Міністерство освіти і науки України Національний університет «Львівська політехніка»



***Звіт***

до лабораторної роботи №7

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «Параметризоване програмування»

Варіант 4

Виконав:

Ст. групи КІ-34

Демчик Н.О.

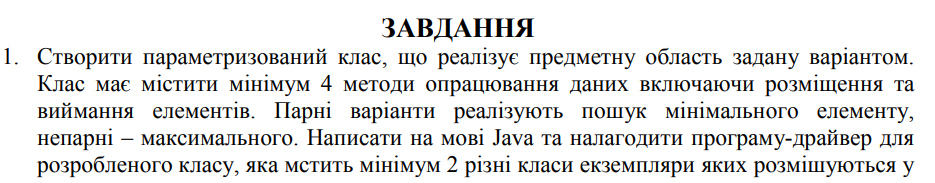
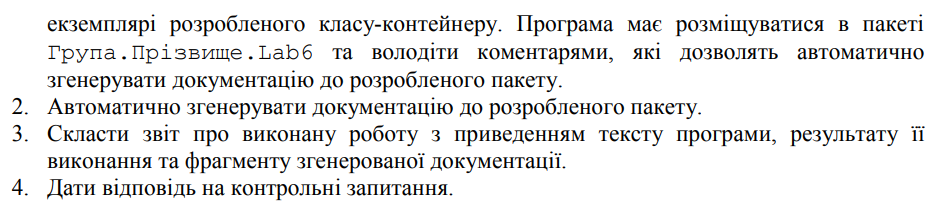
Прийняв:

к.т.н., доцент

Іванов Ю.С.

Львів 2022

**Мета***:* оволодіти навиками параметризованого програмування мовою Java

**ВАРІАНТ ЗАВДАННЯ**



**Код роботи**

*ConveyorWork.java*

import java.util.\*;

public class ConveyorWork {

    public static void main(String[] args) {

        Conveyor<? super BaggageTransporting> simpeExample = new Conveyor<BaggageTransporting>();

        simpeExample.AddData(new RibbonConveyor(23000, "Kapelou", 2500));

        simpeExample.AddData(new RibbonConveyor(25000, "Konsort", 1700));

        simpeExample.AddData(new VibrationalConveyor(14000, "Konveer", 1800, 12));

        simpeExample.AddData(new VibrationalConveyor(17000, "Konsort", 2000, 10));

        simpeExample.AddData(new RibbonConveyor(19000, "Kapelou", 3000));

        BaggageTransporting res = simpeExample.findMin();

        System.out.print("The greatest data on HDD is: \n");

        res.transport();

        res.print();

    }

}

class Conveyor<T extends BaggageTransporting> {

    private ArrayList<T> arr;

    public Conveyor() {

        arr = new ArrayList<T>();

    }

    public T findMin() {

        if (!arr.isEmpty()) {

            T min = arr.get(0);

            for (int i = 1; i < arr.size(); i++) {

                if (arr.get(i).compareTo(min) < 0)

                    min = arr.get(i);

            }

            return min;

        }

        return null;

    }

    public void AddData(T data) {

        arr.add(data);

        System.out.print("Element turned on: ");

        data.print();

    }

    public void DeleteData(int i) {

        arr.remove(i);

    }

}

interface BaggageTransporting extends Comparable<BaggageTransporting> {

    public int getAmountOfElements();

    public void transport();

    public void print();

}

class RibbonConveyor implements BaggageTransporting {

    private int elementsPerDay;

    private String producerName;

    private int length;

    RibbonConveyor(int baggage, String name, int length) {

        elementsPerDay = baggage;

        producerName = name;

        this.length = length;

    }

    public int getAmountOfElements() {

        return elementsPerDay;

    }

    public String getName() {

        return producerName;

    }

    public int getLength() {

        return length;

    }

    public void transport() {

        System.out.println("\*\*\*\*\* Start of updating this one... \*\*\*\*\*");

    }

    public void print() {

        System.out.print(": " + producerName + ", Length of conveyor: " + length +

                ", Elements per day: " + elementsPerDay + ";\n");

    }

    public int compareTo(BaggageTransporting p) {

        Integer s = elementsPerDay;

        return s.compareTo(p.getAmountOfElements());

    }

}

class VibrationalConveyor implements BaggageTransporting {

    private int elementsPerDay;

    private String producerName;

    private int length;

    private int angle;

    VibrationalConveyor(int baggage, String name, int length, int angle) {

        elementsPerDay = baggage;

        producerName = name;

        this.length = length;

        this.angle = angle;

    }

    public int getAmountOfElements() {

        return elementsPerDay;

    }

    public void setAngle(int angle) {

        this.angle = angle;

    }

    public void transport() {

        System.out.println("\*\*\*\*\* Start of updating this one... \*\*\*\*\*");

    }

    public void print() {

        System.out.print(": " + producerName + ", Length of conveyor: " + length +

                ", Elements per day: " + elementsPerDay + ", Angle: " + angle + ";\n");

    }

    public int compareTo(BaggageTransporting p) {

        Integer s = elementsPerDay;

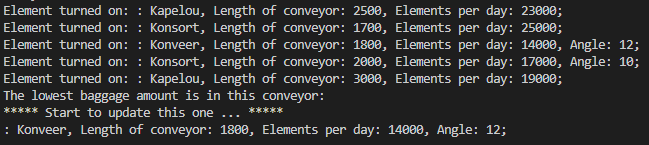
        return s.compareTo(p.getAmountOfElements());

    }

}

**Результат виконання програми**

* Консоль



**Відповіді на контрольні запитання:**

1. Параметризоване програмування є аналогом шаблонів у С++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів.
2. Параметризований клас – це клас з однією або більше змінними типу. Синтаксис оголошення параметризованого класу:

[public] class НазваКласу {

…

}

1. GenericClass <String, Integer> obj = new GenericClass<String, Integer> ();
2. (НазваКласу|НазваОб’єкту).[<Переліт типів>] НазваМетоду(параметри);
3. Модифікатори<параметризованийТип{,параметризованийТип}>типПовернення назваМетоду(параметри);
4. Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, може бути ситуація, коли метод у процесі роботи викликає з-під об’єкта параметризованого типу метод, що визначається у деякому інтерфейсі. У такому випадку немає ніякої гарантії, що цей метод буде реалізований у кожному класі, що передається через змінну типу. Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу.
5. Синтаксис оголошення параметризованого методу з обмеженнями типів:

Модифікатори <параметризований тип extends обмежуючийТип {& обмежуючий тип} {, параметризований тип extends обмежуючийТип {& обмежуючий тип} } > типПовернення назваМетоду(параметри);

1. 1. Всі класи, що утворені з одного і того ж параметризованого класу з використанням різних значень змінних типів є незалежними навіть якщо між цими типами є залежність спадкування.

2. Завжди можна перетворити параметризований клас у «сирий» клас, при роботі з яким захист від некоректного коду є значно слабшим, що дозволяє здійснювати небезпечні присвоєння об’єктів параметризованого класу об’єктам «сирого» класу. Проте у цьому випадку можна зробити помилки, які генеруватимуть виключення на етапі виконання програми.

3. Параметризовані класи можуть розширювати або реалізовувати інші

параметризовані класи. В цьому відношенні вони не відрізняються від звичайних класів.

Наприклад, ArrayList<T> реалізує інтерфейс List<T>. Це значить, що

ArrayList<SubClass> можна перетворити у List<SubClass>. Але

ArrayList<SubClass> це не ArrayList<SupClass> і не List<SupClass>, де

SubClass – підклас суперкласу SupClass.

1. – 10. Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів. Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

**Висновок***:* я оволодів навиками параметризованого програмування мовою Java.