Міністерство освіти і науки України

Національний університет «Львівська політехніка»



# Звіт

до лабораторної роботи №7

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ»

**Виконав:**

Студент групи КІ-35

Семенець О.М.

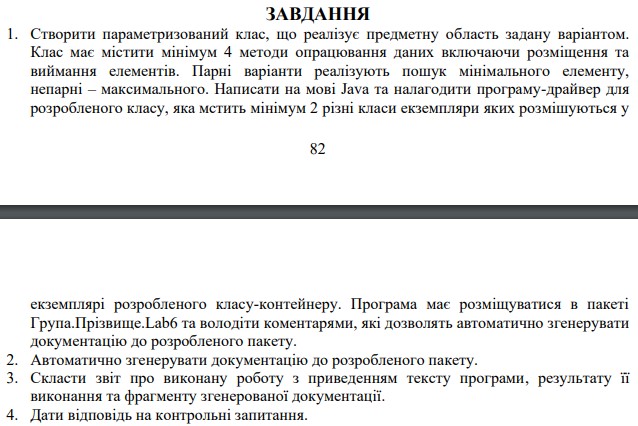
**Прийняв:**

Іванов Ю.С.

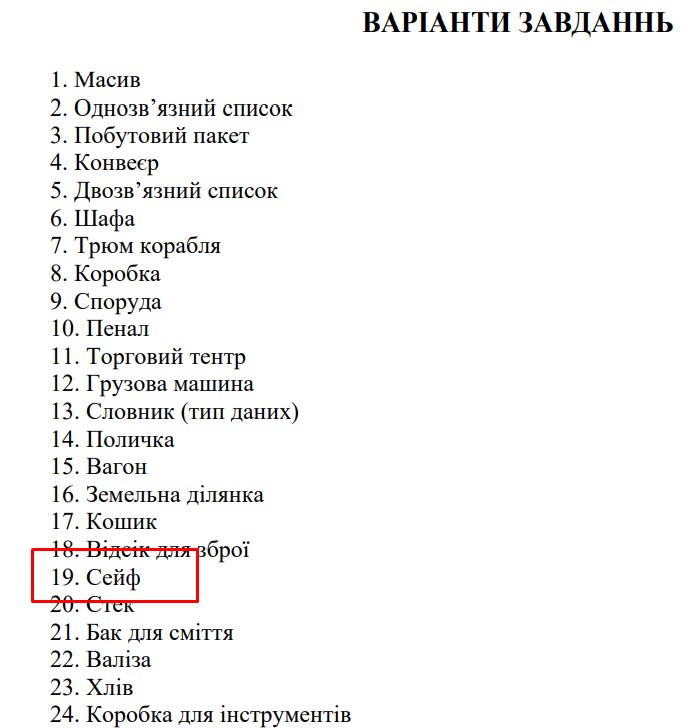
**Львів 2022**

*Мета:* оволодіти навиками параметризованого програмування мовою Java

## Виконання роботи



*Завдання:*



Код програми:

package semenets;

import semenets.SafeInfo;

public class Jewelry implements SafeInfo {

private String mark; private int Weight; private double Volume; private String Type;

public Jewelry(String mark, int Weight, double Volume, String Type) { this.mark = mark; this.Weight = Weight; this.Volume = Volume; this.Type = Type;

}

public String getMark() {

return mark;

}

public String getSafeType() {

return Type;

}

*@Override*

public int compareTo(SafeInfo o) {

Integer i = Weight;

return i.compareTo(o.Weight());

}

*@Override*

public int Weight() { return Weight;

}

*@Override*

public double Volume() { return Volume;

}

*@Override*

public void print() {

System.***out***.println(getMark() + " is safe, it's weight is -> " + Weight() + " pounds " + "this safe has volume -> " + Volume() + "sm^3.

This is safe for " + getSafeType() + ".\n" );

}

}

------------ package semenets;

import semenets.SafeInfo;

public class WeaponSafe implements SafeInfo {

private String mark; private int Weight; private double Volume; private String Type; public WeaponSafe(String mark, int Weight, double Volume, String Type) { this.mark = mark;

this.Weight = Weight; this.Volume = Volume;

this.Type = Type;

}

public String getMark() {

return mark;

}

public String getSafeType() {

return Type;

}

*@Override*

public int compareTo(SafeInfo s) {

Integer i = Weight;

return i.compareTo(s.Weight());

}

*@Override*

public int Weight() { return Weight;

}

*@Override*

public double Volume() {

return Volume;

}

*@Override*

public void print() {

System.***out***.println(getMark() + " is safe, it's weight is -> " + Weight() + " pounds " + "this safe has volume -> " + Volume() + "sm^3.

This is safe for " + getSafeType() + "." + "\n");

}

}

### ----------------

package semenets;

import java.util.ArrayList;

public class Safe<T extends SafeInfo> { private ArrayList<T> arr;

public Safe() {

arr = new ArrayList<T>();

}

//find minimal material weight public T findMinWeight() { 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 WriteData(T data) {

arr.add(data);

System.***out***.println("Element added:"); data.print();

}

public void DeleteData(int x) { arr.remove(x);

}

}

------------- package semenets;

public interface SafeInfo extends Comparable<SafeInfo> {

public int Weight(); public double Volume();

public void print();

}

---------- package semenets;

import bodyanich7.Safe; import bodyanich7.SafeInfo; import bodyanich7.WeaponSafe; import bodyanich7.Jewelry;

public class SafeAllInfo {

public static void main(String[] args) {

Safe<? super SafeInfo> safe = new Safe<SafeInfo>(); safe.WriteData(new WeaponSafe("Burgas", 55, 344.2, "For Pistols")); safe.WriteData(new WeaponSafe("Liverty", 123, 621.4, "For Rifles")); safe.WriteData(new Jewelry("Targo", 10, 100.2, "circlets")); safe.WriteData(new Jewelry("Robyr", 15, 150.0, "pendants"));

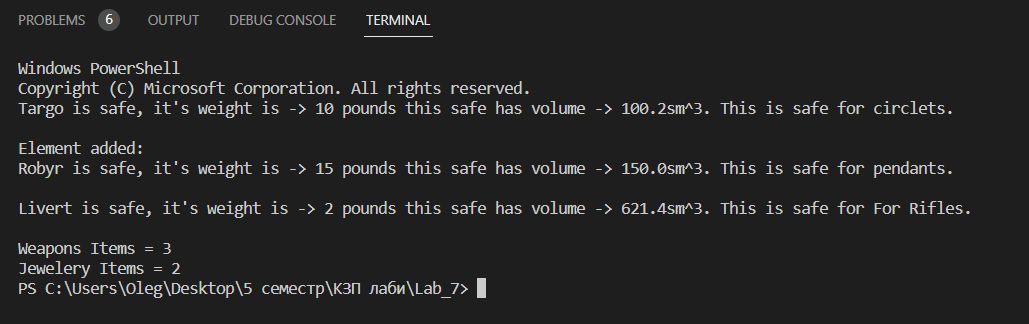
SafeInfo smallestweight = safe.findMinWeight();

System.out.print("\t\t\t\tMININMAL WEIGHT IS IN ELEMENT OF MY PROJECT: \n"); smallestweight.print();

}

}

Результат роботи програми:



Відповіді на КЗ

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}>типПове рнення 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);
6. 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.
   1. 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.
   2. 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****:* На ції лабораторній роботі я оволодів навиками параметризованого програмування мовою Java.