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**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

ОТЧЕТ
по лабораторной работе №3

Название: Классы, наследование и полиморфизм

Дисциплина Языки программирования для работы с большими данными

Студент ИУ6–22М (Группа) _____ (Подпись, дата) М.Э.Хабаров (И.О. Фамилия)

Преподаватель _____ (Подпись, дата) П.В. Степанов (И.О. Фамилия)

2024 г

Цель: освоить принципы классов и наследования на языке Java.

Задание №1

Формулировка задания и код программы представлены в листинге 1:

```
import java.util.Scanner;

/*
    Определить класс Дробь в виде пары (m,n). Класс должен
    содержать несколько конструкторов. Реализовать методы для сложения,
    вычитания, умножения и деления дробей. Объявить массив из k дробей,
    ввести/вывести значения для массива дробей. Создать массив объектов
    и передать его в метод, который изменяет каждый элемент массива
    с четным индексом путем добавления следующего за ним элемента массива
*/

class Fraction {
    private int numerator;
    private int denominator;

    public Fraction() {
        this.numerator = 0;
        this.denominator = 1;
    }

    public Fraction(int numerator, int denominator) {
        this.numerator = numerator;
        if (denominator != 0) {
            this.denominator = denominator;
        } else {
            System.out.println("Знаменатель не может быть равен нулю.
Устанавливаю значение по умолчанию (1).");
            this.denominator = 1;
        }
    }

    public Fraction add(Fraction other) {
        int newNumerator = this.numerator * other.denominator +
other.numerator * this.denominator;
        int newDenominator = this.denominator * other.denominator;
        return new Fraction(newNumerator, newDenominator);
    }

    public Fraction subtract(Fraction other) {
        int newNumerator = this.numerator * other.denominator -
other.numerator * this.denominator;
        int newDenominator = this.denominator * other.denominator;
        return new Fraction(newNumerator, newDenominator);
    }

    public Fraction multiply(Fraction other) {
        int newNumerator = this.numerator * other.numerator;
        int newDenominator = this.denominator * other.denominator;
        return new Fraction(newNumerator, newDenominator);
    }

    public Fraction divide(Fraction other) {
        if (other.numerator == 0) {
            System.out.println("Деление на ноль. Возвращаю дробь (0/1).");
            return new Fraction(0, 1);
        }
```

```
        int newNumerator = this.numerator * other.denominator;
        int newDenominator = this.denominator * other.numerator;
        return new Fraction(newNumerator, newDenominator);
    }

    public String toString() {
        return this.numerator + "/" + this.denominator;
    }
}

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

        Scanner scanner = new Scanner(System.in);

        System.out.print("Введите k: ");
        int k = scanner.nextInt(); // Количество дробей в массиве

        Fraction[] fractions = new Fraction[k];

        // Ввод значений для массива дробей
        System.out.println("Введите дроби:");
        for (int i = 0; i < k; i++) {
            System.out.println("Числитель "+i+"-й "+"дроби: ");
            int m = scanner.nextInt();
            System.out.println("Знаменатель "+i+"-й "+"дроби: ");
            int o = scanner.nextInt();
            fractions[i] = new Fraction(m, o); // Пример значений для дробей
        }

        // Вывод значений массива дробей
        System.out.println("Исходные дроби:");
        for (int i = 0; i < k; i++) {
            System.out.println(fractions[i].toString());
        }

        modifyArray(fractions);

        // Вывод измененных дробей
        System.out.println("\nИзмененные дроби:");
        for (int i = 0; i < k; i++) {
            System.out.println(fractions[i].toString());
        }
    }

    public static void modifyArray(Fraction[] fractions) {
        for (int i = 0; i < fractions.length - 1; i += 2) {
            fractions[i] = fractions[i].add(fractions[i+1]);
        }
    }
}
```

Задание №2

Формулировка задания и код программы представлены в листинге 2:

```
import java.util.Arrays;

/*
    9.  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.
*/

class QuadraticEquation {
    private double a;
    private double b;
    private double 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, b, 0);
    }

    public QuadraticEquation(double a) {
        this(a, 0, 0);
    }

    public double[] findRoots() {
        double discriminant = b * b - 4 * a * c;

        if (discriminant > 0) {
            double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
            double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
            return new double[] {root1, root2};
        } else if (discriminant == 0) {
            double root = -b / (2 * a);
            return new double[] {root};
        } else {
            return new double[0]; // No real roots
        }
    }

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

    public String findIncreasingInterval() {
        return a > 0 ? "(-∞, " + findExtremePoint() + ")" : "(" +
findExtremePoint() + ", +∞)";
    }

    public String findDecreasingInterval() {
        return a > 0 ? "(" + findExtremePoint() + ", +∞)" : "(-∞, " +
findExtremePoint() + ")";
    }
}

public class Main {
    public static void main(String[] args) {
        int k = 3; // Количество квадратных уравнений в массиве

        QuadraticEquation[] equations = new QuadraticEquation[k];

        // Ввод значений для массива квадратных уравнений
        equations[0] = new QuadraticEquation(1, -3, 2); // x^2 - 3x + 2
        equations[1] = new QuadraticEquation(1, -2);     // x^2 - 2x
        equations[2] = new QuadraticEquation(1);         // x^2
```

```
        // Вывод корней для каждого уравнения
        for (int i = 0; i < k; i++) {
            double[] roots = equations[i].findRoots();
            System.out.println("Корни уравнения " + (i+1) + ": " +
Arrays.toString(roots));
        }

        // Нахождение наибольшего и наименьшего корней
        double maxRoot = Double.MIN_VALUE;
        double minRoot = Double.MAX_VALUE;

        for (int i = 0; i < k; i++) {
            double[] roots = equations[i].findRoots();
            for (double root : roots) {
                if (root > maxRoot) {
                    maxRoot = root;
                }
                if (root < minRoot) {
                    minRoot = root;
                }
            }
        }

        System.out.println("\nНаибольший корень: " + maxRoot);
        System.out.println("Наименьший корень: " + minRoot);
    }
}
```

Задание №3

Формулировка задания и код программы представлены в листинге 3:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

/*
    Phone: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки,
Дебет, Кредит,
    Время городских и междугородных разговоров. Создать массив объектов.
    Вывести: a) сведения об абонентах, у которых время внутригородских
разговоров превышает заданное;
    b) сведения об абонентах, которые пользовались междугородной связью;
    c) сведения об абонентах в алфавитном порядке.
*/

class Phone {
    private int id;
    private String lastName;
    private String firstName;
    private String middleName;
    private String address;
    private String creditCardNumber;
    private double debit;
    private double credit;
    private double localCallsTime;
    private double longDistanceCallsTime;

    public Phone(int id, String lastName, String firstName, String
middleName, String address, String creditCardNumber, double debit, double
credit, double localCallsTime, double longDistanceCallsTime) {
        this.id = id;
```

```
        this.lastName = lastName;
        this.firstName = firstName;
        this.middleName = middleName;
        this.address = address;
        this.creditCardNumber = creditCardNumber;
        this.debit = debit;
        this.credit = credit;
        this.localCallsTime = localCallsTime;
        this.longDistanceCallsTime = longDistanceCallsTime;
    }

    public int getId() {
        return id;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getMiddleName() {
        return middleName;
    }

    public void setMiddleName(String middleName) {
        this.middleName = middleName;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getCreditCardNumber() {
        return creditCardNumber;
    }

    public void setCreditCardNumber(String creditCardNumber) {
        this.creditCardNumber = creditCardNumber;
    }

    public double getDebit() {
        return debit;
    }
```

```
    public void setDebit(double debit) {
        this.debit = debit;
    }

    public double getCredit() {
        return credit;
    }

    public void setCredit(double credit) {
        this.credit = credit;
    }

    public double getLocalCallsTime() {
        return localCallsTime;
    }

    public void setLocalCallsTime(double localCallsTime) {
        this.localCallsTime = localCallsTime;
    }

    public double getLongDistanceCallsTime() {
        return longDistanceCallsTime;
    }

    public void setLongDistanceCallsTime(double longDistanceCallsTime) {
        this.longDistanceCallsTime = longDistanceCallsTime;
    }

    @Override
    public String toString() {
        return "Phone{" +
                "id=" + id +
                ", lastName='" + lastName + '\'' +
                ", firstName='" + firstName + '\'' +
                ", middleName='" + middleName + '\'' +
                ", address='" + address + '\'' +
                ", creditCardNumber=" + creditCardNumber +
                ", debit=" + debit +
                ", credit=" + credit +
                ", localCallsTime=" + localCallsTime +
                ", longDistanceCallsTime=" + longDistanceCallsTime +
                '}';
    }

    public static ArrayList<Phone> filterByLocalCallsTime(ArrayList<Phone>
phoneList, double time) {
        ArrayList<Phone> result = new ArrayList<>();
        for (Phone phone : phoneList) {
            if (phone.getLocalCallsTime() > time) {
                result.add(phone);
            }
        }
        return result;
    }

    public static ArrayList<Phone> filterByLongDistanceUsage(ArrayList<Phone>
phoneList) {
        ArrayList<Phone> result = new ArrayList<>();
        for (Phone phone : phoneList) {
            if (phone.getLongDistanceCallsTime() > 0) {
                result.add(phone);
            }
        }
```

```
        return result;
    }

    public static ArrayList<Phone> sortByLastName(ArrayList<Phone> phoneList)
{
        Collections.sort(phoneList,
Comparator.comparing(Phone::getLastName));
        return phoneList;
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Phone> phoneList = new ArrayList<>();

        // Добавляем данные в список абонентов
        phoneList.add(new Phone(1, "Иванов", "Иван", "Иванович", "Москва",
"1234 5678 9012 3456", 100.0, 50.0, 20.0, 10.0));
        phoneList.add(new Phone(2, "Петров", "Петр", "Петрович", "Санкт-
Петербург", "9876 5432 1098 7654", 150.0, 30.0, 15.0, 0.0));

        // Выводим информацию об абонентах с временем внутригородских
разговоров более 10 минут
        System.out.println("Абоненты с временем внутригородских разговоров
более 10 минут:");
        ArrayList<Phone> filteredByLocalCalls =
Phone.filterByLocalCallsTime(phoneList, 10.0);
        for (Phone phone : filteredByLocalCalls) {
            System.out.println(phone);
        }

        // Выводим информацию об абонентах, которые пользовались
междугородней связью
        System.out.println("\nАбоненты, которые пользовались междугородней
связью:");
        ArrayList<Phone> filteredByLongDistanceUsage =
Phone.filterByLongDistanceUsage(phoneList);
        for (Phone phone : filteredByLongDistanceUsage) {
            System.out.println(phone);
        }

        // Выводим информацию об абонентах в алфавитном порядке
        System.out.println("\nАбоненты в алфавитном порядке:");
        ArrayList<Phone> sortedByLastName = Phone.sortByLastName(phoneList);
        for (Phone phone : sortedByLastName) {
            System.out.println(phone);
        }

    }
}
```

Задание №4

Формулировка задания и код программы представлены в листинге 4:

```
import java.util.ArrayList;

/*
    Car: id, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.
    Создать массив объектов. Вывести: a) список автомобилей заданной марки;
    b) список автомобилей заданной модели, которые эксплуатируются больше n
лет;
```

```
    c) список автомобилей заданного года выпуска, цена которых больше
указанной.
*/

class Car {
    private int id;
    private String brand;
    private String model;
    private int year;
    private String color;
    private double price;
    private String regNumber;

    public Car(int id, String brand, String model, int year, String color,
double price, String regNumber) {
        this.id = id;
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.color = color;
        this.price = price;
        this.regNumber = regNumber;
    }

    public int getId() {
        return id;
    }

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

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

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

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
```

```
    public double getPrice() {
        return price;
    }

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

    public String getRegNumber() {
        return regNumber;
    }

    public void setRegNumber(String regNumber) {
        this.regNumber = regNumber;
    }

    @Override
    public String toString() {
        return "Car{" +
                "id=" + id +
                ", brand='" + brand + '\'' +
        ", model='" + model + '\'' +
        ", year=" + year +
                ", color='" + color + '\'' +
        ", price=" + price +
                ", regNumber='" + regNumber +
        '}';
    }

    public static ArrayList<Car> filterByBrand(ArrayList<Car> carList, String
brand) {
        ArrayList<Car> result = new ArrayList<>();
        for (Car car : carList) {
            if (car.getBrand().equalsIgnoreCase(brand)) {
                result.add(car);
            }
        }
        return result;
    }

    public static ArrayList<Car> filterByModelAndYears(ArrayList<Car>
carList, String model, int years) {
        ArrayList<Car> result = new ArrayList<>();
        for (Car car : carList) {
            if (car.getModel().equalsIgnoreCase(model) && (2024 -
car.getYear()) > years) {
                result.add(car);
            }
        }
        return result;
    }

    public static ArrayList<Car> filterByYearAndPrice(ArrayList<Car> carList,
int year, double price) {
        ArrayList<Car> result = new ArrayList<>();
        for (Car car : carList) {
            if (car.getYear() == year && car.getPrice() > price) {
                result.add(car);
            }
        }
        return result;
    }
```

```
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Car> carList = new ArrayList<>();

        // Добавляем данные об автомобилях в список
        carList.add(new Car(1, "Toyota", "Camry", 2018, "Black", 25000.0,
"AB1234"));
        carList.add(new Car(2, "Honda", "Civic", 2015, "White", 20000.0,
"CD5678"));

        // Выводим список автомобилей заданной марки
        System.out.println("Список автомобилей марки Toyota:");
        ArrayList<Car> filteredByBrand = Car.filterByBrand(carList,
"Toyota");
        for (Car car : filteredByBrand) {
            System.out.println(car);
        }

        // Выводим список автомобилей заданной модели, которые
эксплуатируются больше n лет
        System.out.println("\nСписок автомобилей модели Civic, которые
эксплуатируются больше 5 лет:");
        ArrayList<Car> filteredByModelAndYears =
Car.filterByModelAndYears(carList, "Civic", 5);
        for (Car car : filteredByModelAndYears) {
            System.out.println(car);
        }

        // Выводим список автомобилей заданного года выпуска, цена которых
больше указанной
        System.out.println("\nСписок автомобилей 2018 года выпуска с ценой
выше 24000:");
        ArrayList<Car> filteredByYearAndPrice =
Car.filterByYearAndPrice(carList, 2018, 24000.0);
        for (Car car : filteredByYearAndPrice) {
            System.out.println(car);
        }

    }
}
```

Задание №5

Формулировка задания и код программы представлены в листинге 5:

```
import java.util.Arrays;
import java.util.Objects;

/*
    Создать объект класса Дерево, используя классы Лист.
    Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.
*/

class Tree {
    private Leaf[] leaves;

    public Tree(Leaf[] leaves) {
        this.leaves = leaves;
    }
```

```
    public void blossom() {
        System.out.println("The tree is blooming.");
    }

    public void fallLeaves() {
        System.out.println("Leaves are falling.");
    }

    public void coverWithHoarfrost() {
        System.out.println("The tree is covered with hoarfrost.");
    }

    public void turnYellow() {
        for (Leaf leaf : leaves) {
            leaf.setColor("yellow");
        }
        System.out.println("Leaves are turning yellow.");
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Tree tree = (Tree) obj;
        return Arrays.equals(leaves, tree.leaves);
    }

    @Override
    public int hashCode() {
        return Arrays.hashCode(leaves);
    }

    @Override
    public String toString() {
        return "Tree{" +
                "leaves=" + Arrays.toString(leaves) +
                '}';
    }
}

class Leaf {
    private String color;

    public Leaf(String color) {
        this.color = color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
```

```
        Leaf leaf = (Leaf) obj;
        return color.equals(leaf.color);
    }

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

    @Override
    public String toString() {
        return "Leaf{" +
                "color='" + color + '\'' +
                '}';
    }
}

public class Main {
    public static void main(String[] args) {
        Leaf leaf1 = new Leaf("green");
        Leaf leaf2 = new Leaf("red");

        Leaf[] leaves = {leaf1, leaf2};

        Tree tree = new Tree(leaves);

        tree.blossom();
        tree.fallLeaves();
        tree.coverWithHoarfrost();

        System.out.println(leaf1.equals(leaf2));

        tree.turnYellow();

        System.out.println(leaf1.hashCode());
        System.out.println(leaf1.toString());

        System.out.println(tree.equals(new Tree(leaves)));
        System.out.println(tree.hashCode());
        System.out.println(tree.toString());
    }
}
```

Задание №6

Формулировка задания и код программы представлены в листинге 6:

```
import java.util.Objects;
import java.util.Arrays;

/*
    Создать объект класса Пианино, используя класс Клавиша.
    Методы: настроить, играть на пианино, нажимать клавишу.
*/

class Piano {
    private String brand;
    private Key[] keys;

    public Piano(String brand, Key[] keys) {
        this.brand = brand;
        this.keys = keys;
```

```
    }

    public void tune() {
        System.out.println("Piano is tuned.");
    }

    public void play() {
        System.out.println("Playing the piano.");
    }

    public void pressKey(int keyIndex) {
        if (keyIndex >= 0 && keyIndex < keys.length) {
            System.out.println("Key " + keys[keyIndex].getNote() + " is
pressed.");
        } else {
            System.out.println("Invalid key index.");
        }
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Piano piano = (Piano) obj;
        return brand.equals(piano.brand) && Arrays.equals(keys, piano.keys);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(brand);
        result = 31 * result + Arrays.hashCode(keys);
        return result;
    }

    @Override
    public String toString() {
        return "Piano{" +
                "brand='" + brand + '\'' +
                ", keys=" + Arrays.toString(keys) +
                '}';
    }
}

class Key {
    private String note;

    public Key(String note) {
        this.note = note;
    }

    public String getNote() {
        return note;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
```

```
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Key key = (Key) obj;
        return note.equals(key.note);
    }

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

    @Override
    public String toString() {
        return "Key{" +
                "note='" + note + '\'' +
                '}';
    }
}

public class Main {
    public static void main(String[] args) {
        Key key1 = new Key("C");
        Key key2 = new Key("D");

        Key[] keys = {key1, key2};

        Piano piano = new Piano("Yamaha", keys);

        piano.tune();
        piano.play();
        piano.pressKey(0);

        System.out.println(key1.equals(key2));
        System.out.println(key1.hashCode());
        System.out.println(key1.toString());

        System.out.println(piano.equals(new Piano("Yamaha", keys)));
        System.out.println(piano.hashCode());
        System.out.println(piano.toString());
    }
}
```

Задание №7

Формулировка задания и код программы представлены в листинге 7:

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

/*
    Система Автобаза. Диспетчер распределяет заявки на Рейсы между Водителями
    и назначает для этого Автомобиль. Водитель может сделать заявку на
ремонт.
    Диспетчер может отстранить Водителя от работы. Водитель делает отметку
    о выполнении Рейса и состоянии Автомобиля.
*/

class Dispatcher {
    private List<Driver> drivers;
    private List<Car> cars;
    private List<Trip> trips;
```

```
    public Dispatcher(List<Driver> drivers, List<Car> cars, List<Trip> trips)
{
        this.drivers = drivers;
        this.cars = cars;
        this.trips = trips;
    }

    public void assignTrip(Driver driver, Trip trip, Car car) {
        if (drivers.contains(driver) && cars.contains(car)) {
            driver.setCar(car);
            trips.add(trip);
            System.out.println("Trip assigned to driver: " +
driver.getName());
        } else {
            System.out.println("Driver or car not found.");
        }
    }

    public void suspendDriver(Driver driver) {
        if (drivers.contains(driver)) {
            driver.setSuspended(true);
            System.out.println("Driver suspended: " + driver.getName());
        } else {
            System.out.println("Driver not found.");
        }
    }
}

class Driver {
    private String name;
    private Car car;
    private boolean suspended;

    public Driver(String name) {
        this.name = name;
        this.suspended = false;
    }

    public void makeRepairRequest() {
        System.out.println("Repair request made by driver: " + name);
    }

    public void makeTripReport(Trip trip, String carCondition) {
        System.out.println("Trip report submitted by driver: " + name);
        trip.setCompleted(true);
        car.setCarCondition(carCondition);
    }

    public void setCar(Car car) {
        this.car = car;
    }

    public void setSuspended(boolean suspended) {
        this.suspended = suspended;
    }

    public String getName() {
        return name;
    }
}

class Car {
```

```
    private String model;

    private String carCondition;

    public Car(String model) {
        this.model = model;
        this.carCondition = "";
    }

    public void setCarCondition(String carCondition) {
        this.carCondition = carCondition;
    }
}

class Trip {
    private boolean completed;

    public Trip() {
        this.completed = false;
    }

    public void setCompleted(boolean completed) {
        this.completed = completed;
    }
}

public class Main {
    public static void main(String[] args) {
        Driver driver1 = new Driver("John");
        Driver driver2 = new Driver("Alice");

        Car car1 = new Car("Toyota");
        Car car2 = new Car("Ford");

        List<Driver> drivers = new ArrayList<>();
        drivers.add(driver1);
        drivers.add(driver2);

        List<Car> cars = new ArrayList<>();
        cars.add(car1);
        cars.add(car2);

        List<Trip> trips = new ArrayList<>();

        Dispatcher dispatcher = new Dispatcher(drivers, cars, trips);

        Trip trip1 = new Trip();
        dispatcher.assignTrip(driver1, trip1, car1);

        driver1.makeRepairRequest();

        driver1.makeTripReport(trip1, "Good condition");

        dispatcher.suspendDriver(driver2);
    }
}
```

Задание №8

Формулировка задания и код программы представлены в листинге 8:

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

/*
    Система Телефонная станция. Абонент оплачивает Счет за разговоры и
Услуги,
    может попросить Администратора сменить номер и отказаться от услуг.
    Администратор изменяет номер, Услуги и временно отключает Абонента за
неуплату.
*/

class Account {
    private double balance;

    public Account(double initialBalance) {
        this.balance = initialBalance;
    }

    public void debit(double amount) {
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}

class Subscriber {
    private String number;
    private Account account;
    private List<String> services;
    private boolean active;

    public Subscriber(String number, double initialBalance) {
        this.number = number;
        this.account = new Account(initialBalance);
        this.services = new ArrayList<>();
        this.active = true;
    }

    public void changeNumber(String newNumber) {
        this.number = newNumber;
        System.out.println("Number changed to: " + newNumber);
    }

    public void addService(String service) {
        services.add(service);
        System.out.println("Service added: " + service);
    }

    public void removeService(String service) {
        services.remove(service);
        System.out.println("Service removed: " + service);
    }

    public void deactivateSubscriber() {
        active = false;
        System.out.println("Subscriber deactivated.");
    }

    public void payBill(double amount) {
        if (account.getBalance() >= amount) {
            account.debit(amount);
```

```
            System.out.println("Payment successful. Remaining balance: " +
account.getBalance());
        } else {
            System.out.println("Insufficient funds to pay the bill.");
        }
    }

    public boolean isActive() {
        return active;
    }
}

class Administrator {
    public void changeNumber(Subscriber subscriber, String newNumber) {
        subscriber.changeNumber(newNumber);
    }

    public void addService(Subscriber subscriber, String service) {
        subscriber.addService(service);
    }

    public void removeService(Subscriber subscriber, String service) {
        subscriber.removeService(service);
    }

    public void deactivateSubscriber(Subscriber subscriber) {
        subscriber.deactivateSubscriber();
    }
}

public class Main {
    public static void main(String[] args) {
        Subscriber subscriber = new Subscriber("123456789", 100.0);
        Administrator admin = new Administrator();

        admin.addService(subscriber, "Voicemail");
        admin.addService(subscriber, "Call Waiting");

        subscriber.payBill(50.0);

        admin.changeNumber(subscriber, "987654321");

        admin.removeService(subscriber, "Voicemail");

        subscriber.payBill(80.0); // Attempt to pay more than the balance

        admin.deactivateSubscriber(subscriber);
    }
}
```

Вывод: в результате выполнения лабораторной работы были освоены основные принципы ООП на языке Java.