Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5

По дисциплине: «ССП»
Вариант - 12

Выполнил:
Студент 3 курса
Группы ПО-8
Иванюк М.С.
Проверил:
Крощенко А.А

Брест, 2024

Лабораторная работа №5

Цель работы: приобрести практические навыки в области объектно-ориентированного проектирования.

Задание 1: Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Техника ← abstract class Плеер ← class Видеоплеер.

Код программы:

```
package Lab5;

public class task1 {
    public static void main(String[] args){
        Videoplayer videoplayer = new Videoplayer("Samsung BD-J7500","MP4");
        videoplayer.turnOn();
        videoplayer.start();
        videoplayer.start();
        videoplayer.stop();
        videoplayer.stop();
        videoplayer.turnOff();
    }
}

interface Technique{
    void turnOn();
    void turnOff();
}

abstract class Player implements Technique{
    protected String model;
    protected boolean isWorking;

    public abstract void start();
    public abstract void stop();

    Player(){
        this.model = "Undefined";
        this.isWorking = false;
    }
    Player(String name){
        this.model = name;
        this.isWorking = false;
    }

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

    public boolean isWorking(){
        return this.isWorking;
    }

    @Override
    public void turnOn(){
        this.isWorking = true;
        System.out.printf("Плеер %s включен.\n",this.model);
```

```
    }
    @Override
    public void turnOff(){
        this.isWorking = false;
        System.out.printf("Плеер %s выключен.\n",this.model);
    }

}

class Videoplayer extends Player{
    private String videoFormat;
    private boolean isPlaying;

    Videoplayer(String model,String videoFormat){
        super(model);
        this.videoFormat = videoFormat;
        this.isPlaying = false;
    }
    @Override
    public void start() {
        if(this.isWorking()){
            if(!this.isPlaying){
                this.isPlaying = true;
                System.out.printf("Воспроизведение видео на плеере %s в
формете %s.\n",this.model,this.videoFormat);
            }
            else{
                System.out.println("Видео уже воспроизводится.");
            }
        }
        else{
            System.out.println("Видеоплеер выключен. Включите плеер, чтобы
начать воспроизведение видео.");
        }
    }
    @Override
    public void stop() {
        if(this.isPlaying){
            this.isPlaying = false;
            System.out.println("Воспроизведение видео остановлено.");
        }
        else{
            System.out.println("Видео уже остановлено.");
        }

    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe" "-ja
Плеер Samsung BD-J7500 включен.
Воспроизведение видео на плеере Samsung BD-J7500 в формете MP4.
Видео уже воспроизводится.
Воспроизведение видео остановлено.
Видео уже остановлено.
Плеер Samsung BD-J7500 выключен.

Process finished with exit code 0
```

Задание 2: Создать суперкласс Грузоперевозчик и подклассы Самолет, Поезд, Автомобиль. Определить время и стоимость перевозки для указанных городов и расстояний.

Код программы:

```
package Lab5;

public class task2 {
    public static void main(String[] args){
        Route route = new Route("Брест", "Москва", 1100);

        Cargocarrier[] cargocarriers = new Cargocarrier[]{
                new Airplane("Boeing 747", route, 800, 5, 20, 1.5),
                new Train("Грузовой поезд", route, 120, 3, 30, 1),
                new Car("Грузовик", route, 100, 2, 10, 1.2)
        };



       double airplanePrice =
cargocarriers[0].calculateTransportationPrice(route.getDistance());
        double airplaneTime =
cargocarriers[0].calculateTransportationTime(route.getDistance());

        double trainPrice
=cargocarriers[1].calculateTransportationPrice(route.getDistance());
        double trainTime =
cargocarriers[1].calculateTransportationTime(route.getDistance());

        double carPrice =
cargocarriers[2].calculateTransportationPrice(route.getDistance());
        double carTime =
cargocarriers[2].calculateTransportationTime(route.getDistance());

        System.out.println("Маршрут: " + route.getStartCity() + " - " +
route.getEndCity() + ", расстояние: " + route.getDistance() + " km;");

        System.out.println("Транспорт: Самолет (Boeing 747)");
        System.out.println("Стоимость: $" + String.format("%.2f",
airplanePrice));
        System.out.println("Время доставки: " + String.format("%.2f",
airplaneTime) + " часов");

        System.out.println();

        System.out.println("Транспорт: Поезд (Грузовой поезд)");
        System.out.println("Стоимость: $" + String.format("%.2f",
trainPrice));
        System.out.println("Время доставки: " + String.format("%.2f",
trainTime) + " часов");

        System.out.println();

        System.out.println("Транспорт: Автомобиль (Грузовик)");
        System.out.println("Стоимость: $" + String.format("%.2f", carPrice));
        System.out.println("Время доставки: " + String.format("%.2f",
carTime) + " часов");
    }
}
```

```
class Route {
    private String firstCity;
    private String secondCity;
    private double distance; // Расстояние между городами в км

    public Route(String startCity, String endCity, double distance) {
        this.firstCity = startCity;
        this.secondCity = endCity;
        this.distance = distance;
    }

    public String getStartCity() {
        return firstCity;
    }

    public String getEndCity() {
        return secondCity;
    }

    public double getDistance() {
        return distance;
    }

    public void setFirstCity(String firstCity) {
        this.firstCity = firstCity;
    }

    public void setSecondCity(String secondCity) {
        this.secondCity = secondCity;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }
}

abstract class Cargocarrier{
    protected String name;
    protected Route route;
    protected double avgSpeed;
    protected double ratePerKm;
    protected double fuelConsumption; // Потребление топлива в литрах на 100
км
    protected double fuelPrice; // Цена топлива за литр

    public abstract double calculateTransportationTime(double distance);
    public abstract double calculateTransportationPrice(double distance);

    public Cargocarrier(String name, Route route, double avgSpeed, double
ratePerKm, double fuelConsumption, double fuelPrice) {
        this.name = name;
        this.route = route;
        this.avgSpeed = avgSpeed;
        this.ratePerKm = ratePerKm;
        this.fuelConsumption = fuelConsumption;
        this.fuelPrice = fuelPrice;
    }

    public String getName() {
        return name;
    }

    public Route getRoute() {
        return route;
```

```
    }

    public double getAvgSpeed() {
        return avgSpeed;
    }

    public double getRatePerKm() {
        return ratePerKm;
    }

    public double getFuelConsumption() {
        return fuelConsumption;
    }

    public double getFuelPrice() {
        return fuelPrice;
    }

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

    public void setRoute(Route route) {
        this.route = route;
    }

    public void setAvgSpeed(double avgSpeed) {
        this.avgSpeed = avgSpeed;
    }

    public void setRatePerKm(double ratePerKm) {
        this.ratePerKm = ratePerKm;
    }

    public void setFuelConsumption(double fuelConsumption) {
        this.fuelConsumption = fuelConsumption;
    }

    public void setFuelPrice(double fuelPrice) {
        this.fuelPrice = fuelPrice;
    }
}

class Airplane extends Cargocarrier{
    private double additionalFees;

    public Airplane(String name, Route route, double avgSpeed, double
ratePerKm, double fuelConsumption, double fuelPrice) {
        super(name, route, avgSpeed, ratePerKm, fuelConsumption, fuelPrice);
        this.additionalFees = 0.8;
    }

    @Override
    public double calculateTransportationTime(double distance) {
        return distance / super.avgSpeed;
    }

    @Override
    public double calculateTransportationPrice(double distance) {
        double fuelNeeded = (super.fuelConsumption / 100) * distance;
double totalFuelPrice = fuelNeeded * super.fuelPrice;        double
transportPrice = distance * super.ratePerKm;        return (totalFuelPrice +
transportPrice) * additionalFees;    }
}
```

```
class Train extends Cargocarrier{

    public Train(String name, Route route, double avgSpeed, double ratePerKm,
double fuelConsumption, double fuelPrice) {
        super(name, route, avgSpeed, ratePerKm, fuelConsumption, fuelPrice);
    }

    @Override
    public double calculateTransportationTime(double distance) {
        return distance / super.avgSpeed;
    }

    @Override
    public double calculateTransportationPrice(double distance) {
        double fuelNeeded = (super.fuelConsumption / 100) * distance;
        double totalPrice = fuelNeeded * super.fuelPrice;
        double transportPrice = distance * super.ratePerKm;
        return totalPrice + transportPrice;
    }
}

class Car extends Cargocarrier{
    private double profitMargin; // Наценка за перевозку

    public Car(String name, Route route, double avgSpeed, double ratePerKm,
double fuelConsumption, double fuelPrice) {
        super(name, route, avgSpeed, ratePerKm, fuelConsumption, fuelPrice);
        this.profitMargin = 0.4;
    }

    public void setProfitMargin(double profitMargin) {
        this.profitMargin = profitMargin;
    }

    public double getProfitMargin() {
        return profitMargin;
    }

    @Override
    public double calculateTransportationTime(double distance) {
        return distance / super.avgSpeed;
    }

    @Override
    public double calculateTransportationPrice(double distance) {
        double fuelNeeded = (super.fuelConsumption / 100) * distance;
double totalPrice = fuelNeeded * super.fuelPrice;
        double transportPrice = distance * super.ratePerKm;
        return (totalPrice + transportPrice) * profitMargin;
    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe"
Маршрут: Брест - Москва, расстояние: 1100.0 km;
Транспорт: Самолет (Boeing 747)
Стоимость: $4664,00
Время доставки: 1,38 часов

Транспорт: Поезд (Грузовой поезд)
Стоимость: $3630,00
Время доставки: 9,17 часов

Транспорт: Автомобиль (Грузовик)
Стоимость: $932,80
Время доставки: 11,00 часов

Process finished with exit code 0
```

Задание 3: В задании 3 ЛР №4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве. Студентов, Преподавателей и Курсов при обучении может быть несколько.

Код программы:

```
package Lab5;

import java.util.*;

public class task3 {
    public static void main(String[] args){
        Archive archive = new Archive();
        Student student1 = new Student("Бубен Станислав Олеговочи","ПО-8");
        Student student2 = new Student("Бувин Дмитрий Александрович","ПО-8");
        Student student3 = new Student("Назаров Кирилл Викторович","ПО-8");

        Course course1 = new Course("Компьютерные системы и сети");
        Course course2 = new Course("Проектирование интернет систем");
        Teacher teacher = new Teacher("Гречаник Татьяна Викторовна");

        teacher.addCourse(course1);
        teacher.announceCourseRegistration(course1);
        student1.enrollInCourse(course1);
        student2.enrollInCourse(course1);
        student3.enrollInCourse(course1);
        teacher.evaluateTheStudent(archive,course1,student1,new Mark(5));
        teacher.evaluateTheStudent(archive,course1,student2,new Mark(5));
        teacher.evaluateTheStudent(archive,course1,student3,new Mark(4));
        teacher.removeCourse(course1);

        System.out.println();
```

```
        teacher.addCourse(course2);
        teacher.announceCourseRegistration(course2);
        student1.enrollInCourse(course2);
        student2.enrollInCourse(course2);
        student3.enrollInCourse(course2);
        teacher.evaluateTheStudent(archive,course2,student1,new Mark(2));
        teacher.evaluateTheStudent(archive,course2,student2,new Mark(2));
        teacher.evaluateTheStudent(archive,course2,student3,new Mark(4));
        teacher.removeCourse(course2);

        archive.printArchive();
    }
}

abstract class Person{
    protected String name;
    Person(String name){
        this.name = name;
    }

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

interface TeacherInterface{
    void addCourse(Course course);
    void announceCourseRegistration(Course course);
    void removeCourse(Course course);
    void evaluateTheStudent(Archive archive,Course course, Student student,
Mark mark);
    ArrayList<Course> getCourses();
}

class Teacher extends Person implements TeacherInterface{

    private ArrayList<Course> courses;

    public Teacher(String name) {
        super(name);
        this.courses = new ArrayList<>();
    }
    @Override
    public ArrayList<Course> getCourses() {
        return this.courses;
    }
    @Override
    public void addCourse(Course course){
        course.setTeacher(this);
        this.courses.add(course);
    }
    @Override
    public void announceCourseRegistration(Course course) {
        System.out.println(this.name + " объявил(а) запись на курс " +
course.getCourseName()+".");
        course.openRegistration();
    }
    @Override
    public void removeCourse(Course course){
        this.courses.remove(course);
```

```
        System.out.println(this.getName() + " завершил(а) курс  " +
course.getCourseName() + ".");
        course.closeRegistration();
    }

    @Override
    public void evaluateTheStudent(Archive archive,Course course, Student
student, Mark mark){
        System.out.println(this.name + " выставил(а) оценку " +
mark.getMark() + " студенту " + student.getName() + " за курс " +
course.getCourseName() + ".");
        student.removeCourse(course);
        archive.saveMark(course, student, mark);
    }
}

interface StudentInterface{
     void enrollInCourse(Course course);
     String getGroup();
     ArrayList<Course> getCourses();
     void setGroup(String group);
     void removeCourse(Course course);
}
class Student extends Person implements StudentInterface{
    private String group;
    private ArrayList<Course> courses;

    public Student(String name, String group) {
        super(name);
        this.group = group;
        this.courses = new ArrayList<>();
    }
    @Override
    public void enrollInCourse(Course course) {
        if (course.isRegistrationOpen()) {
            course.addStudent(this);
            this.courses.add(course);
            System.out.println("Студент " + this.name +" группы " +
this.group + " начал изучать курс " + course.getCourseName()+".");
        } else {
            System.out.println("Запись на курс " + course.getCourseName() + "
закрыта.");
        }
    }

    @Override
    public String getGroup() {
        return this.group;
    }
    @Override
    public ArrayList<Course> getCourses() {
        return this.courses;
    }
    @Override
    public void setGroup(String group) {
        this.group = group;
    }

    @Override
    public void removeCourse(Course course){
        this.courses.remove(course);
        System.out.println("Студент группы "+this.getGroup() + " " +
this.getName() + " завершил изучение курса " + course.getCourseName()+".");
    }
```

```
    @Override
    public String toString() {
        return "Student{" +
                "name='" + this.name + '\'' +
                ", group='" + this.group + '\'' +
                '}';
    }
}


interface CourseInterface {
    void openRegistration();
    void closeRegistration();
    boolean isRegistrationOpen();
    void addStudent(Student student);
    void setTeacher(Teacher teacher);
    String getTeacherName();
    List<Student> getStudents();
    String getCourseName();
}
class Course implements CourseInterface{
    private String name;
    private Teacher teacher;
    private ArrayList<Student> students;
    private boolean registrationOpen;
    public Course(String name){
        this.name = name;
        this.students = new ArrayList<>();
        this.registrationOpen = false;
    }
    @Override
    public void setTeacher(Teacher teacher) {
        this.teacher = teacher;
    }
    @Override
    public String getTeacherName() {
        return this.teacher.getName();
    }
    @Override
    public List<Student> getStudents() {
        return this.students;
    }
    @Override
    public String getCourseName() {
        return this.name;
    }
    @Override
    public void openRegistration() {
        this.registrationOpen = true;
    }
    @Override
    public void closeRegistration() {
        this.registrationOpen = false;
    }
    @Override
    public boolean isRegistrationOpen() {
        return this.registrationOpen;
    }
    @Override
    public void addStudent(Student student) {
        this.students.add(student);
    }
```

```
}

interface ArchiveInterface{
     void saveMark(Course course, Student student, Mark mark);
     void printArchive();
}
class Archive implements ArchiveInterface{
    private static Map<Course, Map<Student, Mark>> gradesArchive;

    public Archive() {
        gradesArchive = new HashMap<>();
    }

    @Override
    public void saveMark(Course course, Student student, Mark mark) {
        Map<Student, Mark> courseGrades =
gradesArchive.computeIfAbsent(course, m -> new HashMap<>());
        courseGrades.put(student, mark);
    }

    @Override
    public void printArchive(){
        StringBuilder result = new StringBuilder();
        for (Course course : gradesArchive.keySet()) {
            result.append("Course:
").append(course.getCourseName()).append(", Teacher:
").append(course.getTeacherName()).append("\n");
            Map<Student, Mark> courseGrades = gradesArchive.get(course);
            for (Student student : courseGrades.keySet()) {
                Mark grade = courseGrades.get(student);
                result.append("\t\tStudent:
").append(student.getName()).append(", Group: ").append(student.getGroup());
                result.append(", Mark:
").append(grade.getMark()).append("\n");
            }
        }
        System.out.println("\n"+result);
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (Course course : gradesArchive.keySet()) {
            result.append("Course:
").append(course.getCourseName()).append(", Teacher:
").append(course.getTeacherName()).append("\n");
            Map<Student, Mark> courseGrades = gradesArchive.get(course);
            for (Student student : courseGrades.keySet()) {
                Mark grade = courseGrades.get(student);
                result.append("\tStudent:
").append(student.getName()).append(", Group: ").append(student.getGroup())
                        .append(", Mark:
").append(grade.getMark()).append("\n");
            }
        }
        return result.toString();
    }
}

class Mark {
    private int mark;

    public Mark(int mark) {
        this.mark = mark;
```

```
    }
    public int getMark() {
        return this.mark;
    }
    public void setMark(int mark) {
        this.mark = mark;
    }

    @Override
    public String toString() {
        return "Grade{" +
                "grade=" + this.mark +
                '}';
    }
}
```

Результат работы программы:

```
"D:\Porgramming instruments\JDK2023\Java\jdk-21\bin\java.exe" "-javaagent:D:\Porgramming instruments\IntelliJ IDEA Community
Гречаник Татьяна Викторовна объявил(а) запись на курс Компьютерные системы и сети.
Студент Бубен Станислав Олеговочи группы ПО-8 начал изучать курс Компьютерные системы и сети.
Студент Бувин Дмитрий Александрович группы ПО-8 начал изучать курс Компьютерные системы и сети.
Студент Назаров Кирилл Викторович группы ПО-8 начал изучать курс Компьютерные системы и сети.
Гречаник Татьяна Викторовна выставил(а) оценку 5 студенту Бубен Станислав Олеговочи за курс Компьютерные системы и сети.
Студент группы ПО-8 Бубен Станислав Олеговочи завершил изучение курса Компьютерные системы и сети
Гречаник Татьяна Викторовна выставил(а) оценку 5 студенту Бувин Дмитрий Александрович за курс Компьютерные системы и сети.
Студент группы ПО-8 Бувин Дмитрий Александрович завершил изучение курса Компьютерные системы и сети
Гречаник Татьяна Викторовна выставил(а) оценку 4 студенту Назаров Кирилл Викторович за курс Компьютерные системы и сети.
Студент группы ПО-8 Назаров Кирилл Викторович завершил изучение курса Компьютерные системы и сети
Гречаник Татьяна Викторовна завершил(а) курс  Компьютерные системы и сети.

Гречаник Татьяна Викторовна объявил(а) запись на курс Проектирование интернет систем.
Студент Бубен Станислав Олеговочи группы ПО-8 начал изучать курс Проектирование интернет систем.
Студент Бувин Дмитрий Александрович группы ПО-8 начал изучать курс Проектирование интернет систем.
Студент Назаров Кирилл Викторович группы ПО-8 начал изучать курс Проектирование интернет систем.
Гречаник Татьяна Викторовна выставил(а) оценку 2 студенту Бубен Станислав Олеговочи за курс Проектирование интернет систем.
Студент группы ПО-8 Бубен Станислав Олеговочи завершил изучение курса Проектирование интернет систем
Гречаник Татьяна Викторовна выставил(а) оценку 2 студенту Бувин Дмитрий Александрович за курс Проектирование интернет систем
Студент группы ПО-8 Бувин Дмитрий Александрович завершил изучение курса Проектирование интернет систем
Гречаник Татьяна Викторовна выставил(а) оценку 4 студенту Назаров Кирилл Викторович за курс Проектирование интернет систем.
Студент группы ПО-8 Назаров Кирилл Викторович завершил изучение курса Проектирование интернет систем

Course: Проектирование интернет систем, Teacher: Гречаник Татьяна Викторовна
        Student: Бубен Станислав Олеговочи, Group: ПО-8, Mark: 2
        Student: Бувин Дмитрий Александрович, Group: ПО-8, Mark: 2
        Student: Назаров Кирилл Викторович, Group: ПО-8, Mark: 4
Course: Компьютерные системы и сети, Teacher: Гречаник Татьяна Викторовна
        Student: Бубен Станислав Олеговочи, Group: ПО-8, Mark: 5
        Student: Бувин Дмитрий Александрович, Group: ПО-8, Mark: 5
        Student: Назаров Кирилл Викторович, Group: ПО-8, Mark: 4
```

Вывод: в ходе выполнения лабораторной работы я приобрел практические навыки в области объектно-ориентированного программирования.