Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

О Т Ч Е Т

по лабораторной работе №3

Название: Классы: наследование, полиморфизм

Дисциплина: Языки программирования для работы с большими данными

Студент	ИУ6-22М		М.А. Зотов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			П.В. Степанов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2023 г.

Цель: получить опыт работы с классами на языке Java.

Вариант 1 – задание 9

Условие: определить класс Квадратное уравнение. Класс должен содержать несколько конструкторов. Реализовать методы для поиска корней, экстремумов, а также интервалов убывания/возрастания. Создать массив объектов и определить наибольшие и наименьшие по значению корни.

Код:

```
public class QuadraticEquation {
    public final double a, b, c;

    public QuadraticEquation(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public QuadraticEquation(double a, double b) {
        this.a = a;
        this.b = b;
        this.c = 0;
    }

    public QuadraticEquation(double a) {
        this.a = a;
        this.b = 0;
        this.c = 0;
    }

    public String toString() {
        return "(" + Double.toString(this.a) + ")x^2 + (" +
Double.toString(this.b) + ")x + (" +
                Double.toString(this.c) + ")";
    }

    private double getDiscriminant() {
        return b * b - 4 * a * c;
    }

    public double[] getRoots() {
        double discriminant = getDiscriminant();
        if (discriminant < 0) {
            return new double[0];
        } else if (discriminant == 0) {
            return new double[]{-b / (2 * a)};
        } else {
            double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
            double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
            return new double[]{root1, root2};
        }
    }

    public double getExtrema() {
        return -b / (2 * a);
    }

    public double[] getIncreasingInterval() {
```

```
        double extrema = getExtrema();
        if (a > 0) {
            return new double[]{extrema, Double.POSITIVE_INFINITY};
        } else {
            return new double[]{Double.NEGATIVE_INFINITY, extrema};
        }
    }

    public double[] getDecreasingInterval() {
        double extrema = getExtrema();
        if (a > 0) {
            return new double[]{Double.NEGATIVE_INFINITY, extrema};
        } else {
            return new double[]{extrema, Double.POSITIVE_INFINITY};
        }
    }

    public double getLargestRoot() {
        double[] roots = getRoots();
        if (roots.length == 0)
            return Double.NaN;
        else if (roots.length == 1)
            return roots[0];
        else
            return Math.max(roots[0], roots[1]);
    }

    // Method to get the smallest root of the quadratic equation
    public double getSmallestRoot() {
        double[] roots = getRoots();
        if (roots.length == 0)
            return Double.NaN;
        else if (roots.length == 1)
            return roots[0];
        else
            return Math.min(roots[0], roots[1]);
    }
}

public class var1_ex9 {
    public static void main(String[] args) {
        QuadraticEquation[] equations = new QuadraticEquation[]{
                new QuadraticEquation(1, 0, -1),
                new QuadraticEquation(-2),
                new QuadraticEquation(1, -3),
                new QuadraticEquation(3, -6, 3)
        };

        for (QuadraticEquation equation : equations) {
            System.out.println("Equation " + equation.toString() + ":");
            System.out.println("Smallest root of is: " +
Double.toString(equation.getSmallestRoot()));
            System.out.println("Largest root of equation is: " +
Double.toString(equation.getLargestRoot()));
            System.out.println();
        }
    }
}
```

Результат выполнения:

```
Equation (1.0)x^2 + (0.0)x + (-1.0):
Smallest root of is: -1.0
Largest root of equation is: 1.0

Equation (-2.0)x^2 + (0.0)x + (0.0):
Smallest root of is: 0.0
Largest root of equation is: 0.0

Equation (1.0)x^2 + (-3.0)x + (0.0):
Smallest root of is: 0.0
Largest root of equation is: 3.0

Equation (3.0)x^2 + (-6.0)x + (3.0):
Smallest root of is: 1.0
Largest root of equation is: 1.0
```

Рисунок 1 – Результат выполнения 1.9

Вариант 1 – задание 10

Условие: определить класс Булева матрица (BoolMatrix) размерности (n x m). Класс должен содержать несколько конструкторов. Реализовать методы для логического сложения (дизъюнкции), умножения и инверсии матриц. Реализовать методы для подсчета числа единиц в матрице и упорядочения строк в лексикографическом порядке.

Код:

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

public class BoolMatrix {
    public final boolean[][] matrix;
    private final int rows, cols;

    public BoolMatrix(boolean[][] matrix) {
        this.matrix = matrix;
        this.rows = matrix.length;
        this.cols = matrix[0].length;
    }

    public BoolMatrix(int rows, int cols) {
        this.matrix = new boolean[rows][cols];
        this.rows = rows;
        this.cols = cols;
    }

    public BoolMatrix disjunction(BoolMatrix other) {
        if (this.rows != other.rows || this.cols != other.cols) {
            throw new IllegalArgumentException("Matrices must have same
dimensions");
        }
        BoolMatrix result = new BoolMatrix(this.rows, this.cols);
        for (int i = 0; i < this.rows; i++) {
```

```
            for (int j = 0; j < this.cols; j++) {
                result.matrix[i][j] = this.matrix[i][j] ||
other.matrix[i][j];
            }
        }
        return result;
    }

    public BoolMatrix multiplication(BoolMatrix other) {
        if (this.cols != other.rows) {
            throw new IllegalArgumentException("Matrices dimensions do not
allow multiplication");
        }
        BoolMatrix result = new BoolMatrix(this.rows, other.cols);
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++) {
                result.matrix[i][j] = this.matrix[i][j] &&
other.matrix[i][j];
            }
        }
        return result;
    }

    public BoolMatrix inversion() {
        BoolMatrix result = new BoolMatrix(this.cols, this.rows);
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++) {
                result.matrix[i][j] = !this.matrix[i][j];
            }
        }
        return result;
    }

    public int countOnes() {
        int count = 0;
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++) {
                if (this.matrix[i][j]) {
                    count++;
                }
            }
        }
        return count;
    }

    public void orderRows() {
        Arrays.sort(matrix, new Comparator<boolean[]>() {
            @Override
            public int compare(boolean[] row1, boolean[] row2) {
                for (int i = 0; i < row1.length; i++) {
                    if (row1[i] != row2[i]) {
                        return row1[i] ? 1 : -1;
                    }
                }
                return 0;
            }
        });
    }

    public void print() {
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++)
                System.out.print(Boolean.toString(this.matrix[i][j]) + "\t");
```

```
            System.out.println();
        }
    }
}

public class var1_ex10 {
    public static void main(String[] args) {
        boolean[][] tmp = new boolean[2][2];
        tmp[0][0] = true;
        tmp[0][1] = false;
        tmp[1][0] = false;
        tmp[1][1] = false;
        BoolMatrix m1 = new BoolMatrix(tmp);

        System.out.println("First matrix:");
        m1.print();

        tmp = new boolean[2][2];
        tmp[0][0] = true;
        tmp[0][1] = true;
        tmp[1][0] = true;
        tmp[1][1] = false;
        BoolMatrix m2 = new BoolMatrix(tmp);

        System.out.println("\nSecond matrix:");
        m2.print();

        BoolMatrix res_disjunction = m1.disjunction(m2);
        System.out.println("\nResult of disjunction:");
        res_disjunction.print();

        BoolMatrix res_multiplication = m1.multiplication(m2);
        System.out.println("\nResult of multiplication:");
        res_multiplication.print();

        BoolMatrix res_inversion = m1.inversion();
        System.out.println("\nResult of inversion first matrix:");
        res_inversion.print();

        System.out.println("\nAmount of ones in second matrix = " +
Integer.toString(m2.countOnes()));

        m1.orderRows();
        System.out.println("\nResult of sorting first matrix:");
        m1.print();
    }
}
```

Результат выполнения:

```
First matrix:
true    false
false   false

Second matrix:
true    true
true    false

Result of disjunction:
true    true
true    false

Result of multiplication:
true    false
false   false

Result of inversion first matrix:
false   true
true    true

Amount of ones in second matrix = 3

Result of sorting first matrix:
false   false
true    false
```

Рисунок 2 – Результат выполнения 1.10

Вариант 2 – задание 9

Условие: Product: id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество. Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль. Создать массив объектов. Вывести: a) список товаров для заданного наименования; b) список товаров для заданного наименования, цена которых не превосходит заданную; c) список товаров, срок хранения которых больше заданного.

Код:

```
public class Product {
    private int id;
    private String name;
    private String upc;
    private String manufacturer;
    private double price;
    private int shelfLife;
    private int quantity;

    public Product(int id, String name, String upc, String manufacturer,
double price, int shelfLife, int quantity) {
        this.id = id;
        this.name = name;
        this.upc = upc;
```

```
        this.manufacturer = manufacturer;
        this.price = price;
        this.shelfLife = shelfLife;
        this.quantity = quantity;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    public String getUpc() {
        return upc;
    }

    public void setUpc(String upc) {
        this.upc = upc;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    public double getPrice() {
        return price;
    }

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

    public int getShelfLife() {
        return shelfLife;
    }

    public void setShelfLife(int shelfLife) {
        this.shelfLife = shelfLife;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
```

```
    public String toString() {
        return "Product{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", upc='" + upc + '\'' +
                ", manufacturer='" + manufacturer + '\'' +
                ", price=" + price +
                ", shelfLife=" + shelfLife +
                ", quantity=" + quantity +
                '}';
    }
}

import java.util.Scanner;

public class var2_ex9 {
    private static void getProductsByName(Product[] products, String name) {
        System.out.println("\nList of products for " + name + ":");
        for (Product product : products) {
            if (product.getName().equals(name)) {
                System.out.println(product);
            }
        }
    }
    private static void getProductsByNameAndPrice(Product[] products, String
name, double price) {
        System.out.println("\nList of products for " + name + " with price <=
" + Double.toString(price) + ":");
        for (Product product : products) {
            if (product.getName().equals(name) && product.getPrice() <=
price) {
                System.out.println(product);
            }
        }
    }
    private static void getProductsByShelfLife(Product[] products, int
shelfLife) {
        System.out.println("\nList of products for with shelf life <= " +
Integer.toString(shelfLife) + ":");
        for (Product product : products) {
            if (product.getShelfLife() <= shelfLife) {
                System.out.println(product);
            }
        }
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Product[] products = {
                new Product(1, "Product 1", "123456789", "Manufacturer 1",
10.99, 30, 50),
                new Product(2, "Product 2", "987654321", "Manufacturer 2",
5.99, 60, 100),
                new Product(3, "Product 3", "02468", "Manufacturer 3", 20.99,
90, 20),
                new Product(2, "Product 2", "13579", "Manufacturer 4", 6.99,
60, 2000)
        };
        System.out.println("Enter product's name:");
        String name = in.nextLine();
        getProductsByName(products, name);

        System.out.print("\nEnter product's price: ");
        double price = in.nextDouble();
```

```
        getProductsByNameAndPrice(products, name, price);

        System.out.print("\nEnter product's self life: ");
        int shelfLife = in.nextInt();
        getProductsByShelfLife(products, shelfLife);
    }
}
```

Результат выполнения:

```
Enter product's name:
Product 2

List of products for Product 2:
Product{id=2, name='Product 2', upc='987654321', manufacturer='Manufacturer 2', price=5.99, shelfLife=60, quantity=100}
Product{id=2, name='Product 2', upc='13579', manufacturer='Manufacturer 4', price=6.99, shelfLife=60, quantity=2000}

Enter product's price: 6

List of products for Product 2 with price <= 6.0:
Product{id=2, name='Product 2', upc='987654321', manufacturer='Manufacturer 2', price=5.99, shelfLife=60, quantity=100}

Enter product's self life: 60

List of products for with shelf life <= 60:
Product{id=1, name='Product 1', upc='123456789', manufacturer='Manufacturer 1', price=10.99, shelfLife=30, quantity=50}
Product{id=2, name='Product 2', upc='987654321', manufacturer='Manufacturer 2', price=5.99, shelfLife=60, quantity=100}
Product{id=2, name='Product 2', upc='13579', manufacturer='Manufacturer 4', price=6.99, shelfLife=60, quantity=2000}
```

Рисунок 3 – Результат выполнения 2.9

Вариант 2 – задание 10

Условие: Train: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс). Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль. Создать массив объектов. Вывести: a) список поездов, следующих до заданного пункта назначения; b) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа; c) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

Код:

```
import java.sql.Time;

public class Train {
    private String destination;
    private int trainNumber;
    private Time departureTime;
    private int generalSeats;
    private int compartmentSeats;
    private int reservedSeats;
    private int suiteSeats;

    public Train(String destination, int trainNumber, Time departureTime,
                 int generalSeats, int compartmentSeats, int reservedSeats,
int suiteSeats) {
        this.destination = destination;
        this.trainNumber = trainNumber;
        this.departureTime = departureTime;
```

```
        this.generalSeats = generalSeats;
        this.compartmentSeats = compartmentSeats;
        this.reservedSeats = reservedSeats;
        this.suiteSeats = suiteSeats;
    }

    public String getDestination() {
        return destination;
    }

    public void setDestination(String destination) {
        this.destination = destination;
    }

    public int getTrainNumber() {
        return trainNumber;
    }

    public void setTrainNumber(int trainNumber) {
        this.trainNumber = trainNumber;
    }

    public Time getDepartureTime() {
        return departureTime;
    }

    public void setDepartureTime(Time departureTime) {
        this.departureTime = departureTime;
    }

    public int getGeneralSeats() {
        return generalSeats;
    }

    public void setGeneralSeats(int generalSeats) {
        this.generalSeats = generalSeats;
    }

    public int getCompartmentSeats() {
        return compartmentSeats;
    }

    public void setCompartmentSeats(int compartmentSeats) {
        this.compartmentSeats = compartmentSeats;
    }

    public int getReservedSeats() {
        return reservedSeats;
    }

    public void setReservedSeats(int reservedSeats) {
        this.reservedSeats = reservedSeats;
    }

    public int getSuiteSeats() {
        return suiteSeats;
    }

    public void setSuiteSeats(int suiteSeats) {
        this.suiteSeats = suiteSeats;
    }

    public String toString() {
```

```
        return "Train{" +
                "destination='" + destination + '\'' +
                ", trainNumber=" + trainNumber +
                ", departureTime='" + departureTime + '\'' +
                ", generalSeats=" + generalSeats +
                ", compartmentSeats=" + compartmentSeats +
                ", reservedSeats=" + reservedSeats +
                ", suiteSeats=" + suiteSeats +
                '}';
    }
}

import java.sql.Time;
import java.util.Scanner;

public class var2_ex10 {
    private static void getTrainsByDestination(Train[] trains, String
destination) {
        System.out.println("\nList of trains with destination '" +
destination + "':");
        for (Train train : trains) {
            if (train.getDestination().equals(destination)) {
                System.out.println(train);
            }
        }
    }
    private static void getTrainsByDestinationAndDepTime(Train[] trains,
String destination, Time depTime) {
        System.out.println("\nList of trains with destination '" +
destination +
                "' and departure time > " + depTime + ":");
        for (Train train : trains) {
            if (train.getDestination().equals(destination) &&
train.getDepartureTime().after(depTime)) {
                System.out.println(train);
            }
        }
    }
    private static void getTrainsWithGenSeatsByDestination(Train[] trains,
String destination) {
        System.out.println("\nList of trains with general seats and
destination '" + destination + "':");
        for (Train train : trains) {
            if (train.getDestination().equals(destination) &&
train.getGeneralSeats() > 0) {
                System.out.println(train);
            }
        }
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Train[] trains = {
                new Train("Paris", 1, new Time(10, 0, 0), 200, 100, 50, 20),
                new Train("Berlin", 2, new Time(12, 30, 0), 150, 80, 40, 10),
                new Train("London", 3, new Time(14, 45, 0), 250, 120, 60,
30),
                new Train("Paris", 4, new Time(15, 0, 0), 0, 60, 30, 10)
        };
        System.out.println("Enter destination:");
        String destination = in.nextLine();
        getTrainsByDestination(trains, destination);

        System.out.print("\nEnter departure hour: ");
```

```
        int hour = in.nextInt();
        System.out.print("Enter departure minute: ");
        int minute = in.nextInt();
        getTrainsByDestinationAndDepTime(trains, destination, new Time(hour,
minute, 0));

        getTrainsWithGenSeatsByDestination(trains, destination);
    }
}
```

Результат выполнения:

```
Enter destination:
Paris

List of trains with destination 'Paris':
Train{destination='Paris', trainNumber=1, departureTime='10:00:00', generalSeats=200, compartmentSeats=100, reservedSeats=50, suiteSeats=20}
Train{destination='Paris', trainNumber=4, departureTime='15:00:00', generalSeats=0, compartmentSeats=60, reservedSeats=30, suiteSeats=10}

Enter departure hour: 9
Enter departure minute: 30

List of trains with destination 'Paris' and departure time > 09:30:00:
Train{destination='Paris', trainNumber=1, departureTime='10:00:00', generalSeats=200, compartmentSeats=100, reservedSeats=50, suiteSeats=20}
Train{destination='Paris', trainNumber=4, departureTime='15:00:00', generalSeats=0, compartmentSeats=60, reservedSeats=30, suiteSeats=10}

List of trains with general seats and destination 'Paris':
Train{destination='Paris', trainNumber=1, departureTime='10:00:00', generalSeats=200, compartmentSeats=100, reservedSeats=50, suiteSeats=20}
```

Рисунок 4 – Результат выполнения 2.10

Вариант 3 – задание 9

Условие: создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString(). Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString().

Код:

```
import java.util.Objects;

public class Photo {
    private String name;

    public Photo(String name) {
        this.name = name;
    }

    public Photo() {
    }

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

    public String getName() {
        return name;
    }

    @Override
    public boolean equals(Object o) {
```

```
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Photo photo = (Photo) o;
        return Objects.equals(name, photo.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }

    @Override
    public String toString() {
        return "Photo{" +
                "name='" + name + '\'' +
                '}';
    }
}
import java.util.ArrayList;
import java.util.Objects;

public class PhotoAlbum {
    private String name;
    private final ArrayList<Photo> photos;

    public PhotoAlbum(String name) {
        this.name = name;
        this.photos = new ArrayList<>();
    }

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

    public String getName() {
        return name;
    }

    public void addPhoto(Photo photo) {
        photos.add(photo);
    }

    public int getNumberOfPhotos() {
        return photos.size();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        PhotoAlbum that = (PhotoAlbum) o;
        return Objects.equals(name, that.name) &&
                Objects.equals(photos, that.photos);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, photos);
    }

    @Override
    public String toString() {
        return "PhotoAlbum{" +
```

```
                "name='" + name + '\'' +
                ", photos=" + photos +
                '}';
    }
}
public class var3_ex9 {
    public static void main(String[] args) {
        Photo photo1 = new Photo("Beach");
        Photo photo2 = new Photo("Mountains");
        Photo photo3 = new Photo();

        photo3.setName("Mountains");

        System.out.println("Photos 2 and 3 are equal: " +
photo2.equals(photo3));

        System.out.println("Hash code for photo 1: " + photo1.hashCode());

        System.out.println("Photo 2 = " + photo2);

        PhotoAlbum album1 = new PhotoAlbum("Vacation");
        album1.addPhoto(photo1);
        album1.addPhoto(photo2);

        PhotoAlbum album2 = new PhotoAlbum("Office");
        album2.addPhoto(photo3);

        System.out.println("Albums are equal: " + album1.equals(album2));

        System.out.println("Hash code for album 2: " + album2.hashCode());

        System.out.println("Album 1 = " + album1);

        System.out.println("Number of photos in album 1: " +
album1.getNumberOfPhotos());
    }
}
```

Результат выполнения:

```
Photos 2 and 3 are equal: true
Hash code for photo 1: 64057698
Photo 2 = Photo{name='Mountains'}
Albums are equal: false
Hash code for album 2: -1717597175
Album 1 = PhotoAlbum{name='Vacation', photos=[Photo{name='Beach'}, Photo{name='Mountains'}]}
Number of photos in album 1: 2
```

Рисунок 5 – Результат выполнения 3.9

Вариант 3 – задание 10

Условие: создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString(). Создать объект класса Год, используя классы Месяц, День. Методы: задать дату, вывести на консоль день недели по заданной дате, рассчитать количество дней, месяцев в заданном временном промежутке.

Код:

```
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;

public class Year {
    private int year;
    private Month month;
    private Day day;

    public Year(int year, int month, int day) {
        try {
            LocalDate date = LocalDate.of(year, month, day);
        } catch (DateTimeException e) {
            throw new IllegalArgumentException("Invalid date value");
        }
        this.year = year;
        this.month = new Month(month);
        this.day = new Day(day);
    }

    public Year(int year) {
        this.year = year;
        this.month = new Month(1);
        this.day = new Day(1);
    }

    public void setYear(int year) {
        this.year = year;
    }

    public int getYear() {
        return year;
    }

    public void setDate(int month, int day) {
        try {
            LocalDate date = LocalDate.of(year, month, day);
        } catch (DateTimeException e) {
            throw new IllegalArgumentException("Invalid date value");
        }
        this.month = new Month(month);
        this.day = new Day(day);
    }

    public DayOfWeek getDayOfWeek() {
        LocalDate date = LocalDate.of(year, month.getMonth(), day.getDay());
        return date.getDayOfWeek();
    }

    public int calculateDaysBetween(Year otherYear) {
        LocalDate date1 = LocalDate.of(year, this.month.getMonth(),
this.day.getDay());
        LocalDate date2 = LocalDate.of(otherYear.getYear(),
otherYear.month.getMonth(), otherYear.day.getDay());
        return (int) ChronoUnit.DAYS.between(date1, date2);
    }

    public int calculateMonthsBetween(Year otherYear) {
        YearMonth yearMonth1 = YearMonth.of(year, this.month.getMonth());
        YearMonth yearMonth2 = YearMonth.of(otherYear.getYear(),
otherYear.month.getMonth());
```

```
        return (int) ChronoUnit.MONTHS.between(yearMonth1, yearMonth2);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Year year1)) return false;
        return year == year1.year && month.equals(year1.month) &&
day.equals(year1.day);
    }

    @Override
    public int hashCode() {
        int result = year;
        result = 31 * result + month.hashCode();
        result = 31 * result + day.hashCode();
        return result;
    }

    @Override
    public String toString() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd");
        LocalDate date = LocalDate.of(year, month.getMonth(), day.getDay());
        return "Year{" +
                "date=" + formatter.format(date) +
                '}';
    }
}
class Month {
    private final int month;

    public Month(int month) {
        if (month < 1 || month > 12) {
            throw new IllegalArgumentException("Invalid month value");
        }
        this.month = month;
    }

    public int getMonth() {
        return month;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Month month1)) return false;
        return month == month1.month;
    }

    @Override
    public int hashCode() {
        return month;
    }

    @Override
    public String toString() {
        return "Month{" +
                "month=" + month +
                '}';
    }
}
class Day {
```

```
    private final int day;

    public Day(int day) {
        if (day < 1 || day > 31) {
            throw new IllegalArgumentException("Invalid day value");
        }
        this.day = day;
    }

    public int getDay() {
        return day;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Day day1)) return false;
        return day == day1.day;
    }

    @Override
    public int hashCode() {
        return day;
    }

    @Override
    public String toString() {
        return "Day{" +
                "day=" + day +
                '}';
    }
}
public class var3_ex10 {
    public static void main(String[] args) {
        Year year1 = new Year(2023, 3, 15);
        Year year2 = new Year(2023, 2, 2);
        Year year3 = new Year(2023);

        year3.setDate(2, 2);

        System.out.println("Years 1 and 2 are equal: " +
year1.equals(year2));
        System.out.println("Years 2 and 3 are equal: " +
year2.equals(year3));

        System.out.println("Hash code for year 1: " + year1.hashCode());

        System.out.println("Year 2 = " + year2);

        System.out.println("Day of week for year 1: " +
year1.getDayOfWeek());

        System.out.println("Months between year 2 and year 1: " +
year2.calculateMonthsBetween(year1));
        System.out.println("Days between year 2 and year 1: " +
year2.calculateDaysBetween(year1));
    }
}
```

Результат выполнения:

```
Years 1 and 2 are equal: false
Years 2 and 3 are equal: true
Hash code for year 1: 1944211
Year 2 = Year{date=2023-02-02}
Day of week for year 1: WEDNESDAY
Months between year 2 and year 1: 1
Days between year 2 and year 1: 41
```

Рисунок 6 – Результат выполнения 3.10

Вариант 4 – задание 9

Условие: создать объект класса Год, используя классы Месяц, День. Методы: задать дату, вывести на консоль день недели по заданной дате, рассчитать количество дней, месяцев в заданном временном промежутке.

Код:

```
public class OnlineStoreProduct {
    private final String name;
    private final double price;

    public OnlineStoreProduct(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return name + ": $" + price;
    }
}
public class Client {
    private final String name;
    private final String email;
    private final String address;
    private double balance;

    public Client(String name, String email, String address, double balance)
{
        this.name = name;
        this.email = email;
        this.address = address;
        this.balance = balance;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
```

```
    }

    public String getAddress() {
        return address;
    }

    public double getBalance() {
        return balance;
    }

    public void makeTransaction(double amount) {
        balance += amount;
    }

    @Override
    public String toString() {
        return name + " (Email: " + email + ", Address: " + address + ",
Balance: $" + balance + ")";
    }
}
import java.time.LocalDateTime;

public class Order {
    private final int orderNumber;
    private final LocalDateTime orderDate;
    private final Client client;
    private final OnlineStoreProduct[] products;
    private final double total;

    public Order(int orderNumber, Client client, OnlineStoreProduct[]
products) {
        this.orderNumber = orderNumber;
        this.orderDate = LocalDateTime.now();
        this.client = client;
        this.products = products;
        this.total = calculateTotal();
    }

    public int getOrderNumber() {
        return orderNumber;
    }

    public LocalDateTime getOrderDate() {
        return orderDate;
    }

    public Client getClient() {
        return client;
    }

    public OnlineStoreProduct[] getProducts() {
        return products;
    }

    public double getTotal() {
        return total;
    }

    private double calculateTotal() {
        double sum = 0;
        for (OnlineStoreProduct product : products) {
            sum += product.getPrice();
        }
```

```
        return sum;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Order Number: ").append(orderNumber).append("\n");
        sb.append("Order Date: ").append(orderDate).append("\n");
        sb.append("Client: ").append(client).append("\n");
        sb.append("Products: \n");
        for (OnlineStoreProduct product : products) {
            sb.append("\t").append(product).append("\n");
        }
        sb.append("Total: $").append(total).append("\n");
        return sb.toString();
    }
}
import java.util.ArrayList;
import java.util.List;

public class Administrator {
    private List<OnlineStoreProduct> products = null;
    private final List<Client> blacklist;

    public Administrator() {
        products = new ArrayList<>();
        blacklist = new ArrayList<>();
    }

    public void addProduct(OnlineStoreProduct product) {
        products.add(product);
    }

    public List<OnlineStoreProduct> getProducts() {
        return products;
    }

    public void registerSale(Order order) {
        if (blacklist.contains(order.getClient()))
            System.out.println("This client is blacklisted and cannot make
purchases.");
        else if (order.getClient().getBalance() < order.getTotal())
            System.out.println("This client cannot make purchases " +
                    "because he does not have enough money on his balance.");
        else
            order.getClient().makeTransaction(-order.getTotal());
    }

    public void blacklistClient(Client client) {
        blacklist.add(client);
    }

    public List<Client> getBlacklist() {
        return blacklist;
    }

    public String getProductsList() {
        StringBuilder sb = new StringBuilder();
        for (OnlineStoreProduct product : products) {
            sb.append(product).append("\n");
        }
        return sb.toString();
```

```
    }
}
public class var4_ex9 {
    public static void main(String[] args) {
        OnlineStoreProduct product1 = new OnlineStoreProduct("Apple", 2);
        OnlineStoreProduct product2 = new OnlineStoreProduct("Onion", 1);

        Administrator administrator = new Administrator();
        administrator.addProduct(product1);
        administrator.addProduct(product2);
        System.out.println("Products list:\n" +
administrator.getProductsList());

        Client client1 = new Client("Ivan", "ivan@mail.ru", "Moscow", 20);
        OnlineStoreProduct[] products1 = {
                administrator.getProducts().get(0),
                administrator.getProducts().get(1)
        };
        Order order1 = new Order(1, client1, products1);
        System.out.println(order1);

        administrator.registerSale(order1);

        System.out.println("Ivan's balance after registering sale: " +
client1.getBalance());

        Client client2 = new Client("Peter", "peter@mail.ru", "SPB", 0);
        administrator.blacklistClient(client2);

        System.out.println("\nBlacklist:\n" + administrator.getBlacklist());

        OnlineStoreProduct[] products2 = {
                administrator.getProducts().get(0)
        };
        Order order2 = new Order(2, client2, products2);
        System.out.println("\n" + order2);

        System.out.println("Trying to register sale with blacklisted
client...");
        administrator.registerSale(order2);

        Client client3 = new Client("Anna", "anna@mail.ru", "Kazan", 4);

        OnlineStoreProduct[] products3 = {
                administrator.getProducts().get(0),
                administrator.getProducts().get(0),
                administrator.getProducts().get(1)
        };
        Order order3 = new Order(3, client3, products3);
        System.out.println("\n" + order3);

        System.out.println("Trying to register sale with client having not
enough money for it...");
        administrator.registerSale(order3);
    }
}
```

Результат выполнения:

```
Products list:
Apple: $2.0
Onion: $1.0

Order Number: 1
Order Date: 2023-03-17T16:44:29.096390
Client: Ivan (Email: ivan@mail.ru, Address: Moscow, Balance: $20.0)
Products:
    Apple: $2.0
    Onion: $1.0
Total: $3.0

Ivan's balance after registering sale: 17.0

Blacklist:
[Peter (Email: peter@mail.ru, Address: SPB, Balance: $0.0)]

Order Number: 2
Order Date: 2023-03-17T16:44:29.110453
Client: Peter (Email: peter@mail.ru, Address: SPB, Balance: $0.0)
Products:
    Apple: $2.0
Total: $2.0

Trying to register sale with blacklisted client...
This client is blacklisted and cannot make purchases.

Order Number: 3
Order Date: 2023-03-17T16:44:29.110634
Client: Anna (Email: anna@mail.ru, Address: Kazan, Balance: $4.0)
Products:
    Apple: $2.0
    Apple: $2.0
    Onion: $1.0
Total: $5.0
Trying to register sale with client having not enough money for it...
This client cannot make purchases because he does not have enough money on his balance.
```

Рисунок 7 – Результат выполнения 4.9

Вариант 4 – задание 10

Условие: создать объект класса Год, используя классы Месяц, День. Методы: задать дату, вывести на консоль день недели по заданной дате, рассчитать количество дней, месяцев в заданном временном промежутке.

Код:

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Arrays;

public class TrainForOffice {
    private final int trainNumber;
    private final LocalDateTime departureDTTM;
    private final String[] stations;
    private final double price;

    public TrainForOffice(int trainNumber, LocalDateTime departureDTTM,
String[] stations, double price) {
        this.trainNumber = trainNumber;
```

```
        this.departureDTTM = departureDTTM;
        this.stations = stations;
        this.price = price;
    }

    public int getTrainNumber() {
        return trainNumber;
    }

    public LocalDateTime getDepartureDTTM() {
        return departureDTTM;
    }

    public String[] getStations() {
        return stations;
    }

    public double getPrice() {
        return price;
    }

    public String toString() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
hh:mm");
        return "Train{" +
                "trainNumber=" + trainNumber +
                ", departureDate='" + formatter.format(departureDTTM) + '\''
+
                ", stations=" + Arrays.toString(stations) +
                ", price=" + price +
                '}';
    }
}
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Passenger {
    private final String name;
    private final String destinationStation;
    private final LocalDateTime travelDTTM;

    public Passenger(String name, String destinationStation, LocalDateTime
travelDTTM) {
        this.name = name;
        this.destinationStation = destinationStation;
        this.travelDTTM = travelDTTM;
    }

    public String getName() {
        return name;
    }

    public String getDestinationStation() {
        return destinationStation;
    }

    public LocalDateTime getTravelDTTM() {
        return travelDTTM;
    }

    public String toString() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
hh:mm");
```

```
        return "Passenger{" +
                "name=" + name +
                ", destinationStation='" + destinationStation + '\'' +
                ", travelDate='" + formatter.format(travelDTTM) + '\'' +
                '}';
    }
}
public class Ticket {
    private final Passenger passenger;
    private final TrainForOffice train;

    public Ticket(Passenger passenger, TrainForOffice train) {
        this.passenger = passenger;
        this.train = train;
    }

    public Passenger getPassenger() {
        return passenger;
    }

    public TrainForOffice getTrain() {
        return train;
    }

    public String toString() {
        return "Ticket{" +
                "passenger=" + passenger +
                ", train='" + train + '\'' +
                '}';
    }
}
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class RailwayTicketOffice {
    private final Scanner in = new Scanner(System.in);
    private final List<TrainForOffice> trains;
    private final List<Ticket> tickets;

    public RailwayTicketOffice() {
        trains = new ArrayList<>();
        tickets = new ArrayList<>();
    }

    public void addTrain(TrainForOffice train) {
        trains.add(train);
    }

    public void findSuitableTrains(Passenger passenger) {
        List<TrainForOffice> suitableTrains = new ArrayList<>();
        for (TrainForOffice train : trains)
            if (train.getDepartureDTTM().equals(passenger.getTravelDTTM()))
                for (String station : train.getStations())
                    if (station.equals(passenger.getDestinationStation()))
                        suitableTrains.add(train);
        if (suitableTrains.size() > 1) {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < suitableTrains.size(); i++)
                sb.append((i + 1) + " -
").append(suitableTrains.get(i)).append("\n");
            System.out.println("\nPlease, choose train:\n" + sb);
            System.out.print("Your choice: ");
```

```
                generateInvoice(passenger, suitableTrains.get(in.nextInt() - 1));
            } else {
                System.out.println("\nWe are sorry but there are no suitable
trains for you :(");
            }
        }

        private void generateInvoice(Passenger passenger, TrainForOffice train) {
            double price = train.getPrice();
            Ticket ticket = new Ticket(passenger, train);
            tickets.add(ticket);
            System.out.println("\nYou have to pay " + price + " for ticket. Have
a pleasant trip!");
            System.out.println("Your ticket:\n" + ticket);
        }

        public String getTrains() {
            StringBuilder sb = new StringBuilder();
            for (TrainForOffice train : trains) {
                sb.append(train).append("\n");
            }
            return sb.toString();
        }

        public String getTickets() {
            StringBuilder sb = new StringBuilder();
            for (Ticket ticket : tickets) {
                sb.append(ticket).append("\n");
            }
            return sb.toString();
        }
    }
    public class AdministratorOffice {
        private final RailwayTicketOffice railwayTicketOffice;

        public AdministratorOffice(RailwayTicketOffice railwayTicketOffice) {
            this.railwayTicketOffice = railwayTicketOffice;
        }

        public RailwayTicketOffice getRailwayTicketOffice() {
            return railwayTicketOffice;
        }

        public void addTrain(TrainForOffice train) {
            railwayTicketOffice.addTrain(train);
        }
    }
    import java.time.LocalDateTime;
    import java.util.List;
    import java.util.Scanner;

    public class var4_ex10 {
        public static void main(String[] args) {
            String[] stations1 = {
                    "Moscow", "Paris", "Rome"
            };
            TrainForOffice train1 = new TrainForOffice(1,
                    LocalDateTime.of(2022, 4, 1, 17, 30),
                    stations1, 100);
            String[] stations2 = {
                    "Moscow", "Paris", "Rome"
            };
            TrainForOffice train2 = new TrainForOffice(2,
```

```
                LocalDateTime.of(2023, 3, 15, 12, 0),
                stations2, 300);
        String[] stations3 = {
                "Moscow", "Berlin", "Rome"
        };
        TrainForOffice train3 = new TrainForOffice(3,
                LocalDateTime.of(2023, 3, 15, 12, 0),
                stations3, 150);
        String[] stations4 = {
                "Moscow", "Minsk", "Paris"
        };
        TrainForOffice train4 = new TrainForOffice(4,
                LocalDateTime.of(2023, 3, 15, 12, 0),
                stations4, 200);

        RailwayTicketOffice office = new RailwayTicketOffice();
        AdministratorOffice administrator = new AdministratorOffice(office);

        administrator.addTrain(train1);
        administrator.addTrain(train2);
        administrator.addTrain(train3);
        administrator.addTrain(train4);

        System.out.println("Trains:\n" + office.getTrains());

        Passenger passenger1 = new Passenger("Ivan", "Paris",
                LocalDateTime.of(2023, 3, 15, 12, 0));

        System.out.println("Passenger1:\n" + passenger1);

        office.findSuitableTrains(passenger1);

        Passenger passenger2 = new Passenger("Peter", "Istanbul",
                LocalDateTime.of(2023, 3, 15, 12, 0));

        System.out.println("\nPassenger2:\n" + passenger2);

        office.findSuitableTrains(passenger2);

        Passenger passenger3 = new Passenger("Anna", "Rome",
                LocalDateTime.of(2023, 4, 1, 9, 45));

        System.out.println("\nPassenger3:\n" + passenger3);

        office.findSuitableTrains(passenger3);
    }
}
```

Результат выполнения:

```
Trains:
Train{trainNumber=1, departureDate='2022-04-01 05:30', stations=[Moscow, Paris, Rome], price=100.0}
Train{trainNumber=2, departureDate='2023-03-15 12:00', stations=[Moscow, Paris, Rome], price=300.0}
Train{trainNumber=3, departureDate='2023-03-15 12:00', stations=[Moscow, Berlin, Rome], price=150.0}
Train{trainNumber=4, departureDate='2023-03-15 12:00', stations=[Moscow, Minsk, Paris], price=200.0}

Passenger1:
Passenger{name=Ivan, destinationStation='Paris', travelDate='2023-03-15 12:00'}

Please, choose train:
1 - Train{trainNumber=2, departureDate='2023-03-15 12:00', stations=[Moscow, Paris, Rome], price=300.0}
2 - Train{trainNumber=4, departureDate='2023-03-15 12:00', stations=[Moscow, Minsk, Paris], price=200.0}

Your choice: 1

You have to pay 300.0 for ticket. Have a pleasant trip!
Your ticket:
Ticket{passenger=Passenger{name=Ivan, destinationStation='Paris', travelDate='2023-03-15 12:00'}, train='T

Passenger2:
Passenger{name=Peter, destinationStation='Istanbul', travelDate='2023-03-15 12:00'}

We are sorry but there are no suitable trains for you :(

Passenger3:
Passenger{name=Anna, destinationStation='Rome', travelDate='2023-04-01 09:45'}

We are sorry but there are no suitable trains for you :(
```

Рисунок 8 – Результат выполнения 4.10

Вывод: в ходе выполнения лабораторной работы был получен опыт работы с классами на языке Java.