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 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		Т.И. Кадыров
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			П.В. Степанов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы: освоить базовые принципы работы с ООП на языке Java.

Вариант: 8.

Задание 1: Определить класс Комплекс. Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения, деления, присваивания комплексных чисел. Создать два вектора размерности n из комплексных координат. Передать их в метод, который выполнит их сложение.

Код решения приведен в листинге 1.

Листинг 1 — реализация решения

```
public class Complex {
    private double real;
    private double imaginary;

    public Complex() {
        this.real = 0;
        this.imaginary = 0;
    }

    public Complex(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public Complex(Complex other) {
        this.real = other.real;
        this.imaginary = other.imaginary;
    }

    public Complex add(Complex other) {
        return new Complex(this.real + other.real, this.imaginary + other.imaginary);
    }

    public Complex subtract(Complex other) {
        return new Complex(this.real - other.real, this.imaginary - other.imaginary);
    }

    public Complex multiply(Complex other) {
        double real = this.real * other.real - this.imaginary * other.imaginary;
        double imaginary = this.real * other.imaginary + this.imaginary * other.real;
        return new Complex(real, imaginary);
    }

    public Complex divide(Complex other) {
        double denominator = other.real * other.real + other.imaginary * other.imaginary;
        double real = (this.real * other.real + this.imaginary * other.imaginary) / denominator;
        double imaginary = (this.imaginary * other.real - this.real * other.imaginary) / denominator;
        return new Complex(real, imaginary);
```

```
    }

    public void set(Complex other) {
        this.real = other.real;
        this.imaginary = other.imaginary;
    }

    public static Complex[] sumOfVectors(Complex[] vector1, Complex[] vector2) {
        if (vector1.length != vector2.length)
            throw new IllegalArgumentException("Vectors must be of the same length");

        Complex[] sum = new Complex[vector1.length];
        for (int i = 0; i < vector1.length; i++) {
            sum[i] = vector1[i].add(vector2[i]);
        }
        return sum;
    }

    public String toString() {
        return this.real + " + " + this.imaginary + "i";
    }
}

public class Main {
    public static void main(String[] args) {
        Complex[] vector1 = new Complex[]{
                new Complex(1, 1), new Complex(1, 1), new Complex(0, 0)
        };
        Complex[] vector2 = new Complex[]{
                new Complex(1, 1), new Complex(2, 2), new Complex(0, 0)
        };
        Complex[] res = Complex.sumOfVectors(vector1, vector2);
        for (int i = 0; i < res.length; i++) {
            System.out.println(res[i]);
        }
    }
}
```

Задание 2: 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 решения приведен в листинге 2.

Листинг 2 — реализация решения

```
public class QuadraticEquation {
    private double a;
    private double b;
    private double c;
```

```
public QuadraticEquation(double a, double b, double c) throws IllegalArgumentException {
    if (a == 0) {
        throw new IllegalArgumentException("A is zero");
    }
    this.a = a;
    this.b = b;
    this.c = c;
}

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

public double[] findRoots() {
    double discriminant = b * b - 4 * a * c;
    if (discriminant < 0) {
        return new double[0];
    } else if (discriminant == 0) {
        return new double[]{-b / (2 * a)};
    } else {
        double sqrtDiscriminant = Math.sqrt(discriminant);
        return new double[]{
                (-b + sqrtDiscriminant) / (2 * a),
                (-b - sqrtDiscriminant) / (2 * a)
        };
    }
}

public double[] findExtremePoint() {
    double x = -b / (2 * a);
    double y = a * x * x + b * x + c;
    return new double[]{x, y};
}

public String findIntervalOfIncrease() {
    if (a > 0) {
        return "(-∞, +∞)";
    } else if (a < 0) {
        double extremePoint = findExtremePoint()[0];
        return "(-∞, " + extremePoint + ")";
    } else {
        return "There's no interval of increase or decrease.";
    }
}

public String findIntervalOfDecrease() {
    if (a < 0) {
        return "(-∞, +∞)";
    } else if (a > 0) {
        double extremePoint = findExtremePoint()[0];
        return "(" + extremePoint + ", +∞)";
    } else {
        return "There's no interval of increase or decrease.";
    }
}
```

```
}

public class Main {
  public static void main(String[] args) {
    QuadraticEquation[] equations = {
        new QuadraticEquation(1, -3, 2),
        new QuadraticEquation(1, 4, 4),
        new QuadraticEquation(1, -6, 9)
    };

    double minRoot = equations[0].findRoots()[0];
    double maxRoot = equations[0].findRoots()[0];

    for (QuadraticEquation equation : equations) {
      double[] roots = equation.findRoots();
      for (double root : roots) {
        if (!Double.isNaN(root)) {
          if (root < minRoot) {
            minRoot = root;
          }
          if (root > maxRoot) {
            maxRoot = root;
          }
        }
      }
    }

    System.out.println("Minimum root: " + minRoot);
    System.out.println("Maximum root: " + maxRoot);
  }
}
```

Задание 3: 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.

Код решения приведен в листинге 3.

Листинг 3 — реализация решения

```
package transport;

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;

public class Car {
  private int id;
  private String brand;
```

```
    private String model;
    private int year;
    private String color;
    private double price;
    private String registrationNumber;

    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 getRegistrationNumber() {
```

```
    return registrationNumber;
  }

  public void setRegistrationNumber(String registrationNumber) {
    this.registrationNumber = registrationNumber;
  }

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

  public String toString() {
    return "Car ID: " + id + ", Brand: " + brand + ", Model: " + model + ", Year: " + year + ", Color: " + color + ", Price: " +
price + ", Registration Number: " + registrationNumber;
  }

  public long getYearsInUse() {
    int currentYear = LocalDate.now().getYear();
    return ChronoUnit.YEARS.between(LocalDate.of(year, 1, 1), LocalDate.of(currentYear, 1, 1));
  }
}

package transport;

import java.util.ArrayList;

public class CarManager {
  private ArrayList<Car> cars = new ArrayList<>();

  public void addCar(Car car) {
    cars.add(car);
  }

  public ArrayList<Car> getCarsByBrand(String brand) {
    ArrayList<Car> carsByBrand = new ArrayList<>();
    for (Car car : cars) {
      if (car.getBrand().equalsIgnoreCase(brand)) {
        carsByBrand.add(car);
      }
    }
    return carsByBrand;
  }

  public ArrayList<Car> getCarsByModelAndYears(String model, int years) {
    ArrayList<Car> carsByModelAndYears = new ArrayList<>();
    for (Car car : cars) {
      if (car.getModel().equalsIgnoreCase(model) && car.getYearsInUse() > years) {
        carsByModelAndYears.add(car);
      }
```

```
        }
        return carsByModelAndYears;
    }

    public ArrayList<Car> getCarsByYearAndPrice(int year, double price) {
        ArrayList<Car> carsByYearAndPrice = new ArrayList<>();
        for (Car car : cars) {
            if (car.getYear() == year && car.getPrice() > price) {
                carsByYearAndPrice.add(car);
            }
        }
        return carsByYearAndPrice;
    }
}

import transport.Car;
import transport.CarManager;

import java.util.ArrayList;

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

        CarManager carManager = new CarManager();
        carManager.addCar(new Car(1, "Toyota", "Corolla", 2010, "Red", 15000, "1234AB"));
        carManager.addCar(new Car(2, "Honda", "Civic", 2015, "Blue", 20000, "5678CD"));
        carManager.addCar(new Car(3, "Toyota", "Camry", 2018, "White", 25000, "9012EF"));

        ArrayList<Car> toyotaCars = carManager.getCarsByBrand("Toyota");
        System.out.println("Cars by brand Toyota:");
        for (Car car : toyotaCars) {
            System.out.println(car);
        }

        ArrayList<Car> oldCivicCars = carManager.getCarsByModelAndYears("Civic", 5);
        System.out.println("Older than 5 years Civic cars:");
        for (Car car : oldCivicCars) {
            System.out.println(car);
        }

        ArrayList<Car> expensive2018Cars = carManager.getCarsByYearAndPrice(2018, 20000);
        System.out.println("Expensive 2018 cars:");
        for (Car car : expensive2018Cars) {
            System.out.println(car);
        }
    }
}
```

Задание 4: Product: id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество. 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 решения приведен в листинге 4.

Листинг 4 — реализация решения

```
package product;

import java.util.ArrayList;

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;
    }

    public String toString() {
        return "Product ID: " + id + ", Name: " + name + ", UPC: " + UPC + ", Manufacturer: " + manufacturer + ", Price: " + price
+ ", Shelf Life: " + shelfLife + ", Quantity: " + quantity;
    }
}

package product;

import java.util.ArrayList;

public class ProductManager {
    private ArrayList<Product> products = new ArrayList<>();

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

    public ArrayList<Product> getProductsByName(String name) {
        ArrayList<Product> productsByName = new ArrayList<>();
        for (Product product : products) {
            if (product.getName().equalsIgnoreCase(name)) {
                productsByName.add(product);
            }
```

```
        }
        return productsByName;
    }

    public ArrayList<Product> getProductsByNameAndPrice(String name, double maxPrice) {
        ArrayList<Product> productsByNameAndPrice = new ArrayList<>();
        for (Product product : products) {
            if (product.getName().equalsIgnoreCase(name) && product.getPrice() <= maxPrice) {
                productsByNameAndPrice.add(product);
            }
        }
        return productsByNameAndPrice;
    }

    public ArrayList<Product> getProductsByShelfLife(int minShelfLife) {
        ArrayList<Product> productsByShelfLife = new ArrayList<>();
        for (Product product : products) {
            if (product.getShelfLife() > minShelfLife) {
                productsByShelfLife.add(product);
            }
        }
        return productsByShelfLife;
    }
}

import product.Product;
import product.ProductManager;

import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ProductManager productManager = new ProductManager();
        productManager.addProduct(new Product(1, "Milk", "123456789012", "Farmers Inc.", 2.50, 7, 10));
        productManager.addProduct(new Product(2, "Bread", "234567890123", "Bakery Ltd.", 1.50, 5, 20));
        productManager.addProduct(new Product(3, "Eggs", "345678901234", "Eggcellent Farms", 3.00, 10, 15));

        ArrayList<Product> milkProducts = productManager.getProductsByName("Milk");
        System.out.println("Products with name Milk:");
        for (Product product : milkProducts) {
            System.out.println(product);
        }

        ArrayList<Product> cheapBreadProducts = productManager.getProductsByNameAndPrice("Bread", 2.00);
        System.out.println("Cheap Bread products:");
        for (Product product : cheapBreadProducts) {
            System.out.println(product);
        }

        ArrayList<Product> longShelfLifeProducts = productManager.getProductsByShelfLife(6);
        System.out.println("Products with shelf life longer than 6 days:");
        for (Product product : longShelfLifeProducts) {
            System.out.println(product);
        }
    }
```

```
}
```

Задание 5: Создать объект класса Пианино, используя класс Клавиша. Методы: настроить, играть на пианино, нажимать клавишу.

Код решения приведен в листинге 5.

Листинг 5 — реализация решения

```
import java.util.Objects;

public class Key implements PianoComponent {
    private String note;

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

    @Override
    public void press() {
        System.out.println("Key " + note + " is pressed.");
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Key key = (Key) o;
        return Objects.equals(note, key.note);
    }

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

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

public interface PianoComponent {
    void press();
}

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

public class Piano implements PianoComponent {
```

```
    private List<PianoComponent> keys = new ArrayList<>();

    public void addKey(PianoComponent key) {
        keys.add(key);
    }

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

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

    @Override
    public void press() {
        System.out.println("Playing the piano.");
        for (PianoComponent key : keys) {
            key.press();
        }
    }

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

    @Override
    public int hashCode() {
        return this.keys.hashCode();
    }

    @Override
    public String toString() {
        return "Piano{" +
                "keys=" + this.keys.toString() +
                '}';
    }
}

public class Main {
    public static void main(String[] args) {
        Key key1 = new Key("C2");
        Key key2 = new Key("D1");
        Key key3 = new Key("E0");

        Piano piano = new Piano();
        piano.addKey(key1);
        piano.addKey(key2);
        piano.addKey(key3);

        piano.press();
```

```
    }
}
```

Задание 6: Создать объект класса Фотоальбом, используя класс Фотография. Методы: задать название фотографии, дополнить фотоальбом фотографией, вывести на консоль количество фотографий.

Код решения приведен в листинге 6.

Листинг 6 — реализация решения

```
public class Photo {
    private String title;

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

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Photo photo = (Photo) o;
        return title.equals(photo.title);
    }

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

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

import java.util.ArrayList;

public class PhotoAlbum {
    private ArrayList<Photo> photos = new ArrayList<>();

    public void addPhoto(Photo photo) {
```

```
        photos.add(photo);
    }

    public void printNumberOfPhotos() {
        System.out.println("Number of photos in the album: " + 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 photos.equals(that.photos);
    }

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

    @Override
    public String toString() {
        return "PhotoAlbum{" +
                "photos=" + photos +
                '}';
    }
}

public class Main {
    public static void main(String[] args) {
        PhotoAlbum album = new PhotoAlbum();

        album.addPhoto(new Photo("Summer Vacation"));
        album.addPhoto(new Photo("Family Reunion"));
        album.addPhoto(new Photo("Birthday Party"));

        album.printNumberOfPhotos();
    }
}
```

Задание 7: 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.

Код решения приведен в листинге 7.

Листинг 7 — реализация решения

```
package service;

public class Car {
```

```
    private String model;
    private String registrationNumber;
    private int condition;
    private boolean isAvailable;

    public Car(String model, String registrationNumber) {
        this.model = model;
        this.registrationNumber = registrationNumber;
        this.condition = 100;
        this.isAvailable = true;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    public String getModel() {
        return model;
    }

    public String getRegistrationNumber() {
        return registrationNumber;
    }

    public int getCondition() {
        return condition;
    }

    public void setCondition(int condition) {
        this.condition = condition;
    }
}

package service;

public class Driver {
    private static int counter = 1;
    private int id;
    private String name;
    private boolean available;

    public Driver(String name) {
        this.name = name;
        this.available = true;
        id = counter++;
    }

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

```
    public String getName() {
        return name;
    }

    public boolean isAvailable() {
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }

    public void requestRepair(Car car) {
        car.setAvailable(false);
        System.out.println(name + " запросил ремонт.");
    }

    public void completeTrip(Trip trip, int condition) {
        trip.getCar().setCondition(condition);
        trip.getCar().setAvailable(true);
        if (condition < 80) {
            requestRepair(trip.getCar());
        }
        trip.getDriver().setAvailable(true);
        System.out.println(name + " выполнил рейс на автомобиле " + trip.getCar().getRegistrationNumber());
    }
}

package service;

public class Trip {
    private Car car;
    private Driver driver;

    public Trip() {
    }

    public Trip(Car car, Driver driver) {
        this.car = car;
        this.driver = driver;
    }

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

    public void setDriver(Driver driver) {
        this.driver = driver;
    }

    public Car getCar() {
        return car;
    }

    public Driver getDriver() {
```

```
        return driver;
    }
}


package service;

import com.sun.jmx.remote.internal.ArrayQueue;

import java.util.*;

public class Dispatcher {
    private ArrayList<Driver> drivers = new ArrayList<>();
    private ArrayList<Car> cars = new ArrayList<>();

    public void addDriver(Driver driver) {
        drivers.add(0, driver);
    }

    public void removeDriver(Driver driver) {
        drivers.remove(driver);
    }

    public void addCar(Car car) {
        cars.add(0, car);
    }

    public Trip assignTrip(Trip trip) {
        for (int i = 0; i < drivers.size(); i++) {
            if (drivers.get(i).isAvailable()) {
                drivers.get(i).setAvailable(false);
                trip.setDriver(drivers.get(i));
                break;
            }
        }
        for (int i = 0; i < cars.size(); i++) {
            if (cars.get(i).isAvailable()) {
                cars.get(i).setAvailable(false);
                trip.setCar(cars.get(i));
                break;
            }
        }
        if (trip.getCar() != null && trip.getDriver() != null) {
            System.out.println("Водитель " + trip.getDriver().getName() + " отправлен в рейс на автомобиле " +
trip.getCar().getRegistrationNumber());
        } else {
            System.out.println("Не удалось назначить рейс. Водитель не доступен или автомобиль не найден.");
        }
        return trip;
    }

    public void suspendDriver(Driver driver) {
        for (int i = 0; i < drivers.size(); i++) {
            if (drivers.get(i).getId() == driver.getId()) {
                drivers.remove(i);
```

```
            break;
        }
      }
      System.out.println("Водитель " + driver.getName() + " уволен.");
    }
}

import service.*;

public class Main {
    public static void main(String[] args) {
        Dispatcher dispatcher = new Dispatcher();

        Driver driver1 = new Driver("Вася");
        Driver driver2 = new Driver("Петя");
        dispatcher.addDriver(driver1);
        dispatcher.addDriver(driver2);

        Car car1 = new Car("Toyota", "ABC123");
        Car car2 = new Car("Honda", "XYZ789");
        dispatcher.addCar(car1);
        dispatcher.addCar(car2);

        Trip trip1 = new Trip();
        Trip trip2 = new Trip();

        trip1 = dispatcher.assignTrip(trip1);
        trip2 = dispatcher.assignTrip(trip2);

        trip1.getDriver().completeTrip(trip1, 90);
        trip2.getDriver().completeTrip(trip2, 90);

        dispatcher.suspendDriver(driver1);
        dispatcher.suspendDriver(driver2);

        Trip trip3 = new Trip();
        trip3 = dispatcher.assignTrip(trip3);
    }
}
```

Задание 8: Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».

Код решения приведен в листинге 8.

Листинг 8 — реализация решения

```
package shop;

import java.util.ArrayList;
```

```
import java.util.List;

public class Admin {
    private List<Product> productList;
    private List<String> blacklist;

    public Admin() {
        productList = new ArrayList<>();
        blacklist = new ArrayList<>();
    }

    public void addProduct(Product product) {
        productList.add(product);
    }

    public void registerSale(Order order, boolean paymentStatus) {
        SalesRecord salesRecord = new SalesRecord(order, paymentStatus);
    }

    public void addToBlacklist(String customer) {
        blacklist.add(customer);
    }
}

package shop;

public class Client {
    private String name;
    private double balance;

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

    public void placeOrder(Order order) {
        if (balance >= order.calculateTotalPrice()) {
            balance -= order.calculateTotalPrice();
            order.setPaid(true);
        } else {
            System.out.println("Недостаточно средств на счете для совершения покупки.");
        }
    }
}

package shop;

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

public class Order {
    private List<Product> products;
    private boolean paid;

    public Order() {
```

```
        products = new ArrayList<>();
        paid = false;
    }

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

    public double calculateTotalPrice() {
        double totalPrice = 0;
        for (Product product : products) {
            totalPrice += product.getPrice();
        }
        return totalPrice;
    }

    public boolean isPaid() {
        return paid;
    }

    public void setPaid(boolean paid) {
        this.paid = paid;
    }
}

package shop;

public class Product {
    private String name;
    private double price;

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

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

package shop;

public class SalesRecord {
    private Order order;
    private boolean paymentStatus;

    public SalesRecord(Order order, boolean paymentStatus) {
        this.order = order;
        this.paymentStatus = paymentStatus;
    }
```

```
    public Order getOrder() {
        return order;
    }

    public boolean isPaymentStatus() {
        return paymentStatus;
    }
}

import shop.*;

public class Main {
    public static void main(String[] args) {
        Admin admin = new Admin();
        Product laptop = new Product("Laptop", 1000);
        admin.addProduct(laptop);

        Client client = new Client("Alice", 1500);
        Order order = new Order();
        order.addProduct(laptop);
        client.placeOrder(order);

        admin.registerSale(order, order.isPaid());
    }
}
```

Вывод: в ходе лабораторной работы были освоены базовые принципы работы с ООП на языке Java.