Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

ЛАБОРАТОРНА РОБОТА №9

з дисципліни «Основи програмування - 2»

Тема: «Лямбда-вирази та компараториння»

Виконали:
студенти групи ІА-31
Клим’юк В.Л, Самелюк А.С,
Дук М.Д, Сакун Д.С

Перевірив:
асистент кафедри ІСТ
Степанов А. С.

Київ-2024

Тема: Лямбда-вирази та компаратори

Мета: Ця тема має на меті розглянути лямбда-вирази та компаратори в контексті програмування. Під час дослідження ми розглянемо сутність та призначення лямбда-виразів, їхню синтаксичну структуру та можливості в різних програмних мовах. Додатково, ми вивчимо компаратори - інструменти порівняння об'єктів у мовах програмування. Після завершення цього дослідження учасники матимуть глибше розуміння того, як використовувати лямбда-вирази та компаратори для полегшення програмного кодування та оптимізації роботи програм.

Хід роботи

1. Пригадати як використовувати лямбда-вирази та компаратори.
2. Проаналізувати класи свого варіанту з л/р №8 першого семестру, та виділити що найменше дві властивості, за якими можна сортувати об’єкти цих класів (наприклад, для класу Людина це може бути ім’я та прізвище, для класу Документ – назва та дата створення, …). Після цього:

2.1) за допомогою лямбда-виразів створити компаратор для сортування за однією з цих двох ознак в порядку зростання;

2.2) за допомогою дефолтного метода Comparator.reversed() створити компаратор для сортування за обраною ознакою у зворотному порядку;

2.3) за допомогою дефолтного метода Comparator.thenComparing() створити компаратор, який буде порівнювати об’єкти за однією ознакою, а у разі коли вони співпадають порівнювати за іншою ознакою;

2.4) за допомогою статичних методів Comparator.nullsFirst() або Comparator.nullsLast() створити компаратор, який дозволить порівнювати null посилання на об’єкти з іншими об’єктами.

3. Продемонструвати використання усіх створених компараторів (відсортувати масив об’єктів та/або зберегти об’єкти у TreeSet).

```
import java.util.*;

class Color {
    private int red;
    private int green;
    private int blue;

    public Color(int red, int green, int blue) {

        if ((red < 0 || red > 255) || (green < 0 || green > 255) || (blue < 0 || blue
> 255))
            throw new IllegalArgumentException("RGB values must be in range from 0 to
255");

        this.red = red;
        this.green = green;
        this.blue = blue;
    }

    public String toString() {
        return "RGB(" + red + ", " + green + ", " + blue + ")";
    }

    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Color other = (Color) obj;
        return red == other.red && green == other.green && blue == other.blue;
    }
}


abstract class Pixel {
    protected int x;
    protected int y;
    protected Color color;

    public Pixel(int x, int y, Color color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }

    public String toString() {
        return "Pixel at (" + x + ", " + y + ") with color " + color.toString();
    }

    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
```

```
        }
        Pixel other = (Pixel) obj;
        return x == other.x && y == other.y && color.equals(other.color);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public abstract void setColor(Color color);
}


class MutablePixel extends Pixel {
    public MutablePixel(int x, int y, Color color) {
        super(x, y, color);
    }

    @Override
    public void setColor(Color color) {
        super.color = color;
    }
}

public class Main {
    public static void main(String[] args) {
        Pixel[] pixels = {
                new MutablePixel(5, 3, new Color(255, 0, 0)),
                new MutablePixel(2, 7, new Color(0, 255, 0)),
                new MutablePixel(8, 1, new Color(0, 0, 255)),
                new MutablePixel(3, 5, new Color(255, 255, 255))
        };

        //sort by x in ascending order
        Arrays.sort(pixels, (Pixel p1, Pixel p2) -> Integer.compare(p1.getX(),
p2.getX()));

        System.out.println("Sort by x-coordinates in ascending order:");
        for (Pixel pixel : pixels) {
            System.out.println(pixel);
        }

        //sort by x in reverse order
        TreeSet<Pixel> reverseXSet = new TreeSet<>(Comparator.comparing((Pixel p) ->
p.getX()).reversed());
        reverseXSet.addAll(Arrays.asList(pixels));

        System.out.println("\nSort by x coordinates in reverse order:");
        for (Pixel pixel : reverseXSet) {
            System.out.println(pixel);
        }


        //sort by x, if equality - by y
        Comparator<Pixel> comparator = Comparator.comparing((Pixel p) ->
p.getX()).thenComparing((Pixel p) -> p.getY());
        Arrays.sort(pixels, comparator);

        System.out.println("\nSort by x-coordinates, and in case of equality - by y-
coordinates:");
```

```
        for (Pixel pixel : pixels) {
            System.out.println(pixel);
        }

        //sort by x with null-references allowed
        Pixel[] pixelsWithNull = {
                null,
                new MutablePixel(5, 3, new Color(255, 0, 0)),
                null,
                new MutablePixel(2, 7, new Color(0, 255, 0)),
                new MutablePixel(8, 1, new Color(0, 0, 255)),
                null,
                new MutablePixel(3, 5, new Color(255, 255, 255)),
                null
        };
        Arrays.sort(pixelsWithNull, Comparator.nullsFirst((p1, p2) -> {
            if (p1 == null && p2 == null) {
                return 0;
            } else if (p1 == null) {
                return -1;
            } else if (p2 == null) {
                return 1;
            }
            return Integer.compare(p1.getX(), p2.getX());
        }));


        System.out.println("\nSort by x coordinates with null references allowed:");
        for (Pixel pixel : pixelsWithNull) {
            System.out.println(pixel);
        }
    }
}
```

Код 1.1

```
Sort by x-coordinates in ascending order:
Pixel at (2, 7) with color RGB(0, 255, 0)
Pixel at (3, 5) with color RGB(255, 255, 255)
Pixel at (5, 3) with color RGB(255, 0, 0)
Pixel at (8, 1) with color RGB(0, 0, 255)

Sort by x coordinates in reverse order:
Pixel at (8, 1) with color RGB(0, 0, 255)
Pixel at (5, 3) with color RGB(255, 0, 0)
Pixel at (3, 5) with color RGB(255, 255, 255)
Pixel at (2, 7) with color RGB(0, 255, 0)

Sort by x-coordinates, and in case of equality - by y-coordinates:
Pixel at (2, 7) with color RGB(0, 255, 0)
Pixel at (3, 5) with color RGB(255, 255, 255)
Pixel at (5, 3) with color RGB(255, 0, 0)
Pixel at (8, 1) with color RGB(0, 0, 255)

Sort by x coordinates with null references allowed:
null
null
null
null
Pixel at (2, 7) with color RGB(0, 255, 0)
Pixel at (3, 5) with color RGB(255, 255, 255)
Pixel at (5, 3) with color RGB(255, 0, 0)
Pixel at (8, 1) with color RGB(0, 0, 255)
```

Рисунок 1.1 – Результат роботи код

4. Відповісти на контрольні питання.

Висновки: Під час дослідження теми "Лямбда-вирази та компаратори" ми з'ясували, що лямбда-вирази є потужним інструментом у багатьох сучасних мовах програмування, дозволяючи створювати анонімні функції без необхідності їх описування окремо. Вони забезпечують зручний і компактний спосіб виразити функціональність, зокрема у випадках, коли потрібно передати функцію як аргумент у інший метод чи функцію. Крім того, ми розглянули компаратори, що є інструментами порівняння об'єктів у програмуванні.