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

Лабораторная работа №5
По дисциплине «СПП» за 5 семестр

Выполнил:

Студент группы ПО-3

Ковалёва А. И.

Проверил:

Крощенко А. А.

Брест 2020

Вариант 12

Цель: 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.

Текст программы:

```
package com.company;
import java.util.ArrayList;

interface Technique {
    void turnOn();
    void turnOff();
}
abstract class Player implements Technique {
    private boolean work;
    protected String model;

    @Override
    public void turnOff() {
        work = false;
    }

    @Override
    public void turnOn() {
        work = true;
    }

    public void Player() {
        work = false;
        model = " ";
    }
    public void Player(String model) {
        work=false;
        setModel(model);
    }
    public boolean getState() {
        return work;
    }
    abstract public void setModel(String model);
}

class VideoPlayer extends Player {
    private ArrayList<String> movies;

    public VideoPlayer() {
        super.Player();
        movies = new ArrayList<String>();
    }
```

```java
    public VideoPlayer(String model, ArrayList<String> movies) {
        super.Player(model);
        this.movies = movies;
    }

    @Override
    public void turnOn() {
        super.turnOn();
        System.out.println("Player is on");
    }

    @Override
    public void turnOff() {
        super.turnOff();
        System.out.println("Player is off");
    }

    public void addMovie(String movie) {
        movies.add(movie);
    }

    public void showMovies() {
        if (super.getState()) {
            System.out.println("Movie list:");
            for (String movie : movies)
                System.out.println(movie);
            System.out.println();
        } else {
            System.out.println("Player is off");
        }
    }

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

public class Main {
    public static void main(String[] args) {
        ArrayList<String> movies = new ArrayList<String>();
        movies.add("movie 1");
        movies.add("movie 2");

        VideoPlayer player = new VideoPlayer("Samsung", movies);
        player.turnOn();
        player.showMovies();

        player.addMovie("movie 3");
        player.showMovies();
        player.turnOff();
    }
}
```

Результат выполнения:

```
Player is on
Movie list:
movie 1
movie 2

Movie list:
movie 1
movie 2
movie 3

Player is off
```

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

Текст программы:

```
package com.company;

import java.util.ArrayList;

enum Cities {
    Brest,
    Minsk,
    Moscow,
    Gomel,
    London
}

enum VehicleType {
    Car,
    Plain,
    Train
}

class Route {
    private double time;
    private Cities city;
    private int cost;

    Route(Cities city, int cost, double time) {
        this.city = city;
        this.cost = cost;
        this.time = time;
    }

    void show() {
        System.out.println("Route to " + city);
        System.out.println("Time: " + time + " Cost: " + cost);
    }

    int getCost() {
        return cost;
    }

    Cities getCity() {
        return city;
    }

    double getTime() {
        return time;
    }
}

abstract class Vehicle {
    private String model;
    private VehicleType type;
    private ArrayList<Route> routes;

    Vehicle(String model, VehicleType type) {
        this.model = model;
        this.type = type;
        routes = new ArrayList<Route>();
    }

    void calculate(double length) { }

    void show() {
        boolean hasRoutes = !routes.isEmpty();
        System.out.println("Vehicle: " + type + " .Model: " + model);
        if (!hasRoutes) {
            System.out.println("No routes");
        } else {
            for (Route route : routes) {
                route.show();
            }
        }
        System.out.println();
    }

    void findRouteTo(Cities city) {
        System.out.println("Routes to " + city);
        System.out.println("Vehicle (" + type + ") :" + model);
        for (Route route : routes) {
            if (route.getCity() == city) {
                route.show();
            }
        }
        System.out.println();
    }

    boolean hasRouteTo(Cities city) {
        for (Route route : routes) {
            if (route.getCity() == city) {
                return true;
            }
        }
        return false;
    }

    void addNewWay(Route route) {
        routes.add(route);
    }

    VehicleType getType() {
        return type;
    }

    String getModel() {
        return model;
    }
}

class Container {
    ArrayList<Vehicle> vehicles = new ArrayList<Vehicle>();

    void add(Vehicle vehicle) {
        vehicles.add(vehicle);
    }

    void searchVehicleTo(Cities city) {
        boolean hasRoute = false;
        System.out.println("Vehicles to " + city);
        for (Vehicle vehicle : vehicles)
            if (vehicle.hasRouteTo(city)) {
                vehicle.findRouteTo(city);
                hasRoute = true;
            }
        if (hasRoute == false)
            System.out.println("No one goes to " + city);
    }

    void search(double length) {
        boolean t = false;
        for (Vehicle obj : vehicles)
            if (obj.getType() == VehicleType.Car) {
                obj.calculate(length);
                t = true;
            }
        if (t == false)
            System.out.println("No vehicles");
    }

    void show() {
        for (Vehicle vehicle : vehicles) {
            vehicle.show();
        }
    }
}

class Plane extends Vehicle {
    Plane(String model) {
        super(model, VehicleType.Plain);
    }
    @Override
    void show() {
        super.show();
    }
}

class Train extends Vehicle {
    Train(String model) {
        super(model, VehicleType.Train);
    }

    @Override
    void show() {
        super.show();
    }
}

class Car extends Vehicle {
    private double speed;
    private double price;

    @Override
    public void show() {
        super.show();
        System.out.println("Price for 100 km/h:" + price);
    }

    public Car(String model, double speed, double price) {
        super(model, VehicleType.Car);
        this.speed = speed;
        this.price = price;
    }

    @Override
    public void calculate(double length) {
        double time = length / speed;
        double cost = length * price / 100;
        System.out.println("Vehicle (" + this.getType() + ") :" +
this.getModel());
        System.out.print("Price: ");
        System.out.printf(String.format("%.2f \n", cost).replace(",",
"."));
        System.out.print("Time: ");
        System.out.printf(String.format("%.2f \n", time).replace(",",
"."));
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        Container container = new Container();
        Route route1 = new Route(Cities.Moscow, 345, 1.2);
        Route route2 = new Route(Cities.London, 400, 3.4);
        Route route3 = new Route(Cities.Minsk, 120, 0.2);
        Route route4 = new Route(Cities.Brest, 230, 2);
        Route route5 = new Route(Cities.Gomel, 500, 1);
        Plane air1 = new Plane("AAA");
        Plane air2 = new Plane("BBB");
        Train train1 = new Train("Express");
        Car car1 = new Car("bmw", 90, 100);
        Car car2 = new Car("reno", 70, 40);

        container.add(air1);
        container.add(air2);
        container.add(car1);
        container.add(car2);
        container.add(train1);

        air1.addNewWay(route1);
        air1.addNewWay(route2);
        air2.addNewWay(route3);
        air2.addNewWay(route4);
        air2.addNewWay(route5);
        car2.addNewWay(route3);
        train1.addNewWay(route5);

        System.out.println("Search vehicles to Moscow");
        System.out.println("------------------");
        container.searchVehicleTo(Cities.Moscow);
        System.out.println("------------------");
        System.out.println("Calculate pricae for route 500 km by car");
        System.out.println("------------------");
        container.search(500);
        System.out.println("------------------");
        System.out.println("Show all");
        System.out.println("------------------");
        container.show();
    }
}
```

Результат выполнения:

```
Search vehicles to Moscow
------------------
Vehicles to Moscow
Routes to Moscow
Vehicle (Plain) :AAA
Route to Moscow
Time: 1.2 Cost: 345

------------------
Calculate pricae for route 500 km by car
------------------
Vehicle (Car) :bmw
Price: 500.00
Time: 5.56

Vehicle (Car) :reno
Price: 200.00
```

```
Time: 7.14

------------------
Show all
------------------
Vehicle: Plain .Model: AAA
Route to Moscow
Time: 1.2 Cost: 345
Route to London
Time: 3.4 Cost: 400

Vehicle: Plain .Model: BBB
Route to Minsk
Time: 0.2 Cost: 120
Route to Brest
Time: 2.0 Cost: 230
Route to Gomel
Time: 1.0 Cost: 500

Vehicle: Car .Model: bmw
No routes

Price for 100 km/h:100.0
Vehicle: Car .Model: reno
Route to Minsk
Time: 0.2 Cost: 120

Price for 100 km/h:40.0
Vehicle: Train .Model: Express
Route to Gomel
Time: 1.0 Cost: 500
```

Задание 3:

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.

Текст программы:

```
package com.company; import java.util.ArrayList;

interface Do {
    void work(Course obj);
}

class Archive {
    Optional.Student student;
    int mark;
    Archive(Optional.Student student,int mark)  {
        this.student = student;
        this.mark = mark;
    }
    void show() {
        student.show();
        System.out.println("Mark: " + mark);
        System.out.println();
```

```java
    }
}

class Optional {
    private ArrayList<Archive> arh;
    private ArrayList<Course> arr;

    Optional() {
        arh = new ArrayList<Archive>();
        arr = new ArrayList<Course>();
    }

    void add(Course obj) {
        arr.add(obj);
    }

    void showCourse() {
        for (Course course: arr) {
            course.show();
        }
    }

    void showArchive() {
        for (Archive archive: arh) {
            archive.show();
        }
    }

    class Teacher extends Person implements Do {
        Teacher(String name) {
            super(name);
        }

        void addCourse(Course course) {
            work(course);
        }

        public void work(Course obj) {
            add(obj);
        }

        void setMark(Student student, int mark) {
            if (student.isLearning()) {
                student.setMark(mark);
                Archive archive = new Archive(student, mark);
                arh.add(archive);
            }
        }
    }

    class Student extends Person implements Do {
        int mark;
        Course studCourse;
        boolean learn;

        Student(String name) {
            super(name);
            learn = false;
        }
        public void work(Course obj) {
            if (arr.contains(obj)) {
                studCourse = obj;
                System.out.println("Added course");
                learn = true;
            } else {
                System.out.println("No such course");
            }
        }

        boolean isLearning() {
            return learn;
        }

        void setMark(int mark) {
            this.mark = mark;
        }

        void show() {
            System.out.print("Student name:");
            super.show();
            studCourse.show();
        }
    }
}

abstract class Person {
    String name;
    Person() {
        name = "";
    }
    Person(String name) {
        this.name=name;
    }
    void show() {
        System.out.println(name);
    }
}

class Course {
    int num;
    String title;
    Course() {
        num = 0;
        title = "";
    }
    Course(String title, int num) {
        this.title=title;
        this.num=num;
    }
    void show() {
        System.out.println("Course num: " + num);
        System.out.println("Course name: " + title);
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        Optional class1 = new Optional();

        Optional.Teacher teacher1= class1.new Teacher("Alex");
        Optional.Teacher teacher2= class1.new Teacher("Kate");

        Course mathCourse = new Course("Math",1);
        Course historyCourse = new Course("History",2);
        Course englishCourse = new Course("English",3);

        teacher1.addCourse(mathCourse);
        teacher2.addCourse(historyCourse);
        teacher2.addCourse(englishCourse);

        Optional.Student student1 = class1.new Student("Liz");
        Optional.Student student2 = class1.new Student("Nastya");

        student2.work(englishCourse);
        student1.work(historyCourse);

        teacher2.setMark(student1,9);
        teacher1.setMark(student2,7);

        class1.showCourse();
        System.out.println("-----------------");
        class1.showArchive();
    }
}
```

Результат выполнения:

```
Added course
Added course
Course num: 1
Course name: Math

Course num: 2
Course name: History

Course num: 3
Course name: English

-----------------
Student name:Liz
Course num: 2
Course name: History

Mark: 9
```

Student name:Nastya
Course num: 3
Course name: English

Mark: 7

Вывод: В ходе выполненной работы приобрела практические навыки в области объектно-ориентированного проектирования.