**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет   
імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра інженерії програмного забезпечення**

**ЗВІТ**

*з лабораторної роботи № 2*

**«Класи та перерахування»**

**Варіант № 15**

Дисципліна «Об'єктно-орієнтоване програмування»

Спеціальність «Інженерія програмного забезпечення»

121–ЛР.01–209.22210929

***Здобувачка*** *\_\_\_\_\_\_\_\_\_\_****О. К.******Шумакова***

*(підпис)*

*\_\_17.01.2024 \_\_*

*(дата)*

***Викладач*** *\_\_\_\_\_\_\_\_\_\_\_\_\_****Г. В.******Горбань***

*(підпис)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Миколаїв – 2024**

**Лабораторна робота №2**

**Тема роботи:**Керуючі конструкції, масиви.

**Завдання 1:**

Визначити клас *Point*з двома приватними полями *x, y*.

**Завдання 2:**

Визначити клас *Rhombus*, який містить:

* Чотири поля *p1, p2, p3* і *p4* типу *Point*.
* Конструктор без аргументів, який створює ромб зі значеннями за замовчуванням для точок.
* Конструктор, який створює ромб із зазначеними точками.
* Метод *getArea*(), який обчислює та повертає площу ромба.
* Метод *getPerimeter*(), який обчислює та повертає периметр ромба.
* Метод *contains(double x, double y)*, який повертає *true*, якщо вказана точка *(x, y)* знаходиться всередині поточного ромба.
* Метод *contains(Rhombus r)*, який повертає *true*, якщо вказаний ромб повністю знаходиться всередині поточного ромба.

**Завдання 3:**

Ініціалізувати у *main*() декілька об’єктів типу *Rhombus*. Реалізувати метод для підрахунку суми площин ромбів. Метод повинен приймати будь-яку кількість параметрів.

**Лістинг коду 1:**

**Point.Java**

public class Point {

private double x;

private double y;

public Point(double x, double y) {

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

}

**Rhombus.Java**

public class Rhombus {

private Point p1, p2, p3, p4;

public Rhombus() {

this.p1 = new Point(0, 0);

this.p2 = new Point(1, 1);

this.p3 = new Point(2, 0);

this.p4 = new Point(1, -1);

}

public Rhombus(Point p1, Point p2, Point p3, Point p4) {

if (isRhombus(p1, p2, p3, p4)) {

this.p1 = p1;

this.p2 = p2;

this.p3 = p3;

this.p4 = p4;

} else {

throw new IllegalArgumentException(

"The provided points do not form a rhombus."

);

}

}

private double distance(Point p1, Point p2) {

return Math.sqrt(

Math.pow(p2.getX() - p1.getX(), 2) + Math.pow(p2.getY() - p1.getY(), 2)

);

}

private boolean isRhombus(Point p1, Point p2, Point p3, Point p4) {

double d1 = distance(p1, p2);

double d2 = distance(p2, p3);

double d3 = distance(p3, p4);

double d4 = distance(p4, p1);

return d1 == d2 && d2 == d3 && d3 == d4;

}

public double getArea() {

double d1 = distance(p1, p3);

double d2 = distance(p2, p4);

return 0.5 \* d1 \* d2;

}

public double getPerimeter() {

return 2 \* (distance(p1, p2) + distance(p2, p3));

}

public boolean contains(double x, double y) {

double minX = Math.min(

Math.min(p1.getX(), p2.getX()),

Math.min(p3.getX(), p4.getX())

);

double maxX = Math.max(

Math.max(p1.getX(), p2.getX()),

Math.max(p3.getX(), p4.getX())

);

double minY = Math.min(

Math.min(p1.getY(), p2.getY()),

Math.min(p3.getY(), p4.getY())

);

double maxY = Math.max(

Math.max(p1.getY(), p2.getY()),

Math.max(p3.getY(), p4.getY())

);

return x >= minX && x <= maxX && y >= minY && y <= maxY;

}

public boolean contains(Rhombus r) {

return (

contains(r.p1.getX(), r.p1.getY()) &&

contains(r.p2.getX(), r.p2.getY()) &&

contains(r.p3.getX(), r.p3.getY()) &&

contains(r.p4.getX(), r.p4.getY())

);

}

}

**Main.Java**

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

Rhombus r1 = new Rhombus();

Rhombus r2 = new Rhombus(

new Point(0, 0),

new Point(1, 2),

new Point(2, 0),

new Point(1, -2)

);

Rhombus r3 = new Rhombus(

new Point(-1, -1),

new Point(1, -1),

new Point(1, 1),

new Point(-1, 1)

);

System.out.println("Area of r1: \n" + r1.getArea());

System.out.println("Area of r2: \n" + r2.getArea());

System.out.println("Area of r3: \n" + r3.getArea());

System.out.println("Perimeter of r1: \n" + r1.getPerimeter());

System.out.println("Perimeter of r2: \n" + r2.getPerimeter());

System.out.println("Perimeter of r3: \t" + r3.getPerimeter());

System.out.println("r1 contains r2: \n" + r1.contains(r2));

System.out.println("r1 contains r3: \n" + r1.contains(r3));

System.out.println(

"r1 contains point (0.5, 0.5): " + r1.contains(0.5, 0.5)

);

System.out.println("Total area: \n" + totalArea(r1, r2, r3));

}

public static double totalArea(Rhombus... rhombuses) {

return Arrays.stream(rhombuses).mapToDouble(Rhombus::getArea).sum();

}

}

**Результати роботи програми:**

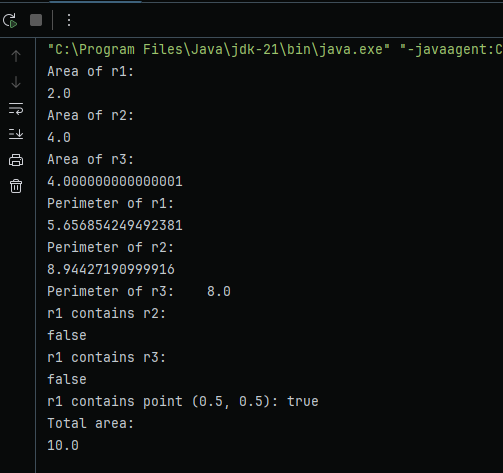


Рисунок 1

**Завдання 4:**

1. Згідно зі своїм варіантом, створити перерахування. В залежності від варіанту, це може бути вид, тип, різновид, статус, категорія, жанр тощо.
2. Додати в створене перерахування поле, геттер до нього та конструктор з параметром. Як поле можна задати наступне: ціна, рік, унікальний ідентифікатор, середнє значення певної властивості або значення за замовчуванням тощо.
3. Ініціалізувати масив об'єктів типу створеного перерахування. Вивести на консоль значення полів для кожного елементу масиву.

**Лістинг коду 2:**

**Aircraft.java**

public enum Aircraft {

AIRBUS("Airbus", 200000000),

BOEING("Boeing", 250000000),

CESSNA("Coocoorooznic", 300000);

private String name;

private int price;

Aircraft(String name, int price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public int getPrice() {

return price;

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Aircraft[] aircrafts = Aircraft.values();

for (Aircraft aircraft : aircrafts) {

System.out.println(

"Назва: " + aircraft.getName() + ", Ціна: " + aircraft.getPrice()

);

}

}

}

**Результати роботи програми:**

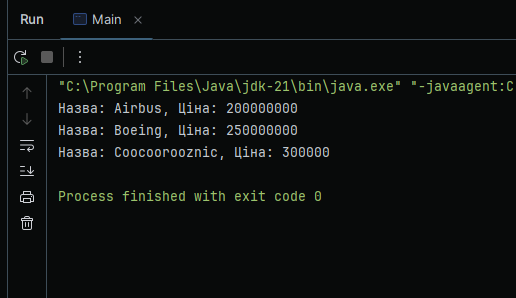


Рисунок 2

**Теоретичні питання для захисту:**

1. Які види відношень можуть існувати між класами?
2. Для чого можна використати специфікатор *final*?
3. В чому різниця між перевизначенням та перевантаженням методів?
4. Які модифікатори доступу елементів класу використовуються в Java? Пояснити різницю між ними.
5. Чи може статичний метод викликати нестатичний? Чи може нестатичний метод викликати статичний?
6. Чи можна викликати конструктор суперкласу з будь-якого методу підкласу? Чи можна викликати конструктор суперкласу з конструктора підкласу?
7. Чим перерахування відрізняються від інших типів? Чи може перерахування реалізовувати інтерфейс? Чи може перерахування розширювати інше перерахування?