переменной newContent, а также классов FileWriter и BufferedWriter происходит замена существующей информации внутри файла.

```

System.out.print("Введите новую информацию для замены в файле: ");
String newContent = scanner.nextLine();

FileWriter fileWriterReplace = new FileWriter(fileName);
BufferedWriter bufferedWriterReplace = new BufferedWriter(fileWriterReplace);

bufferedWriterReplace.write(newContent);
bufferedWriterReplace.close();

```

Рисунок 13.3 – Код задания 3 – работа новой переменной newContent, а также классов FileWriter и BufferedWriter

На рисунке 13.4 представлен код задания 4. Посредством работы классов FileWriter и BufferedWriter, а также переменной appendContent происходит запись информации в файл.

```
FileWriter fileWriterAppend = new FileWriter(fileName, append: true);
BufferedWriter bufferedWriterAppend = new BufferedWriter(fileWriterAppend);

System.out.print("Введите текст для добавления в конец файла: ");
String appendContent = scanner.nextLine();

bufferedWriterAppend.write(appendContent);
bufferedWriterAppend.close();
```

Рисунок 13.4 – Код задания 4 – работа классов FileWriter и BufferedWriter, а также переменной appendContent

На рисунке 13.5 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-jav:
Введите информацию для записи в файл: 123
Содержимое файла:
123
Введите новую информацию для замены в файле: 456
Введите текст для добавления в конец файла: test
Готово.

Process finished with exit code 0
```

Рисунок 13.5 – Результат выполнения всей программы

На рисунке 13.6 представлено содержимое файла.

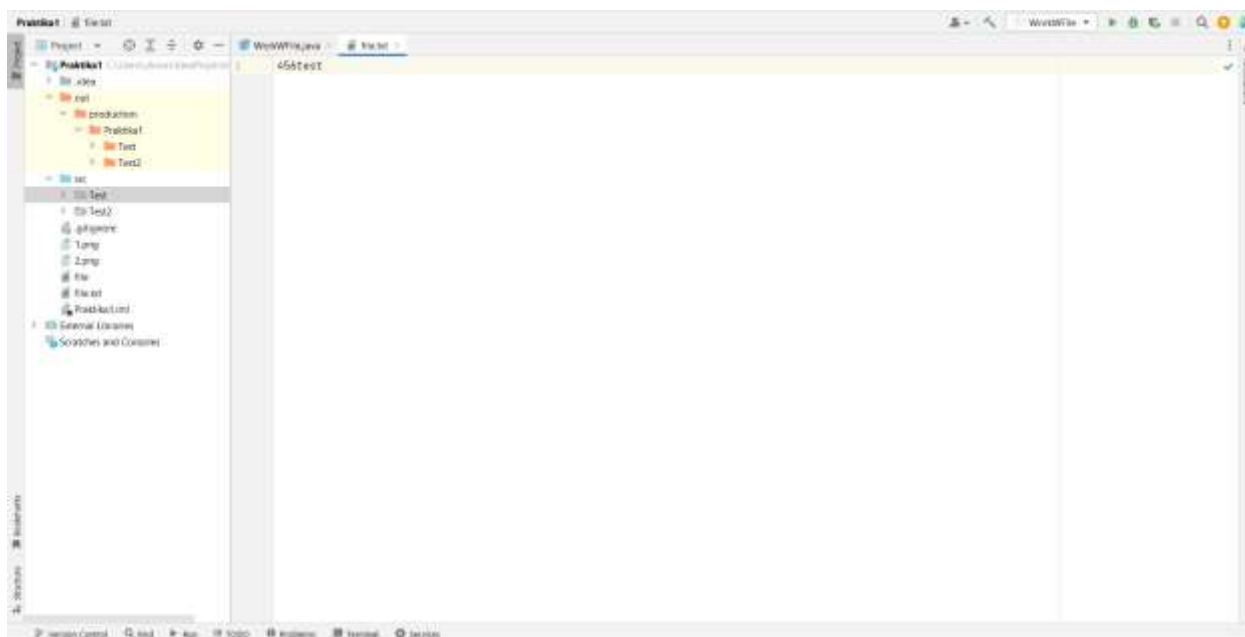


Рисунок 13.6 – Содержимое файла

Листинг полного кода программы представлен в приложении А на рисунках А.1 и А.2.

Выводы по работе:

В ходе работы была освоена работа с файлами, а также были получены навыки по чтению и записи данных в файл.

Практическая работа № 14

Цель работы

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

Теоретическое введение

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

- **BoundedWaitList**: Этот список ожидания имеет ограниченную емкость, указываемую в момент создания. Он не принимает более элементов, чем

заранее задано (возможное количество потенциальных элементов в списке ожидания).

- **UnfairWaitList:** В этом списке ожидания, можно удалить элемент, который не является первым в очереди - и помните он не может вернуться обратно! (Возможны различные реализации, но в вашей реализации необходимо удалить первое вхождение данного элемента.) Также возможно, чтобы, например, первый элемент будет отправлен обратно в конец списка.

После описания всей задачи в целом, мы сможем решить, что мы нам нужен интерфейс `IWaitList`, и затем нужно создать три разных класса для трех списков ожидания. Также предполагается, что один из списков ожидания должен быть супер-классом для двух других списков ожидания.

Выполнение лабораторной работы

Задание

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

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

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

Решение

На рисунке 14.1 представлен код класса-списка. Видны функции добавления и удаления элементов, функции проверки наличия и проверки на пустоту.

```

1 import java.util.Collection;
  1 usage 3 implementations new *
2 public interface IWaitList <E> {
  15 usages 2 implementations new *
3     void add(E element);
  2 usages 1 implementation new *
4     E remove();
  1 usage 1 implementation new *
5     boolean contains(E element);
  1 usage 1 implementation new *
6     boolean containsAll(Collection<E> collection);
  2 usages 1 implementation new *
7     boolean isEmpty();
8 }

```

Рисунок 14.1 – Код интерфейса IWaitList – методы add, remove, contains и containsAll, isEmpty

На рисунке 14.2 подробно изображен код методов contains, containsAll.

```

5 public class WaitList <E> implements IWaitList<E> {
  22 usages
6     protected ConcurrentLinkedQueue<E> components;
  4 usages new *
7     public WaitList() { components = new ConcurrentLinkedQueue<>(); }
  1 usage new *
8     public WaitList(Collection<E> collection) { components = new ConcurrentLinkedQueue<>(collection); }
  1 override new *
9     @Override
10    public String toString() { return "WaitList{" + "components=" + components + '}'; }
  15 usages 1 override new *
11    @Override
12    public void add(E element) { components.add(element); }
  2 usages new *
13    @Override
14    public E remove() { if(isEmpty()) throw new IllegalStateException("Очередь пуста!");
  21    return components.remove();
  new *
15    } @Override
16    public boolean contains(E element) { boolean res = false;
  24    for(int i = 0; i < components.size(); i++){ if el = components.remove();
  25    if(el.equals(element)) res = true;
  26    components.add(el);
  27    } return res;
  new *
17    } @Override
18    public boolean containsAll(Collection<E> collection) { ArrayList<E> al = new ArrayList<>(collection);
  30    for(int i = 0; i < collection.size(); i++){
  31    boolean res = false;
  32    for(int j = 0; j < components.size(); j++){ if el = components.remove();
  33    if(el.equals(al.get(i))) res = true;
  34    components.add(el);
  35    } if(!res) return false;
  36    } return true;
  37    }
  2 usages new *
19    @Override
20    public boolean isEmpty() { return components.isEmpty(); }
21 }

```

Рисунок 14.2 – Код класса IWaitList – работа методов contains и containsAll

На рисунке 14.3 представлен код remove и moveToBack.

```

1 public class UnfairWaitList<E> extends WaitList<E> {
2     1 usage new *
3     public UnfairWaitList() { super(); }
4     2 usages new *
5     public void remove (E element){
6         boolean removed = false;
7         for(int i = 0; i < components.size(); i++){
8             E el = components.remove();
9             if(!removed && el.equals(element)){
10                 removed = true; // i--
11             }else{
12                 components.add(el);
13             }
14         }
15         if(removed){
16             components.add(components.remove());
17         }
18     }
19     1 usage new *
20     public void moveToBack(E element){
21         int prevSize = components.size();
22         remove(element);
23         if(components.size() < prevSize){
24             components.add(element);
25         }
26     }
27 }

```

Рисунок 14.3 – Код remove и moveToBack

На рисунке 14.4 представлен код класса BoundedWaitList. Видны исключения в случае ошибки, а также работа методов getCapacity, add и toString.

```

1 public class BoundedWaitList<E> extends WaitList<E> {
2     0 usages
3     private int capacity;
4     1 usage new *
5     public BoundedWaitList(int capacity){
6         super();
7         if(capacity <= 0) throw new IllegalArgumentException("Максимальный размер должен быть больше 0! Получено значение: " + capacity);
8         this.capacity = capacity;
9     }
10    1 usage new *
11    public BoundedWaitList(Collection<E> col){
12        super(col);
13        this.capacity = col.size();
14    }
15    2 usages new *
16    public int getCapacity() { return capacity; }
17    14 usages new *
18    @Override
19    public void add(E element) {
20        if(components.size() == capacity) throw new IllegalStateException("Очередь заполнена!");
21        components.add(element);
22    }
23    new *
24    @Override
25    public String toString() {
26        return "BoundedWaitList{" +
27            "capacity=" + capacity +
28            ", components=" + components +
29            "}";
30    }
31 }

```

Рисунок 14.4 – Код BoundedWaitList – работа класса

Выводы по работе:

В ходе работы были получены знания по различным видам списков ожидания.

Практическая работа № 15

Цель работы

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

Теоретическое введение

Шаблон проектирования в программной инженерии — это метод решения часто возникающей проблемы при разработке программного обеспечения. Проектирование по модели указывает, какой тип архитектуры вы используете для решения проблемы или разработки модели. Существует два типа моделей проектирования: архитектура модели 1 и архитектура модели 2 (MVC). Проекты моделей, основанные на архитектуре MVC (model-viewcontroller), следуют шаблону проектирования MVC и отделяют логику приложения от пользовательского интерфейса при разработке программного обеспечения. Как следует из названия, шаблон MVC имеет три уровня: Модель — представляет бизнес-уровень приложения (model), Вид — определяет представление приложения (view), Контроллер — управляет потоком приложения (controller).

Выполнение лабораторной работы

Задание

Напишите реализацию программного кода по UML диаграмме, представленной на рисунке 15.1. Программа должна демонстрировать использование паттерна MVC. 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 EmployeeController будет действовать как уровень контроллера.

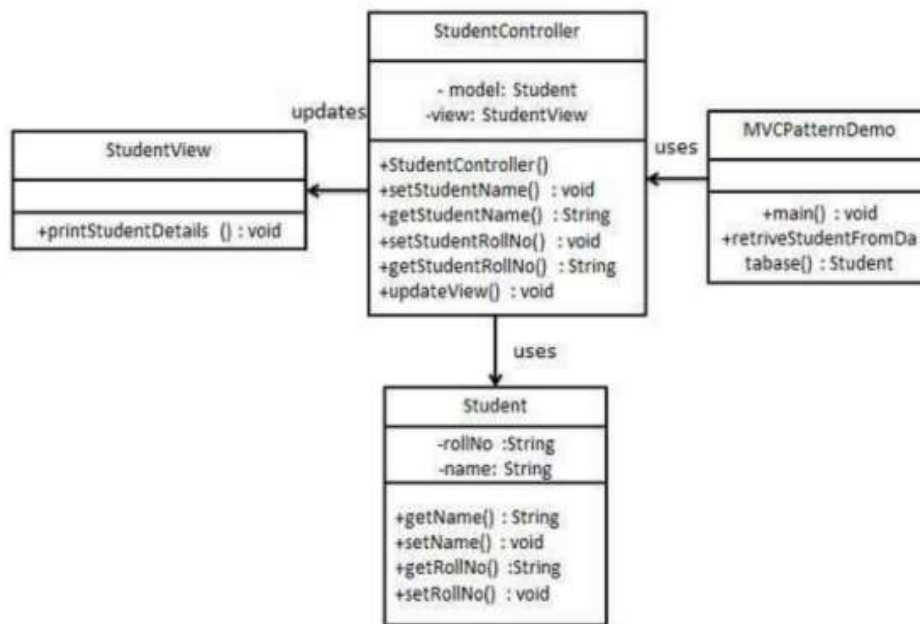


Рисунок 15.1 – UML диаграмма классов проекта, реализующего MVC

Решение

Разработка программы для задания 1.

На рисунке 15.2 представлена модель Student из задания 1. Есть роль и имя, а также методы эти данные возвращающие.

```

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

    2 usages
    public String getRollNo() {
        return rollNo;
    }

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

    public String getName() {
        return name;
    }

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

```

Рисунок 15.2 – Модель Student

На рисунке 15.3 изображен код представления модели Student из задания

1. На экран выведутся имя и роль студента.

```

public class StudentView {
    1 usage
    public void printStudentDetails(String studentName, String studentRollNo) {
        System.out.println("Студент: ");
        System.out.println("Имя: " + studentName);
        System.out.println("Роль: " + studentRollNo);
    }
}

```

Рисунок 15.3 – Код представления модели Student из задания 1

На рисунке 15.4 представлен код класса StudentController задания 1. Класс StudentController представляет контроллер в архитектуре MVC. Его главная задача - связывать модель (класс Student) и представление (класс StudentView) и обеспечивать взаимодействие между ними.

```
public class StudentController {  
    7 usages  
    private Student model;  
    2 usages  
    private StudentView view;  
  
    1 usage  
    public StudentController(Student model, StudentView view) {  
        this.model = model;  
        this.view = view;  
    }  
  
    1 usage  
    public void setStudentName(String name) { model.setName(name); }  
  
    no usages  
    public String getStudentName() { return model.getName(); }  
  
    1 usage  
    public void setStudentRollNo(String rollNo) { model.setRollNo(rollNo); }  
  
    no usages  
    public String getStudentRollNo() { return model.getRollNo(); }  
  
    1 usage  
    public void updateView() { view.printStudentDetails(model.getName(), model.getRollNo()); }  
}
```

Рисунок 15.4 – Код класса StudentController задания 1 – связка классов Student и StudentView

На рисунке 15.5 представлен код исполняемого класса задания 1. Это главный класс, который содержит метод main. Этот метод представляет точку входа в приложение. Сначала создается экземпляр Student, потом обновление данных этого класса, далее обновление представления и возврат с определенным именем и ролью.

```

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

        Student model = retrieveStudentFromDatabase();
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);

        controller.setStudentName("Иван");
        controller.setStudentRollNo("Куратор");

        controller.updateView();
    }

    1 usage
    public static Student retrieveStudentFromDatabase() {
        Student student = new Student();
        student.setName("Роберт");
        student.setRollNo("Староста");
        return student;
    }
}

```

Рисунок 15.5 – Код исполняемого класса задания 1 – работа классов Student, StudentView и StudentController, а также методов этих классов

На рисунке 15.6 представлен результат выполнения всей программы.

```

"C:\Program Files\Java\jdk-20\bin\java.exe"
Студент:
Имя: Иван
Роль: Куратор

Process finished with exit code 0

```

Рисунок 15.6 – Результат выполнения всей программы.

Разработка программы для задания 2.

На рисунке 15.7 представлена модель Employee из задания 2. Есть зарплата, имя и премия, а также методы считающие итоговую зарплату, задающие эти данные и возвращающие их.

```

class Employee {
    3 usages
    private String name;
    4 usages
    private double salary;
    4 usages
    private double bonus;
    1 usage
    public Employee(String name, double salary, double bonus) {
        this.name = name;
        this.salary = salary;
        this.bonus = bonus;
    }
    1 usage
    public double calculateTotalSalary() {
        return salary + bonus;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    1 usage
    public double getSalary() {
        return salary;
    }
    1 usage
    public void setSalary(double salary) {
        this.salary = salary;
    }
    1 usage
    public double getBonus() {
        return bonus;
    }
    1 usage
    public void setBonus(double bonus) {
        this.bonus = bonus;
    }
}

```

Рисунок 15.7 – Модель Employee задания 2

На рисунке 15.8 код представления модели Employee из задания 2. На экран выведутся имя, ЗП премия и итоговая ЗП работника.

```

class EmployeeView {
    1 usage
    public void printEmployeeDetails(String name, double salary, double bonus, double totalSalary) {
        System.out.println("Данные работника:");
        System.out.println("Имя: " + name);
        System.out.println("ЗП: " + salary);
        System.out.println("Премия: " + bonus);
        System.out.println("Итоговая ЗП: " + totalSalary);
    }
}

```

Рисунок 15.8 – Код представления модели Employee из задания 2

На рисунке 15.9 представлен код класса EmployeeController задания 2. Класс EmployeeController представляет контроллер в архитектуре MVC. Его главная задача - связывать модель (класс Employee) и представление (класс EmployeeView) и обеспечивать взаимодействие между ними.

```

class EmployeeController {
    8 usages
    private Employee model;
    2 usages
    private EmployeeView view;

    1 usage
    public EmployeeController(Employee model, EmployeeView view) {
        this.model = model;
        this.view = view;
    }

    1 usage
    public void updateEmployeeDetails(String name, double salary, double bonus) {
        model.setName(name);
        model.setSalary(salary);
        model.setBonus(bonus);
    }

    1 usage
    public void updateView() {
        double totalSalary = model.calculateTotalSalary();
        view.printEmployeeDetails(model.getName(), model.getSalary(), model.getBonus(), totalSalary);
    }
}

```

Рисунок 15.9 – Код класса EmployeeController задания 2 – связка модели (класс Employee) и представления (класс EmployeeView)

На рисунке 15.10 представлен код исполняемого класса задания 2. Это главный класс, который содержит метод main. Этот метод представляет точку входа в приложение. Сначала создается экземпляр Employee, потом

обновление данных этого класса, далее обновление представления и возврат с определенным именем и ролью.

```
public class MVCEmployeeDemo {  
    public static void main(String[] args) {  
        // Инициализируем модель, представление и контроллер  
        Employee model = new Employee( name: "Максим Лавров", salary: 50000, bonus: 2000);  
        EmployeeView view = new EmployeeView();  
        EmployeeController controller = new EmployeeController(model, view);  
  
        // Обновляем данные о сотруднике через контроллер  
        controller.updateEmployeeDetails( name: "Максим Лавров", salary: 60000, bonus: 2500);  
  
        // Обновляем представление  
        controller.updateView();  
    }  
}
```

Рисунок 15.10 – Код исполняемого класса задания 2 – работа классов Employee, EmployeeView и EmployeeController

На рисунке 15.11 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Данные работника:  
Имя: Макс Лавр  
ЗП: 60000.0  
Премия: 2500.0  
Итоговая ЗП: 62500.0  
  
Process finished with exit code 0
```

Рисунок 15.11 – Результат выполнения всей программы.

Выводы по работе:

В ходе работы были получены знания по применению событийного программирования и паттерна MVC.

Практическая работа № 16

Цель работы

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

Теоретическое введение

Механизм исключительных ситуаций в Java поддерживается пятью ключевыми словами:

- try
- catch
- finally
- throw
- throws

В языке Джава Java всего около 50 ключевых слов, и пять из них связано как раз с исключениями, это- try, catch, finally, throw, throws. Из них catch, throw и throws применяются к экземплярам класса, причём работают они только с Throwable и его наследниками.

Выполнение лабораторной работы

Задание

Задание 1: выполните следующую программу и посмотрите, что происходит:

Листинг 16.1 – Пример обработки деления на ноль

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

Задание 2: измените код в листинге 18.3 на следующий:

Листинг 16.2 – 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 );
    }
}

```

Добавить блоки try catch.

Задание 3: Добавьте ещё один catch в код задания 2.

Задание 4: Добавьте finally в код задания 2.

Решение

На рисунках 16.1 – 16.4 представлен код классов и изображения результатов работы.

```

1  ▶ public class Zadaniye_1 {
      1 usage new *
2      public void exceptionDemo1() {
3          System.out.println(2 / 0);
4      }
      1 usage new *
5      public void exceptionDemo2() {
6          System.out.println(2.0 / 0.0);
7      }
      1 usage new *
8      public void exceptionDemo3() {
9          try{
10             System.out.println( 2/0 );
11         } catch ( ArithmeticException e ) {
12             System.out.println("Attempted division by zero");
13         }
14     }
      new *
15  ▶ public static void main(String[] args) {
16      Zadaniye_1 task = new Zadaniye_1();
17      task.exceptionDemo2();
18      task.exceptionDemo3();
19      task.exceptionDemo1();
20  }
21  }

```

Рисунок 16.1 – Код первого задания – работа методов exceptionDemo1, exceptionDemo2 и exceptionDemo3

```

Infinity
Attempted division by zero
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Zadaniye_1.exceptionDemo1(Zadaniye_1.java:3)
    at Zadaniye_1.main(Zadaniye_1.java:19)

```

Рисунок 16.2 – Вывод первого задания

```

176 177
public class Zadanie_2 {
    1 usage new *
    public void exceptionDemo() {
        Scanner myScanner = new Scanner( System.in );
        System.out.print( "Enter an integer " );
        String intString = myScanner.next();
        int i = Integer.parseInt(intString);
        try {
            System.out.println( 2 / i );
        } catch ( ArithmeticException e ){
            System.out.println("Attempted division by zero");
        } catch (NumberFormatException e){
            System.out.println("Attempted division not by number");
        } finally {
            System.out.println("Try Catch Finaly");
        }
    }
}
new *
public static void main(String[] args) {
    Zadanie_2 task = new Zadanie_2();
    task.exceptionDemo();
}
}

```

Рисунок 16.3 – Код второго - четвёртого заданий – взаимодействие ключевых слов try, catch

```

Enter an integer 0
Attempted division by zero
Try Catch Finaly
Enter an integer 666pa
Attempted division not by number
Try Catch Finaly

```

Рисунок 16.4 – Результат выполнения программы

Выводы по работе:

В ходе работы были получены знания по написанию и применению исключений в Java.

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. В этой практической работе мы научимся создавать свои собственные пользовательские классы исключений.

Выполнение лабораторной работы

Задание

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.

Предлагается модернизировать задачу сортировки студентов по среднему баллу. Необходимо разработать пользовательский интерфейс для задачи поиска и сортировки. Дополнить ее поиском студента по фιο – в случае отсутствия такого студента необходимо выдавать собственное исключение. Схема классов программы приведена на рисунке 17.1.

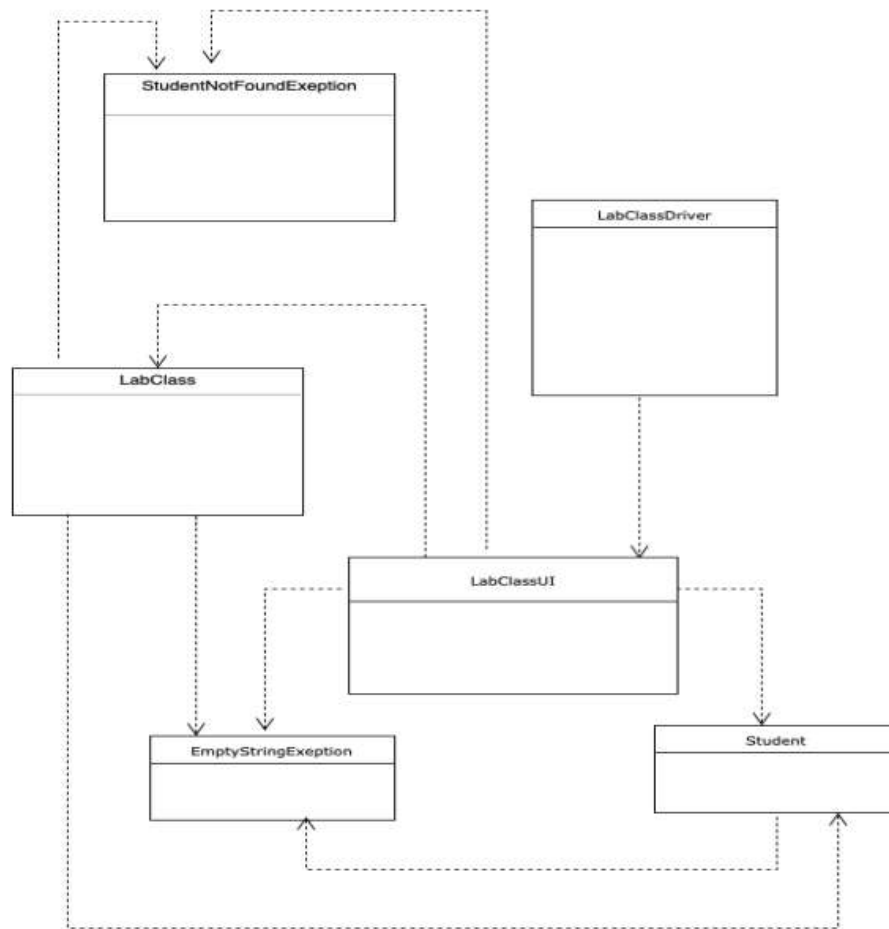


Рисунок 17.1 – UML диаграмма проекта

Решение

Разработка программы для задания 1.

На рисунке 17.2 представлен код программы, где происходит проверка по номеру ИНН, а также внедрены исключения, если номер не соответствует шаблону.

```

package Test.prak17;

import java.util.Scanner;

public class Pokupka {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.print("Введите ФИО: ");
            String fullName = scanner.nextLine();

            System.out.print("Введите номер (12 цифр): ");
            String number = scanner.nextLine();

            if (number.length() != 12) {
                throw new IllegalArgumentException("Недействительный номер. Номер должен содержать 12 цифр.");
            }

            if (!number.equals("102938475610")) {
                throw new IllegalArgumentException("Недействительный номер. Номер должен быть 102938475610.");
            }

            //можно добавить и другие эталонные номера, а также
            System.out.println("Заказ оформлен успешно для " + fullName + " с номером " + number);
        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}

```

Рисунок 17.2 – Код класса Рокурка задания 1 – связка условных конструкций и работа исключений

На рисунке 17.3 представлен результат выполнения всей программы.

```

Введите ФИО: Ионов Платон Борисович
Введите номер (12 цифр): 102938475610
Заказ оформлен успешно для Ионов Платон Борисович с номером 102938475610

```

Рисунок 17.3 – Результат выполнения всей программы

Разработка программы для задания 2.

На рисунке 17.4 представлена модернизированная задача сортировки студентов по среднему баллу. Есть метод сортирующий студентов по среднему баллу, а также метод их ищущий. Помимо этого, добавлено исключение, которое вызывается, если студент не найден.

```

package Test.prak17;

import java.util.Comparator;
import java.util.Scanner;

public class StudentMain {
    public static void main(String[] args) {
        Student[] students = {
            new Student( name: "Ионов Ф. П.", averageGrade: 4.5),
            new Student( name: "Петров Д. А.", averageGrade: 3.7),
            new Student( name: "Сидоров Р. С.", averageGrade: 4.0)
        };

        SortStrategy sortStrategy = new BubbleSortStrategy();
        SearchStrategy searchStrategy = new LinearSearchStrategy();

        // Сортировка по среднему баллу
        sortStrategy.sort(students, Comparator.comparingDouble(Student::getAverageGrade));

        // Запрос имени студента
        Scanner scanner = new Scanner(System.in);
        System.out.print("Введите имя студента: ");
        String studentName = scanner.nextLine();

        // Поиск студента по введенному имени
        try {
            Student foundStudent = searchStrategy.search(students, studentName);
            System.out.println("Найден студент: " + foundStudent.getName() + ", средний балл: " + foundStudent.getAverageGrade());
        } catch (StudentNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

Рисунок 17.4 – Код программы задания 2 – работа методов сортировки и поиска, а также работа исключения

На рисунке 17.5 представлен результат выполнения всей программы.

```

"C:\Program Files\Java\jdk-20\bin\java.exe" "-j;
Введите имя студента: Ионов Ф. П.
Найден студент: Ионов Ф. П., средний балл: 4.5

Process finished with exit code 0

```

Рисунок 17.5 – Результат выполнения всей программы

Выводы по работе:

В ходе работы были получены знания по написанию и применению исключений в Java.

Практическая работа № 18

Цель работы

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

Теоретическое введение

Понятие дженериков. Введение в Дженерики. В JDK представлены дженерики (перевод с англ. generics), которые поддерживают абстрагирование по типам (или параметризованным типам). В объявлении классов дженерики представлены обобщенными типами, в то время как пользователи классов могут быть конкретными типами, например во время создания объекта или вызова метода. Вы, конечно, знакомы с передачей аргументов при вызове методов в языке C++. 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

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.

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.

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.

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

Решение

Разработка программы для задания 1.

На рисунке 18.1 представлен код класса Generic с тремя параметрами, он содержит три переменные типа (T, V, K), конструктор, принимающий на вход параметры типа (T, V, K), методы возвращающие значения трех переменных и метод, выводящий на консоль имена классов для трех переменных класса.

```

package Test;

public class GenericClass<T, V, K> {
    3 usages
    private T t;
    3 usages
    private V v;
    3 usages
    private K k;

    1 usage
    public GenericClass(T t, V v, K k) {
        this.t = t;
        this.v = v;
        this.k = k;
    }

    no usages
    public T getT() {
        return t;
    }

    no usages
    public V getV() {
        return v;
    }

    no usages
    public K getK() {
        return k;
    }

    1 usage
    public void printClassNames() {
        System.out.println("Class name of T: " + t.getClass().getName());
        System.out.println("Class name of V: " + v.getClass().getName());
        System.out.println("Class name of K: " + k.getClass().getName());
    }

    public static void main(String[] args) {
        GenericClass<Integer, String, Double> genericInstance = new GenericClass<>(t 42, v "Hello", k 3.14);
        genericInstance.printClassNames();
    }
}

```

Рисунок 18.1 – Код класса Generic задания 1 – работа конструктора и методов

На рисунке 18.2 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Class name of T: java.lang.Integer  
Class name of V: java.lang.String  
Class name of K: java.lang.Double  
  
Process finished with exit code 0
```

Рисунок 18.2 – Результат выполнения всей программы

Разработка программы для задания 2.

На рисунке 18.3 представлен код класса `Animal` с ограничениями на параметры типа: `T` должен реализовать интерфейс `Comparable` (классы оболочки, `String`), `V` должен реализовать интерфейс `Serializable` и расширять класс `Animal`.

```

package Test.Prak18;
import java.io.Serializable;

1 inheritance
public class Animal<T extends Comparable<T>, V extends Animal & Serializable> {
    3 usages
    private T t;
    3 usages
    private V v;
    2 usages
    public Animal(T t, V v) {
        this.t = t;
        this.v = v;
    }
    1 usage
    public Animal() {
    }
    no usages
    public T getT() {
        return t;
    }
    no usages
    public V getV() {
        return v;
    }
    1 usage
    public void printClassNames() {
        System.out.println("Class name of T: " + t.getClass().getName());
        System.out.println("Class name of V: " + v.getClass().getName());
    }
    public static void main(String[] args) {
        Animal<Integer, Dog> animalInstance = new Animal<>(-42, new Dog());
        animalInstance.printClassNames();
    }
}

class Dog extends Animal implements Serializable {
    no usages
    public Dog(Comparable comparable, Animal animal) {
        super(comparable, animal);
    }
    1 usage
    public Dog() {
        super();
    }
}

```

Рисунок 18.3 – Код класса Animal задания 2 – реализация Comparable, Serializable

На рисунке 18.4 представлен результат выполнения всей программы.

```

"C:\Program Files\Java\jdk-20\bin\java.exe"
Class name of T: java.lang.Integer
Class name of V: Test.Prak18.Dog

Process finished with exit code 0

```

Рисунок 18.4 – Результат выполнения всей программы

Разработка программы для задания 3.1

На рисунке 18.5 представлен код обобщенного класса MinMax, который содержит методы для нахождения минимального и максимального элемента массива. Массив является переменной класса. Массив передается в класс через конструктор.

```
package Test;
public class MinMax<T extends Comparable<T>> {
    13 usages
    private T[] array;
    2 usages
    public MinMax(T[] array) {
        this.array = array;
    }
    2 usages
    public T findMin() {
        if (array == null || array.length == 0) {
            return null;
        }
        T min = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i].compareTo(min) < 0) {
                min = array[i];
            }
        }
        return min;
    }
    2 usages
    public T findMax() {
        if (array == null || array.length == 0) {
            return null;
        }
        T max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i].compareTo(max) > 0) {
                max = array[i];
            }
        }
        return max;
    }
    public static void main(String[] args) {
        Integer[] intArray = { 5, 2, 9, 1, 7 };
        MinMax<Integer> intMinMax = new MinMax<>(intArray);
        System.out.println("Min: " + intMinMax.findMin());
        System.out.println("Max: " + intMinMax.findMax());

        Double[] doubleArray = { 3.14, 2.71, 1.618, 0.0, 42.0 };
        MinMax<Double> doubleMinMax = new MinMax<>(doubleArray);
        System.out.println("Min: " + doubleMinMax.findMin());
        System.out.println("Max: " + doubleMinMax.findMax());
    }
}
```

Рисунок 18.5 – Код класса MinMax задания 3 – взаимодействие с массивом типа T, а также работа методов findMin() и findMax()

На рисунке 18.6 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Min: 1  
Max: 9  
Min: 0.0  
Max: 42.0  
  
Process finished with exit code 0
```

Рисунок 18.6 – Результат выполнения всей программы

Разработка программы для задания 3.2.

На рисунке 18.7 представлен код необобщенного класса Калькулятор, который содержит обобщенные статические методы - sum, multiply, divide, subtraction. Параметры этих методов - два числа разного типа, над которыми производится операция.

2 usages

```
public static <T extends Number> T sum(T a, T b) {  
    if (a instanceof Integer) {  
        return (T) Integer.valueOf( a.intValue() + b.intValue());  
    } else if (a instanceof Double) {  
        return (T) Double.valueOf( a.doubleValue() + b.doubleValue());  
    }  
    return null;  
}
```

2 usages

```
public static <T extends Number> T multiply(T a, T b) {  
    if (a instanceof Integer) {  
        return (T) Integer.valueOf( a.intValue() * b.intValue());  
    } else if (a instanceof Double) {  
        return (T) Double.valueOf( a.doubleValue() * b.doubleValue());  
    }  
    return null;  
}
```

2 usages

```
public static <T extends Number> T divide(T a, T b) {  
    if (a instanceof Integer) {  
        return (T) Integer.valueOf( a.intValue() / b.intValue());  
    } else if (a instanceof Double) {  
        return (T) Double.valueOf( a.doubleValue() / b.doubleValue());  
    }  
    return null;  
}
```

2 usages

```
public static <T extends Number> T subtract(T a, T b) {  
    if (a instanceof Integer) {  
        return (T) Integer.valueOf( a.intValue() - b.intValue());  
    } else if (a instanceof Double) {  
        return (T) Double.valueOf( a.doubleValue() - b.doubleValue());  
    }  
    return null;  
}
```

Рисунок 18.7 – Код класса Calculator задания 3.2 – работа методов sum(), multiply(), divide(), subtract()

На рисунке 18.8 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Sum: 15  
Multiply: 50  
Divide: 2  
Subtract: 5  
Sum: 10.5  
Multiply: 22.5  
Divide: 2.5  
Subtract: 4.5  
  
Process finished with exit code 0
```

Рисунок 18.8 – Результат выполнения всей программы

Разработка программы для задания 4.

На листинге 1.1 представлен код обобщенного класса `Matrix`, который реализует операции с матрицами.

Листинг 18.1 – Код класса `Matrix` задания 4 – реализация операций над матрицами

```
package Test;  
  
public class Matrix<T> {  
    private final int rows;  
    private final int columns;  
    private final T[][] data;  
    public Matrix(int rows, int columns) {  
        this.rows = rows;  
        this.columns = columns;  
        this.data = (T[][]) new Object[rows][columns];  
    }  
    public Matrix(T[][] data) {  
        this.rows = data.length;  
        this.columns = data[0].length;  
        this.data = data;  
    }  
    public int getRows() {  
        return rows;  
    }  
}
```



```

public int getColumns() {
    return columns;
}

public T get(int row, int column) {
    if (row < 0 || row >= rows || column < 0 || column >= columns) {
        throw new IndexOutOfBoundsException("Invalid matrix indices");
    }
    return data[row][column];
}

public void set(int row, int column, T value) {
    if (row < 0 || row >= rows || column < 0 || column >= columns) {
        throw new IndexOutOfBoundsException("Invalid matrix indices");
    }
    data[row][column] = value;
}

public Matrix<T> add(Matrix<T> other) {
    if (rows != other.getRows() || columns != other.getColumns()) {
        throw new IllegalArgumentException("Matrix dimensions must match
for addition");
    }
    Matrix<T> result = new Matrix<>(rows, columns);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            T sum = (T) addElements(data[i][j], other.get(i, j));
            result.set(i, j, sum);
        }
    }
    return result;
}

public Matrix<T> multiply(Matrix<T> other) {
    if (columns != other.getRows()) {
        throw new IllegalArgumentException("Number of columns in the
first matrix must match the number of rows in the second matrix for
multiplication");
    }

    int resultRows = rows;
    int resultColumns = other.getColumns();
    Matrix<T> result = new Matrix<>(resultRows, resultColumns);

    for (int i = 0; i < resultRows; i++) {
        for (int j = 0; j < resultColumns; j++) {

```

```

        T sum = (T) multiplyElements(data[i][j], other, j);
        result.set(i, j, sum);
    }
}
return result;
}

private T addElements(T a, T b) {
    if (a instanceof Integer) {
        return (T) (Integer.valueOf(((Integer) a) + ((Integer) b)));
    } else if (a instanceof Double) {
        return (T) (Double.valueOf(((Double) a) + ((Double) b)));
    } else {
        throw new IllegalArgumentException("Unsupported data type");
    }
}

private T multiplyElements(T a, Matrix<T> other, int columnIndex) {
    if (a instanceof Integer) {
        int sum = 0;
        for (int i = 0; i < columns; i++) {
            sum += ((Integer) a) * ((Integer) other.get(i, columnIndex));
        }
        return (T) (Integer.valueOf(sum));
    } else if (a instanceof Double) {
        double sum = 0.0;
        for (int i = 0; i < columns; i++) {
            sum += ((Double) a) * ((Double) other.get(i, columnIndex));
        }
        return (T) (Double.valueOf(sum));
    } else {
        throw new IllegalArgumentException("Unsupported data type");
    }
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            sb.append(data[i][j]).append("\t");
        }
    }
}

```

```
        sb.append("\n");
    }
    return sb.toString();
}

public static void main(String[] args) {
    // Пример использования класса Matrix
    Integer[][] data1 = {
        {1, 2},
        {3, 4}
    };
    Integer[][] data2 = {
        {5, 6},
        {7, 8}
    };
    Matrix<Integer> matrix1 = new Matrix<>(data1);
    Matrix<Integer> matrix2 = new Matrix<>(data2);
    System.out.println("Matrix 1:");
    System.out.println(matrix1);
    System.out.println("Matrix 2:");
    System.out.println(matrix2);
    Matrix<Integer> sumMatrix = matrix1.add(matrix2);
    System.out.println("Sum Matrix:");
    System.out.println(sumMatrix);
    Matrix<Integer> productMatrix = matrix1.multiply(matrix2);
    System.out.println("Product Matrix:");
    System.out.println(productMatrix);
}
}
```

На рисунке 18.9 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"  
Matrix 1:  
1  2  
3  4  
  
Matrix 2:  
5  6  
7  8  
  
Sum Matrix:  
6  8  
10 12  
  
Product Matrix:  
12 28  
36 56  
  
Process finished with exit code 0
```

Рисунок 18.9 – Результат выполнения всей программы

Выводы по работе:

В ходе работы были получены знания по написанию и применению дженериков в Java.

Практическая работа № 19

Цель работы

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

Теоретическое введение

Обобщенные типы (Generics) в Java позволяют создавать классы, интерфейсы и методы, которые могут работать с различными типами данных, сохраняя при этом типовую безопасность. Обобщенные типы позволяют написать более универсальный и безопасный код, так как типы данных определяются динамически. Принцип обобщенных типов основан на параметризации типов. Вместо конкретных типов данных, используются типовые параметры (типовые переменные)

Стирание типов (Type Erasure) в Java происходит во время компиляции и означает, что параметры типов удаляются из скомпилированного байткода. Это делается для обеспечения обратной совместимости с предыдущими версиями Java, которые не поддерживали обобщенные типы. После стирания типов обобщенный код работает с Object, и компилятор вставляет приведения типов (cast) при необходимости.

Выполнение лабораторной работы

Задание

Написать метод для конвертации массива строк/чисел в список.

Написать класс, который умеет хранить в себе массив любых типов данных (int, long etc.).

Реализовать метод, который возвращает любой элемент массива по индексу.

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

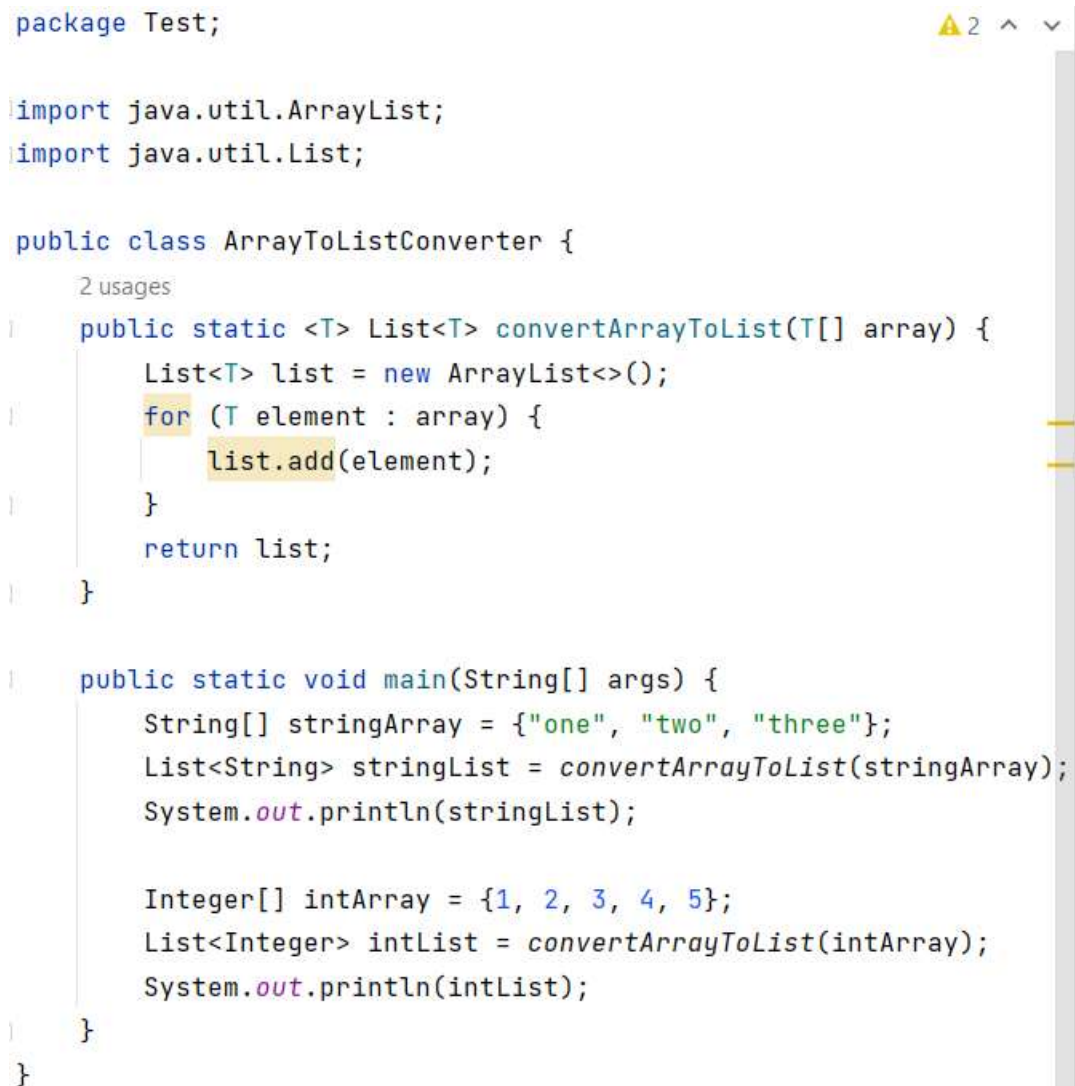
*Реализуйте вспомогательные методы в классе Solution, которые

должны создавать соответствующую коллекцию и помещать туда переданные объекты. Методы `newArrayList`, `newHashSet` параметризуйте общим типом `T`. Метод `newHashMap` параметризуйте парой `<K, V>`, то есть типами `K`-ключ и `V`-значение. Аргументы метода `newHashMap` должны принимать. Класс содержит три переменные типа `(T, V, K)`, конструктор, принимающий на вход

Решение

Разработка программы для задания 1.

На рисунке 19.1 представлен код класса `ArrayToListConverter`. С помощью функции `convertArrayToList` происходит преобразование массива строк/чисел в список.

The image shows a screenshot of a code editor with Java code. The code defines a package 'Test', imports 'java.util.ArrayList' and 'java.util.List', and defines a class 'ArrayToListConverter'. The class has two methods: 'convertArrayToList' which takes an array of type 'T' and returns a 'List<T>', and a 'main' method that demonstrates the conversion of string and integer arrays to lists. The 'convertArrayToList' method is highlighted with a yellow background. The 'main' method is also highlighted with a yellow background. The code is as follows:

```
package Test;

import java.util.ArrayList;
import java.util.List;

public class ArrayToListConverter {
    2 usages
    public static <T> List<T> convertArrayToList(T[] array) {
        List<T> list = new ArrayList<>();
        for (T element : array) {
            list.add(element);
        }
        return list;
    }

    public static void main(String[] args) {
        String[] stringArray = {"one", "two", "three"};
        List<String> stringList = convertArrayToList(stringArray);
        System.out.println(stringList);

        Integer[] intArray = {1, 2, 3, 4, 5};
        List<Integer> intList = convertArrayToList(intArray);
        System.out.println(intList);
    }
}
```

Рисунок 19.1 – Код класса `ArrayToListConverter` задания 1 – работа функции преобразования

На рисунке 19.2 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\j  
[one, two, three]  
[1, 2, 3, 4, 5]  
  
Process finished with exit code 0
```

Рисунок 19.2 – Результат выполнения всей программы

Разработка программы для задания 2 и 3.

На рисунке 19.3 представлен код класса `MyArray`, который хранит в себе массив любых типов данных и который имеет метод `getElementByIndex`, служащий для возврата любого элемента массива по индексу.

```
package Test;
```

1 ^

```
public class MyArray<T> {
```

3 usages

```
private T[] array;
```

1 usage

```
1 public MyArray(T[] array) {  
2     this.array = array;  
3 }  
4 }
```

1 usage

```
1 public T getElementByIndex(int index) {  
2     if (index >= 0 && index < array.length) {  
3         return array[index];  
4     }  
5     return null;  
6 }  
7 }
```

```
1 public static void main(String[] args) {  
2     Integer[] intArray = {1, 2, 3, 4, 5};  
3     MyArray<Integer> intArrayWrapper = new MyArray<>(intArray);  
4  
5     System.out.println(intArrayWrapper.getElementByIndex(2));  
6 }  
7 }
```

Рисунок 19.3 – Код класса MyArray заданий 2 и 3 – реализация хранения любых типов данных и getElementByIndex

На рисунке 19.4 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe"
```

```
3
```

```
Process finished with exit code 0
```

Рисунок 19.4 – Результат выполнения всей программы

Разработка программы для задания 4.

На рисунке 19.5 представлен код класса Directory, который имеет функцию сохранения содержимого каталога в качестве списка и вывода первых 5 элементов на экран.

```
package Test;

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

public class Directory {
    1 usage
    public static List<String> getDirectoryContent(String directoryPath) {
        List<String> fileList = new ArrayList<>();
        File directory = new File(directoryPath);

        if (directory.isDirectory()) {
            File[] files = directory.listFiles();
            if (files != null) {
                for (int i = 0; i < Math.min(5, files.length); i++) {
                    fileList.add(files[i].getName());
                }
            }
        }

        return fileList;
    }

    public static void main(String[] args) {
        String directoryPath = "C:/Program Files/Java/jdk-20";
        List<String> fileList = getDirectoryContent(directoryPath);

        for (String fileName : fileList) {
            System.out.println(fileName);
        }
    }
}
```

Рисунок 19.5 – Код класса Directory задания 4 – работа функции getDirectoryContent

На рисунке 19.6 представлен результат выполнения всей программы.

```
"C:\Program Files\Java\jdk-20\bin\java.exe'  
bin  
conf  
include  
jmods  
legal  
  
Process finished with exit code 0
```

Рисунок 19.6 – Результат выполнения всей программы

Выводы по работе:

В ходе работы были получены знания по написанию, применению и стиранию обобщенных типов в Java.

Практическая работа № 20

Цель работы

Научиться разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

Теоретическое введение

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

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

Стеку присущи следующие характеристики:

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

Выполнение лабораторной работы

Задание

Задание 1. Напишите программу-калькулятор арифметических выражений записанных в обратной польской нотации (RPN-калькулятор).

Задание 2. Напишите графический интерфейс для калькулятора, используя знания, полученные ранее при программировании GUI с использованием SWING и AWT. Используйте паттерн проектирования MVC.

Задание 3. Постройте систему тестов и проверьте, что ваш калькулятор успешно проходит все тесты и «защищён от дурака» (как дурака-пользователя программы, так и дурака-программиста, использующего ваш стек и калькулятор). Например, если вводится выражение, в котором число операций превосходит число помещенных в стек элементов (например, $1\ 2\ +\ *$), то программа не допустит уменьшения переменной `sp` до отрицательных значений, а выдаст предупреждение «Невозможно выполнить POP для пустого стека».

Решение

Разработка программы для задания 1 и 3.

На листинге 20.1 представлен код класса `RPNCalculator`. С помощью метода `evaluateRPN` возвращается результат вычисления, метода `isNumeric` проверяется строка число или нет, метода `isOperator` проверяется оператор строка или нет. Также внесены исключения и проверка на дурака.

Листинг 20.1 – Код класса `RPNCalculator` задания 1 – реализация операций

```
package Calculator20prak;

import java.util.Stack;

public class RPNCalculator {
    public static double evaluateRPN(String expression) {
        String[] tokens = expression.split("\\s+");
        Stack<Double> stack = new Stack<>();

        for (String token : tokens) {
            if (isNumeric(token)) {
                stack.push(Double.parseDouble(token));
            } else if (isOperator(token)) {
                double operand2 = stack.pop();
                double operand1 = stack.pop();
                double result = performOperation(operand1, operand2, token);
            }
        }
        return stack.pop();
    }
}
```

```

        stack.push(result);
    } else {
        throw new IllegalArgumentException("Invalid token: " +
token);
    }
}

    if (stack.size() != 1) {
        throw new IllegalArgumentException("Invalid expression: " +
expression);
    }

    return stack.pop();
}

private static boolean isNumeric(String str) {
    return str.matches("-?\\d+(\\.\\d+)?");
}

private static boolean isOperator(String str) {
    return str.equals("+") || str.equals("-") || str.equals("*") ||
str.equals("/");
}

private static double performOperation(double operand1, double operand2,
String operator) {
    switch (operator) {
        case "+":
            return operand1 + operand2;
        case "-":
            return operand1 - operand2;
        case "*":
            return operand1 * operand2;
        case "/":
            if (operand2 == 0) {
                throw new ArithmeticException("Division by zero");
            }
            return operand1 / operand2;
        default:
            throw new IllegalArgumentException("Invalid operator: " +
operator);
    }
}

```

```

    }

    public static void main(String[] args) {
        String expression = "3 4 + 2 *";
        double result = evaluateRPN(expression);
        System.out.println("Result: " + result);
    }
}

```

На рисунке 20.1 представлен результат выполнения всей программы.

```

"C:\Program Files\Java\jdk-20\bin\java.exe"
Result: 14.0

Process finished with exit code 0

```

Рисунок 20.1 – Результат выполнения всей программы

Разработка программы для задания 2.

На листинге 20.2 представлен код класса CalculatorApp. Он представляет точку входа в программу и отвечает за создание и инициализацию модели, представления и контроллера калькулятора.

Листинг 20.2 – Код класса CalculatorApp задания 2 – реализация операций

```

package Calculator20prak;

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

// Модель
class CalculatorModel {
    private String expression = "";

    public void appendToExpression(String input) {
        expression += input;
    }
}

```

```

    public void clearExpression() {
        expression = "";
    }

    public double evaluateExpression() {
        return RPNCalculator.evaluateRPN(expression);
    }

    public String getExpression() {
        return expression;
    }
}

// Представление
class CalculatorView extends JFrame {
    private JTextField display;
    private CalculatorController controller;

    public CalculatorView(CalculatorController controller) {
        this.controller = controller;
        setTitle("RPN Calculator");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        display = new JTextField();
        display.setEditable(false);
        add(display, BorderLayout.NORTH);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(4, 4));

        String[] buttonLabels = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "0", ".", "=", "+"
        };

        for (String label : buttonLabels) {
            JButton button = new JButton(label);
            button.addActionListener(new ButtonClickListener());

```

```

        buttonPanel.add(button);
    }

    add(buttonPanel, BorderLayout.CENTER);

    pack();
    setLocationRelativeTo(null);
}

public void updateDisplay(String text) {
    display.setText(text);
}

private class ButtonClickListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton source = (JButton) e.getSource();
        controller.processButtonClick(source.getText());
    }
}

public void setController(CalculatorController controller) {
    this.controller = controller;
}
}

// Контроллер
class CalculatorController {
    private CalculatorModel model;
    private CalculatorView view;

    public CalculatorController(CalculatorModel model, CalculatorView view)
    {
        this.model = model;
        this.view = view;
        view.setController(this);
    }

    public void processButtonClick(String buttonLabel) {
        switch (buttonLabel) {
            case "=":
                try {

```



```

        double result = model.evaluateExpression();
        view.updateDisplay(Double.toString(result));
    } catch (IllegalArgumentException ex) {
        view.updateDisplay("Error");
    }
    model.clearExpression();
    break;
default:
    model.appendToExpression(buttonLabel);
    view.updateDisplay(model.getExpression());
}
}
}

public class CalculatorApp {
    public static void main(String[] args) {
        CalculatorModel model = new CalculatorModel();
        CalculatorView view = new CalculatorView(null);
        CalculatorController controller = new CalculatorController(model,
view);
        view.setController(controller);

        SwingUtilities.invokeLater(() -> {
            view.setVisible(true);
        });
    }
}

```

На рисунке 20.2 представлен результат выполнения всей программы.

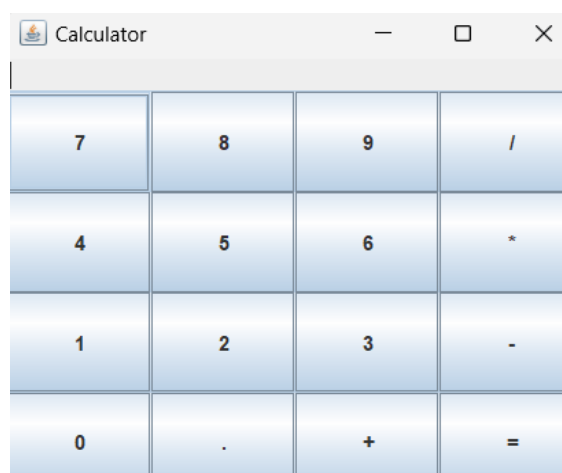


Рисунок 20.4 – Результат выполнения всей программы

Выводы по работе:

В ходе работы были получены знания по написанию и применению программ с абстрактными типами данных в Java.

Листинг первой части кода задания 1 практической работы №15

```
// Knack StudentController
// usage
public class StudentController {
    // usage
    private Student model;
    // usage
    private StudentView view;

    // usage
    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    // usage
    public void setStudentName(String name) { model.setName(name); }

    // usage
    public String getStudentName() { return model.getName(); }

    // usage
    public void setStudentRollNo(String rollNo) { model.setRollNo(rollNo); }

    // usage
    public String getStudentRollNo() { return model.getRollNo(); }

    // usage
    public void updateView() { view.printStudentDetails(model.getName(), model.getRollNo()); }
}

// Knack MVCPatternDemo
public class MVCPatternDemo {
    public static void main(String[] args) {

        Student model = retrieveStudentFromDatabase();
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);

        controller.setStudentName("Kash");
        controller.setStudentRollNo("Kyparop");

        controller.updateView();
    }

    // usage
    public static Student retrieveStudentFromDatabase() {
        Student student = new Student();
        student.setName("Роберт");
        student.setRollNo("Староста");
        return student;
    }
}
```

Рисунок 20.5 – Листинг второй части кода задания 1 практической работы №15

```

package Test;

// Модель
4 usages
class Employee {
    3 usages
    private String name;
    4 usages
    private double salary;
    4 usages
    private double bonus;
    1 usage
    public Employee(String name, double salary, double bonus) {
        this.name = name;
        this.salary = salary;
        this.bonus = bonus;
    }
    1 usage
    public double calculateTotalSalary() { return salary + bonus; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    1 usage
    public double getSalary() { return salary; }
    1 usage
    public void setSalary(double salary) { this.salary = salary; }
    1 usage
    public double getBonus() { return bonus; }
    1 usage
    public void setBonus(double bonus) { this.bonus = bonus; }
}

// Представление
4 usages
class EmployeeView {
    1 usage
    public void printEmployeeDetails(String name, double salary, double bonus, double totalSalary) {
        System.out.println("Данные работника:");
        System.out.println("Имя: " + name);
        System.out.println("ЗП: " + salary);
        System.out.println("Премия: " + bonus);
        System.out.println("Итоговая ЗП: " + totalSalary);
    }
}

```

Рисунок 20.6 – Листинг первой части кода задания 2 практической работы №15

```

// Контроллер
2 usages
class EmployeeController {
    8 usages
    private Employee model;
    2 usages
    private EmployeeView view;

    1 usage
    public EmployeeController(Employee model, EmployeeView view) {
        this.model = model;
        this.view = view;
    }

    1 usage
    public void updateEmployeeDetails(String name, double salary, double bonus) {
        model.setName(name);
        model.setSalary(salary);
        model.setBonus(bonus);
    }

    1 usage
    public void updateView() {
        double totalSalary = model.calculateTotalSalary();
        view.printEmployeeDetails(model.getName(), model.getSalary(), model.getBonus(), totalSalary);
    }
}

public class MVCEmployeeDemo {
    public static void main(String[] args) {
        // Инициализируем модель, представление и контроллер
        Employee model = new Employee( name: "Максим Лавров", salary: 50000, bonus: 2000);
        EmployeeView view = new EmployeeView();
        EmployeeController controller = new EmployeeController(model, view);

        // Обновляем данные о сотруднике через контроллер
        controller.updateEmployeeDetails( name: "Макс Лавр", salary: 60000, bonus: 2500);

        // Обновляем представление
        controller.updateView();
    }
}

```

Рисунок 20.7 – Листинг второй части кода задания 2 практической работы №15

Практическая работа № 21

Цель работы

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

Теоретическое введение

Теоретические сведения. Стирание типов 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.

Помните! Что Дженерики реализуются компилятором Java в качестве интерфейсного преобразования, называемого стиранием, которое переводит или перезаписывает код, который использует дженерики в не обобщенный код (для обеспечения обратной совместимости). Это преобразование стирает всю информацию об общем типа. Например, ArrayList станет ArrayList. Параметр формального типа, такой как , заменяется объектом по умолчанию (или верхней границей типа). Когда результирующий код не корректен, компилятор вставляет оператор преобразования типа. Когда класс создается с использованием актуального или фактического параметра типа, например. MyGenericArrayList , компилятор гарантирует, что add(E e) работает только с типом String. Он также вставляет соответствующий оператор понижающее преобразование типов в соответствие с типом возвращаемого значения E для метода get().

Выполнение лабораторной работы

Задание

Задание 1. Написать метод для конвертации массива строк/чисел в список.

Задание 2. Написать класс, который умеет хранить в себе массив любых типов данных (int, long etc.).

Решение

Решение задачи 1 представлено на рисунке 21.1.

```
public class AnyTypeArray {
    private Object[] array;
    private int size;

    public AnyTypeArray(int num) {
        if (num <= 0) {
            throw new IllegalArgumentException("Capacity must be a positive value.");
        }
        array = new Object[num];
        size = 0;
    }

    public void add(Object element) {
        if (size >= array.length) {
            // Увеличиваем размер массива при необходимости
            int newCapacity = array.length * 2;
            Object[] newArray = new Object[newCapacity];
            System.arraycopy(array, 0, newArray, 0, array.length);
            array = newArray;
        }
        array[size++] = element;
    }

    public Object get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("Index is out of bounds.");
        }
        return array[index];
    }

    public int size() { return size; }
}
```

Рисунок 21.1 – Код класса для решения задачи 1

Решение задачи 2 представлено на рисунке 21.2.

```
public class AnyTypeArray {  
    private Object[] array;  
    private int size;  
  
    public AnyTypeArray(int num) {  
        if (num <= 0) {  
            throw new IllegalArgumentException("Capacity must be a positive value.");  
        }  
        array = new Object[num];  
        size = 0;  
    }  
  
    public void add(Object element) {  
        if (size >= array.length) {  
            // Увеличиваем размер массива при необходимости  
            int newCapacity = array.length * 2;  
            Object[] newArray = new Object[newCapacity];  
            System.arraycopy(array, 0, newArray, 0, array.length);  
            array = newArray;  
        }  
        array[size++] = element;  
    }  
  
    public Object get(int index) {  
        if (index < 0 || index >= size) {  
            throw new IndexOutOfBoundsException("Index is out of bounds.");  
        }  
        return array[index];  
    }  
  
    public int size() { return size; }  
}
```

Рисунок 21.2 – Код класса для решения задачи 2

Практическая работа № 22 Абстрактные типы данных. Стек

Цель работы

Научиться разрабатывать программы с абстрактными типами данных на языке Джава и применять паттерн MVC при разработке программ.

Теоретическое введение

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

- Примеры стеков – пирамида, стопка тарелок или книг, магазин в пистолете. Стек имеет следующие характеристики:
- Стек — это абстрактный тип данных с заранее определенной емкостью, что означает, что эта структура данных имеет ограниченный размер, то есть может хранить количество элементов, определенное размерностью стека.

Это структура данных, в которой строго определен порядок вставки и удаления элементов, и этот порядок может быть LIFO или FILO.

Порядок работы со стеком

Рассмотрим пример, допустим стек работает по схеме LIFO. Как видно на рис. ниже, в стеке пять блоков памяти; поэтому размер стека равен 5. Предположим, мы хотим хранить элементы в стеке, и предположим, что в начале стек пуст. Мы приняли размер стека равным 5, как показано на рис.22.1 ниже, в который мы будем помещать элементы один за другим, пока стек не заполнится.

Поскольку наш стек заполнен, то количество элементов в нем равно 5. В приведенных выше случаях мы можем наблюдать, что он заполняется элементами снизу вверх, при каждом добавлении нового элемента в стек. Стек растет снизу вверх.

Стандартные операции со стеком

Ниже приведены некоторые общие операции, реализованные в стеке:

- `push()`: когда мы добавляем элемент в стек, эта операция называется `push`. Если стек заполнен, возникает состояние переполнения.
- `pop()`: Когда мы удаляем элемент из стека, эта операция называется `pop`. Если стек пуст, это означает, что в стеке нет элементов, это состояние известно как состояние потери значимости.
- `isEmpty()`: определяет, пуст стек в настоящий момент или нет.
- `isFull()`: определяет, заполнен стек или нет.
- `peek()`: возвращает элемент в заданной позиции.
- `count()`: возвращает общее количество элементов, доступных в стеке.
- `change()`: изменяет элемент в заданной позиции.
- `display()`: печатает все элементы, доступные в стеке.

Выполнение лабораторной работы

Задание

Задание 1. Напишите программу-калькулятор арифметических выражений записанных в обратной польской нотации (RPN-калькулятор).

Задание 2. Напишите графический интерфейс для калькулятора, используя знания полученные ранее при программировании GUI с использованием SWING и AWT. Используйте паттерн проектирования MVC.

Решение

Для выполнения следующих двух заданий была написана логика калькулятора с использованием паттерна проектирования MVC.

На рисунке 22.1 представлены классы для выполнения заданий 1 и 2.

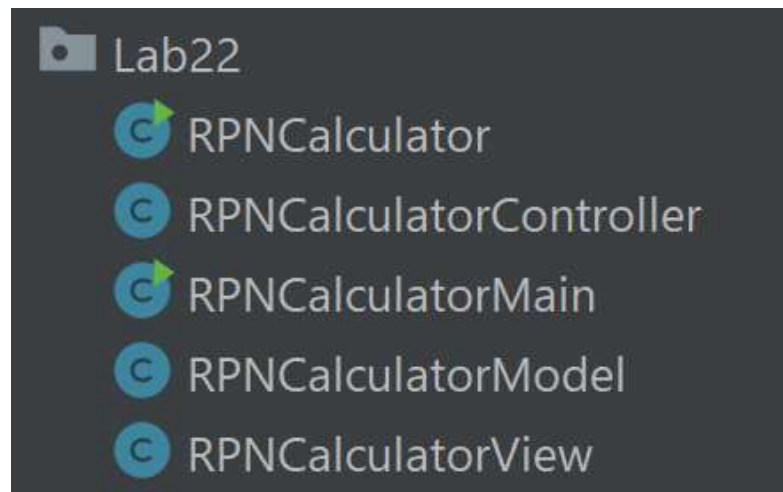


Рисунок 22.1 – Классы реализующие 1 и 2 задания

Задание 1

На рисунках 22.2 и 22.3 представлены основные методы калькулятора, к которым будет обращаться контроллер в будущем.

```
public static boolean isNumber(String symb) {  
    try {  
        Double.parseDouble(symb);  
        return true;  
    } catch (Exception e) {  
        return false;  
    }  
}  
  
public static boolean isOper(String symb) {  
    return symb.equals("+") || symb.equals("-") || symb.equals("*") || symb.equals("/");  
}  
  
public double operation(Double num1, Double num2, String oper) {  
    switch (oper) {  
        case "+" -> {  
            return num1 + num2;  
        }  
        case "-" -> {  
            return num1 - num2;  
        }  
        case "*" -> {  
            return num1 * num2;  
        }  
        case "/" -> {  
            return num1 / num2;  
        }  
        default -> throw new IllegalArgumentException("Неверный оператор");  
    }  
}
```

Рисунок 22.2 – Первая часть кода, реализовывающего калькулятор

```

public void calculate() {
    String[] expr = expression.split(режек: " ");
    Stack<Double> stack = new Stack<>();
    for (String elem : expr) {
        if (isNumber(elem)) {
            stack.push(Double.parseDouble(elem));
        } else if (isOper(elem)) {
            if (stack.size() < 2) {
                throw new IllegalArgumentException("Недостаточное количество чисел");
            } else {
                double first = stack.pop();
                double second = stack.pop();
                stack.push(operation(second, first, elem));
            }
        }
    }
    expression = String.valueOf(stack.pop());
}
}

```

Рисунок 22.3 – Вторая часть кода, реализовывающего калькулятор

На рисунке 22.4 показан контроллер калькулятора, который лишь вызывает методы основного реализовывающего его класса.

```

private static void setupButtons(RPNCalculatorModel model, RPNCalculatorView view) {
    for (JButton button : view.basicButtons) {
        button.addActionListener(e -> {
            JButton button1 = (JButton) e.getSource();
            String but = button1.getText();
            model.addToExpression(but);
            view.setTextField(model.getExpression());
        });
    }

    view.clearButton.addActionListener(e -> {
        model.clearExpression();
        view.setTextField(model.getExpression());
    });

    view.backspaceButton.addActionListener(e -> {
        model.backspaceExpression();
        view.setTextField(model.getExpression());
    });

    view.spaceButton.addActionListener(e -> {
        model.addToExpression(" ");
        view.setTextField(model.getExpression());
    });

    view.equalButton.addActionListener(e -> {
        try {
            model.calculate();
            view.setTextField(model.getExpression());
        } catch (IllegalArgumentException | ArithmeticException e1) {
            System.out.println("Error: " + e1.getMessage());
        }
    });
}
}

```

Рисунок 22.3 – Вторая часть кода, реализовывающего калькулятор

Задание 2

Для визуального интерфейса калькулятору необходим класс view, показан на рисунке 22.4

```
public class RPNCalculatorView {  
    private JFrame frame;  
    private JTextField textField;  
    JButton[] basicButtons;  
    JButton equalButton;  
    JButton spaceButton;  
    JButton backspaceButton;  
    JButton clearButton;  
  
    public RPNCalculatorView() {...}  
  
    public void setTextField(String text) { textField.setText(text); }  
  
    public String getTextField() { return textField.getText(); }  
}
```

Рисунок 22.4 – Код описывающий класс view

Выводы по работе

Следуя указаниям из методических указаний и строго следуя шаблону проектирования, получилось выполнить 1 и 2 задание практической работы.

Практическая работа № 23 Абстрактные типы данных. Очередь

Цель работы

Научиться разрабатывать программы с абстрактными типами данных на языке Джава.

Теоретические сведения. Очередь

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

2. Очередь называется работает по дисциплине обслуживания «первый пришел — первый обслужен» (FCFS, first come first served) 3. Например, люди, стоящие в кассу магазина образуют очередь оплаты покупок. Пример очереди можно увидеть на рис. 23.1. Операция dequeue означает удаление элемента из начала очереди, а операция enqueue добавление элемента в конец очереди.

Использование очередей в разработке программ.

Очереди широко используются

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

Классификация очередей

Очередь — это структура данных, похожая на очередь в реальном мире.

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

Задание 1

Реализовать очередь на массиве

- Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.
- Реализуйте классы, представляющие циклическую очередь с применением массива.
 - Класс `ArrayQueueModule` должен реализовывать один экземпляр очереди с использованием переменных класса.
 - Класс `ArrayQueueADT` должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).
 - Класс `ArrayQueue` должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).
 - Должны быть реализованы следующие функции(процедуры)/методы:

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

- Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.
- Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.
- Напишите тесты реализованным классам.

Решение

Реализации классов: ArrayQueue, ArrayQueueADT, ArrayQueueModule, LinkedListQueue представлены на рисунках 23.1-23.4 соответственно.


```

public class ArrayQueue {
    private Object[] elements = new Object[10];
    private int size = 0;
    private int head = 0;
    private int tail = 0;

    // Инвариант и пред-постусловия аналогичны ArrayQueueModule.

    public void enqueue(Object element) {
        assert element != null;
        ensureCapacity(size + 1);
        elements[tail] = element;
        tail = (tail + 1) % elements.length;
        size++;
    }

    public Object element() {
        assert size > 0;
        return elements[head];
    }

    public Object dequeue() {
        assert size > 0;
        Object element = elements[head];
        elements[head] = null;
        head = (head + 1) % elements.length;
        size--;
        return element;
    }

    public int size() { return size; }

    public boolean isEmpty() { return size == 0; }

    public void clear() {
        Arrays.fill(elements, val: null);
        size = 0;
        head = 0;
        tail = 0;
    }

    private void ensureCapacity(int capacity) {
        if (capacity > elements.length) {
            int newCapacity = Math.max(2 * elements.length, capacity);
            elements = Arrays.copyOf(elements, newCapacity);
        }
    }
}

```

Рисунок 23.1 – Код описывающий класс ArrayQueue

```

public class ArrayQueueADT {
    private Object[] elements = new Object[10];
    private int size = 0;
    private int head = 0;
    private int tail = 0;

    // Инвариант и пред-постусловия аналогичны ArrayQueueModule.

    public static void enqueue(ArrayQueueADT queue, Object element) {
        assert element != null;
        ensureCapacity(queue, capacity: queue.size + 1);
        queue.elements[queue.tail] = element;
        queue.tail = (queue.tail + 1) % queue.elements.length;
        queue.size++;
    }

    public static Object element(ArrayQueueADT queue) {
        assert queue.size > 0;
        return queue.elements[queue.head];
    }

    public static Object dequeue(ArrayQueueADT queue) {
        assert queue.size > 0;
        Object element = queue.elements[queue.head];
        queue.elements[queue.head] = null;
        queue.head = (queue.head + 1) % queue.elements.length;
        queue.size--;
        return element;
    }

    public static int size(ArrayQueueADT queue) { return queue.size; }

    public static boolean isEmpty(ArrayQueueADT queue) { return queue.size == 0; }

    public static void clear(ArrayQueueADT queue) {
        Arrays.fill(queue.elements, null);
        queue.size = 0;
        queue.head = 0;
        queue.tail = 0;
    }

    private static void ensureCapacity(ArrayQueueADT queue, int capacity) {
        if (capacity > queue.elements.length) {
            int newCapacity = Math.max(2 * queue.elements.length, capacity);
            queue.elements = Arrays.copyOf(queue.elements, newCapacity);
        }
    }
}

```

Рисунок 23.2 – Код описывающий класс ArrayQueueADT

```

public class ArrayQueueModule {
    private static int size;
    private static Object[] elements = new Object[10];
    private static int head = 0;
    private static int tail = 0;

    // Инвариант:
    // - elements - массив для хранения элементов очереди
    // - size - текущий размер очереди
    // - head - индекс начала очереди (первого элемента)
    // - tail - индекс конца очереди (следующего после последнего элемента)
    // - Если size > 0, то элементы очереди находятся в массиве elements от head до tail-1.
    // - Если size == 0, очередь пуста и head == tail.

    public static void enqueue(Object element) {
        assert element != null;
        ensureCapacity(size + 1);
        elements[tail] = element;
        tail = (tail + 1) % elements.length;
        size++;
    }

    public static Object element() {
        assert size > 0;
        return elements[head];
    }

    public static Object dequeue() {
        assert size > 0;
        Object element = elements[head];
        elements[head] = null;
        head = (head + 1) % elements.length;
        size--;
        return element;
    }

    public static int size() { return size; }

    public static boolean isEmpty() { return size == 0; }

    public static void clear() {
        Arrays.fill(elements, null);
        size = 0;
        head = 0;
        tail = 0;
    }

    private static void ensureCapacity(int capacity) {
        if (capacity > elements.length) {
            int newCapacity = Math.max(2 * elements.length, capacity);
            elements = Arrays.copyOf(elements, newCapacity);
        }
    }
}

```

Рисунок 23.3 – Код описывающий класс ArrayQueueModule

```

public class LinkedQueue {
    private Node front;
    private Node rear;
    private int size;

    private class Node {
        Object data;
        Node next;

        Node(Object data) {
            this.data = data;
            this.next = null;
        }
    }

    public void enqueue(Object element) {
        Node newNode = new Node(element);
        if (isEmpty()) {
            front = newNode;
            rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
        size++;
    }

    public Object dequeue() {
        if (isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        Object removed = front.data;
        front = front.next;
        size--;
        if (isEmpty()) {
            rear = null;
        }
        return removed;
    }

    public Object peek() {
        if (isEmpty()) {
            throw new NoSuchElementException("Queue is empty");
        }
        return front.data;
    }

    public int size() { return size; }

    public boolean isEmpty() { return size == 0; }

    public void clear() {
        front = null;
        rear = null;
        size = 0;
    }
}

```

Рисунок 23.4 – Код описывающий класс LinkedQueue

Выводы по работе

Следуя указаниям из методических указаний мы научились работать с очередью в Java.

Практическая работа № 24

Паттерны проектирования. Порождающие паттерны: абстрактная фабрика, фабричный метод

Цель работы

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

Теоретические сведения. Очередь

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

Некоторые из преимуществ использования шаблонов проектирования:

1. 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.
2. 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.
3. Поскольку шаблоны проектирования заранее определены то это упрощает понимание и отладку нашего кода. Это приводит к более быстрому развитию проектов, так как новые члены команды понимают код.

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

Шаблон проектирования Фабрика (factory), его также называют фабричный метод используется, когда у нас есть суперкласс с несколькими подклассами, и на основе ввода нам нужно вернуть один из подклассов. Этот шаблон снимает с себя ответственность за создание экземпляра класса из клиентской программы в класс фабрики. Давайте сначала узнаем, как реализовать фабричный шаблон проектирования в java, а затем мы рассмотрим преимущества фабричного шаблона. Мы увидим некоторые примеры использования фабричного шаблона проектирования в JDK. Обратите внимание, что этот шаблон также известен как шаблон проектирования фабричный метод.

Суперкласс в шаблоне проектирования Фабрика может быть интерфейсом, абстрактным классом или обычным классом Java.

Выполнение лабораторной работы

Задание

Задание 1

Разработать программную реализацию по UML диаграмме, представленной на рис.24.1 с использованием изучаемых паттернов.

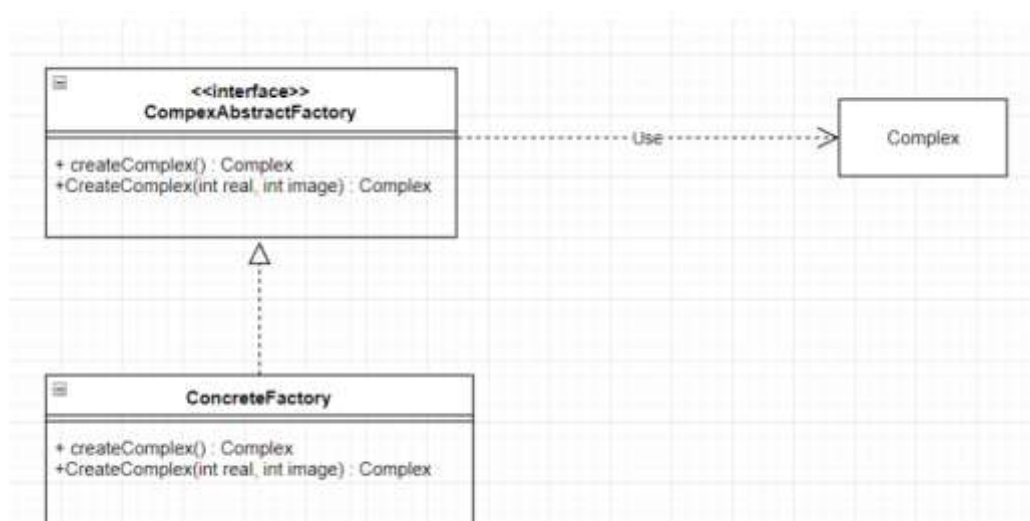


Рисунок 24.1 – UML схема ComplexNumber

Решение

Задание 1

По UML схеме были созданы два класса и один интерфейс с зависимостями, полями и методами соответственно схеме, эти классы представлены на рисунках 24.2-24.4.

```
public class Complex {  
    private int real;  
    private int imaginary;  
  
    public Complex() {  
        this.real = 0;  
        this.imaginary = 0;  
    }  
  
    public Complex(int real, int imaginary) {  
        this.real = real;  
        this.imaginary = imaginary;  
    }  
  
    // Геттеры и сеттеры для действительной и мнимой части  
  
    @Override  
    public String toString() { return real + " + " + imaginary + "i"; }  
}
```

Рисунок 24.2 – Код класса Complex

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

Рисунок 24.3 – Код класса ConcreteFactory, описывающий ComplexAbstractFactory


```
public interface ComplexAbstractFactory {  
    1 implementation  
    Complex createComplex();  
  
    1 implementation  
    Complex createComplex(int real, int imaginary);  
}
```

Рисунок 24.4 – Код описывающий интерфейс ComplexAbstractFactory

Выводы по работе

Следуя указаниям из методических указаний и строго следуя шаблону проектирования, получилось выполнить задание практической работы. Были усвоены порождающие паттерны при разработке программ на Java. А именно в этой работе использовались: абстрактная фабрика и фабричный метод.