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

Лабораторная работа №5

По дисциплине: «Современные платформы программирования»

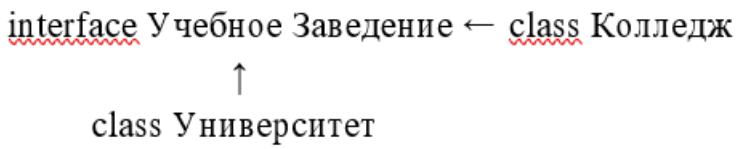
Выполнил:
студент 3 курса
группы ПО-8
Таразевич Н.А.

Проверил:
Крощенко А.А.

Брест, 2024

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:



Код программы:

```
public interface EducationalInstitution {
    String getAddress();

    void conductAdmissions();
    void conductGraduation();

}


public class College implements EducationalInstitution {
    private String address;

    private String name;


    public College(String name, String address){
        this.name = name;

        this.address = address;
    }


    @Override

    public String getAddress() {
        return address;

    }


    @Override

    public void conductAdmissions() {
        System.out.println("Conduct admissions in college - "

                           + name + " by the address " + address);
    }


    @Override

    public void conductGraduation() {
        System.out.println("Conduct graduation in college - "

                          + name + " by the address " + address);
    }

}
```

```java
public class University implements EducationalInstitution{
    private String address;

    private String name;


    public University(String name, String address){
        this.name = name;

        this.address = address;
    }


    @Override

    public String getAddress() {
        return address;

    }


    @Override

    public void conductAdmissions() {
        System.out.println("Conduct admissions in university - "

                           + name + " by the address " + address);
    }


    @Override

    public void conductGraduation() {
        System.out.println("Conduct graduation in university - "

                           + name + " by the address " + address);
    }

}
```

Входные данные:

```java
public class EducationalInstitutionTest {
    public static void main(String[] args) {

        List<EducationalInstitution> educationalInstitutions = new ArrayList<>();
        educationalInstitutions.add(new College("MuchosranksiyColledg",

                "Muchosranks, yl. Pushkina, dom Kolatychkina "));
        educationalInstitutions.add(new University("KrutoiUniversitet",

                "Paris, pod EefelevoiBachnei"));


        for (EducationalInstitution educationalInstitution : educationalInstitutions){
            educationalInstitution.conductAdmissions();

        }
    }

}
```

Результат работы программы:

```
D:\SDK\JDK\bin\java.exe "-javaagent:D:\JetBrains\IntelliJ IDEA 2023.3.4\lib\idea_rt.jar=51548:D:\JetBrains\Inte
Conduct admissions in college - MuchosranksiyColledg by the address Muchosranks, yl. Pushkina, dom Kolatychkina
Conduct admissions in university - KrutoiUniversitet by the address Paris, pod EefelevoiBachnei
```

Задание 2: требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов.

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

Код программы:

```
public class Position {

    private float xPosition;
    private float yPosition;


    Position(float xPosition, float yPosition){
        this.xPosition = xPosition;
        this.yPosition = yPosition;

    }

    public static double getDistanceBetweenPositions(Position loc1, Position loc2){
        return Math.sqrt(Math.pow(loc1.xPosition - loc2.xPosition, 2) +

                         Math.pow(loc1.yPosition - loc2.yPosition, 2));
    }

    …
}
```

```
public interface Positionable {
    Position getPosition();

    void setPosition(Position position);
}
```

```
public class City implements Positionable {
    private Position position;

    private String name;


    public City(String name, Position position){
        this.name = name;

        this.position = position;
    }

    @Override

    public Position getPosition() {
        return position;
```

```
    }
    @Override

    public void setPosition(Position position) {
        this.position = position;

    }
    …

}
```

```
public abstract class CargoCarrier implements Positionable {
    private static int nextPersonId = 1;

    private String name;
    private final int id;

    private Position currentPosition = new Position(0, 0);


    public CargoCarrier(String name){
        this.name = name;

        id = nextPersonId++; }

    @Override

    public Position getPosition(){
        return currentPosition;

    } @Override

    public void setPosition(Position position){
        currentPosition = position;

    }

    public abstract double getTransportationTime(Positionable positionable);
    public abstract double getTransportationCost(Positionable positionable); …

}



public class Car extends CargoCarrier {
    private double speed;

    private double costPerKm;


    public Car(String name, double speed, double costPerKm) {
        super(name);

        this.speed = speed;
        this.costPerKm = costPerKm;

    } @Override

    public double getTransportationTime(Positionable positionable) {

        return Position.getDistanceBetweenPositions(positionable.getPosition(),
                                                    this.getPosition()) / speed;

    } @Override

    public double getTransportationCost(Positionable positionable) {

        return Position.getDistanceBetweenPositions(positionable.getPosition(),
                                                    this.getPosition()) * costPerKm;

    } }

…
```

Входные данные:

```
public class CargoCarrierTest {

    public static void main(String[] args) {
        ArrayList<CargoCarrier> cargoCarriers = new ArrayList<>();
```

```
        cargoCarriers.add(new Car("KrasivayaMachina", 50, 5));
        cargoCarriers.add(new Train("BistriyPoesd", 100, 1));
        cargoCarriers.add(new Plane("MedleniySamolët", 500, 10));


        City cityBrest = new City("Brest", new Position(500, 0));
        System.out.println(cityBrest.getName() + " " + cityBrest.getPosition() + ":");
        for (CargoCarrier cargoCarrier : cargoCarriers) {

            System.out.println("\t" + cargoCarrier.getName() + " "
                               + cargoCarrier.getPosition() + ":");

            System.out.println("\t\t transportation time: "

                               + cargoCarrier.getTransportationTime(cityBrest));
            System.out.println("\t\t transportation cost: "

                               + cargoCarrier.getTransportationCost(cityBrest));
        }

    }
}
```

Результат работы программы:

```
D:\SDK\JDK\bin\java.exe "-javaagent:D:\JetBrains\Intel
Brest {xPosition= 500.0, yPosition= 0.0}:
    KrasivayaMachina {xPosition= 0.0, yPosition= 0.0}:
        transportation time: 10.0
        transportation cost: 2500.0
    BistriyPoesd {xPosition= 0.0, yPosition= 0.0}:
        transportation time: 5.0
        transportation cost: 500.0
    MedleniySamolët {xPosition= 0.0, yPosition= 0.0}:
        transportation time: 1.0
        transportation cost: 5000.0
```

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

Код программы:

```
abstract public class Person {
    private String name;
    private final int id;

    private static int nextPersonId = 1;


    public Person(String name){
```

```
        this.name = name;

        id = nextPersonId++;
    }

}


public class Enrollee extends Person{
    public Enrollee(String name) {

        super(name);
    }

}


public class Teacher extends Person{
    public Teacher(String name) {

        super(name);
    }

    public int getMark(){

        return (int) (Math.random() * 100);
    }

}


public class Faculty {
    private String name;

    private List<Subject> requiredExams = new ArrayList<>();

    private List<EnrolleeData> registeredEnrolles = new ArrayList<>();
    private List<EnrolleeData> evolvedEnrolles = new ArrayList<>();


    public static class EnrolleeData {
        private Enrollee enrollee;

        private Map<Subject, Integer> examMarks = new HashMap<>();


        EnrolleeData(Enrollee enrollee){
            this.enrollee = enrollee;

        }

        public void setExamMark(Subject subject, int mark){
            if (!examMarks.containsKey(subject))

                examMarks.put(subject, mark);
        }

        public int getExamScore(){
            int examScore = 0;

            for (int examMark : examMarks.values()){
                examScore += examMark;

            }

            return examScore / examMarks.size();
        }

    }
}
```

```
 Faculty(String name, List<Subject> requiredExams){
        this.name = name;

        this.requiredExams.addAll(requiredExams.stream().distinct().toList()); }


    public Faculty registerEnrollee(Enrollee enrollee){ if
        (!registeredEnrolles.contains(enrollee)){

            EnrolleeData enrolleeData = new EnrolleeData(enrollee);
            registeredEnrolles.add(enrolleeData);

        }

        return this; }


    public Faculty conductExam(Subject subject, Teacher teacher){ if
        (requiredExams.contains(subject))

            for (EnrolleeData enrolleeData : registeredEnrolles) if
                (!enrolleeData.havePassedExam(subject))

                    enrolleeData.setExamMark(subject, teacher.getMark());
        return this;

    }


    public void evolveFromEnrolleeToStudent(int passingScore){
        for (int i = 0; i < registeredEnrolles.size(); i++) {

            EnrolleeData enrolleeData = registeredEnrolles.get(i);

            if (enrolleeData.getExamMarks().size() == requiredExams.size() &&
                    enrolleeData.getExamScore() >= passingScore){

                registeredEnrolles.remove(i--);
                evolvedEnrolles.add(enrolleeData);

            } }

    } }
```

Входные данные:

```
public static void main(String[] args) {
    String facultyName = "CreativnoyeImya";

    List<Subject> requiredExams = new ArrayList<>(List.of(Subject.HISTORY,
Subject.MATHEMATICS));

    Faculty faculty = new Faculty(facultyName, requiredExams);


    faculty.registerEnrollee(new Enrollee("Petya"))
           .registerEnrollee(new Enrollee("Kirill"));


    Teacher teacher = new Teacher("Prepodavatel");


    faculty.conductExam(Subject.HISTORY, teacher)
           .conductExam(Subject.MATHEMATICS, teacher);


    System.out.println("All enrolles:");

    for (Faculty.EnrolleeData enrolleeData : faculty.getRegisteredEnrolles()) {
```

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

        faculty.evolveFromEnrolleeToStudent(70);

        System.out.println("\nEnrolled people:");
        for (Faculty.EnrolleeData enrolleeData : faculty.getEvolvedEnrolles()) {
            System.out.println(enrolleeData);
    } }
```

Результат работы программы:

```
D:\SDK\JDK\bin\java.exe "-javaagent
All enrolles:
enrollee:
    id=1
    name='Petya'
    exams:
        история - 59
        математика - 5
enrollee:
    id=2
    name='Kirill'
    exams:
        история - 60
        математика - 82

Enrolled people:
enrollee:
    id=2
    name='Kirill'
    exams:
        история - 60
        математика - 82

Process finished with exit code 0
```

Вывод: я приобрёл практические навыки в области объектно-ориентированного проектирования