Міністерство освіти і науки України

Національний університет «Львівська політехніка»



Звіт

до лабораторної роботи №6

з дисципліни: “Кросплатформні засоби програмування”

на тему: «Параметризоване програмування»

Варіант №12

Виконала:

ст.гр. КІ-307

Коваль К. О.

Прийняв:

Іванов Ю. С.

**Львів 2023**

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

**Завдання:**

1. Створити параметризований клас, що реалізує предметну область задану варіантом. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.

2. Автоматично згенерувати документацію до розробленого пакету.

3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.

4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.

5. Дати відповідь на контрольні запитання:

1. Дайте визначення терміну «параметризоване програмування».

2. Розкрийте синтаксис визначення простого параметризованого класу.

3. Розкрийте синтаксис створення об’єкту параметризованого класу.

4. Розкрийте синтаксис визначення параметризованого методу.

5. Розкрийте синтаксис виклику параметризованого методу.

6. Яку роль відіграє встановлення обмежень для змінних типів?

7. Як встановити обмеження для змінних типів?

8. Розкрийте правила спадкування параметризованих типів.

9. Яке призначення підстановочних типів?

10. Застосування підстановочних типів.

**Варіант 12: Грузова машина**

**Виконання:**

package KI307.Koval.Lab6;

import java.util.\*;

/\*\*

\* Class Lab6KovalKI307 realize example program to lab6

\* **@author** Koval Kateryna

\* **@version** 1.2

\* **@since** version 1.0

\*/

public class Lab6KovalKI307 {

public static void main(String[] args) throws Exception {

Truck <? super Data> truck = new Truck <Data> ();

truck.AddData(new Sofa("Beige" , "Velvet", 120));

truck.AddData(new Chair("Green" , 18));

truck.AddData(new Bookcase(5 , 30));

truck.AddData(new Sofa("Brown" , "Leather", 85));

truck.AddData(new Chair("Black" , 12));

truck.AddData(new Bookcase(3 , 22));

truck.DeleteData(4);

Data res1 = truck.findMin();

System.***out***.println("The least by weight: ");

res1.print();

}

}

/\*\*

\* Parameterized class <code>Truck</code> implements truck

\*/

class Truck<T extends Data>{

private ArrayList<T> arr;

public Truck() {

arr = new ArrayList<T>();

}

/\*\*

\* method finds the smallest element by weight

\* **@return** min

\*/

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***.println("Element added: ");

data.print();

}

public void DeleteData(int i){

arr.remove(i);

System.***out***.println("Element #" + i + " is deleted.");

}

}

/\*\*

\* Interface <code>Data</code> implements data

\*/

interface Data extends Comparable<Data>

{

public int getWeight();

public void print();

}

/\*\*

\* Class <code>Sofa</code> implements sofa

\*/

class Sofa implements Data{

private String color;

private String material;

private int weight;

/\*\*

\* Constructor

\* **@param** <code>color\_t</code> the color of sofa

\* **@param** <code>material\_t</code> material of sofa

\* **@param** <code>weight\_t</code> the weight of sofa

\*/

public Sofa(String color\_t, String material\_t, int weight\_t){

color = color\_t;

material = material\_t;

weight = weight\_t;

}

/\*\*

\* Method return color of sofa

\* **@return** color

\*/

public String getColor(){

return color;

}

/\*\*

\* Method set color for sofa

\* **@param** <code>color\_s</code> color of sofa

\*/

public void setColor(String color\_t){

color=color\_t;

}

/\*\*

\* Method return material of sofa

\* **@return** material

\*/

public String getMaterial(){

return material;

}

/\*\*

\* Method set material for sofa

\* **@param** <code>material\_t</code> material for sofa

\*/

public void setMaterial(String material\_t){

material = material\_t;

}

/\*\*

\* Method return weight of sofa

\* **@return** weight

\*/

public int getWeight(){

return weight;

}

/\*\*

\* Method compare objects type Data

\* **@param** <code>p</code> the weight of object

\*/

public int compareTo(Data p){

Integer h = weight;

return h.compareTo(p.getWeight());

}

/\*\*

\* Method print information about sofa

\*/

public void print(){

System.***out***.println("Type: Sofa, Color: " + color + ", Material: " + material + ", Weight: " + weight + " kg\n");

}

}

/\*\*

\* class <code>Chair</code> implements Chair

\*/

class Chair implements Data{

private String chairColor;

private int chairWeight;

/\*\*

\* Constructor

\* **@param** <code>chairColor\_t</code> chair color

\* **@param** <code>chairWeight\_t</code> chair weight

\*/

public Chair(String chairColor\_t,int chairWeight\_t){

chairColor = chairColor\_t;

chairWeight=chairWeight\_t;

}

/\*\*

\* Method return color of Chair

\* **@return** chairColor

\*/

public String getChairColor(){

return chairColor;

}

/\*\*

\* Method set Chair color

\* **@param** <code>chairColor\_t</code> the color of Chair

\*/

public void setChairColor(String chairColor\_t){

chairColor=chairColor\_t;

}

/\*\*

\* Method return weight of Chair

\* **@return** chairWeight

\*/

public int getWeight(){

return chairWeight;

}

/\*\*

\* Method set Chair weight

\* **@param** <code>chairWeight\_t</code> the Chair weight

\*/

public void setWeight(int chairWeight\_t){

chairWeight = chairWeight\_t;

}

/\*\*

\* Method compare objects type Data

\* **@param** <code>p</code> the weight of object

\*/

public int compareTo(Data p){

Integer h = chairWeight;

return h.compareTo(p.getWeight());

}

/\*\*

\* Method print information about Chair

\*/

public void print(){

System.***out***.println("Type: Chair, Color: " + chairColor + ", Weight: " + chairWeight + " kg\n");

}

/\*\*

\* class <code>Chair</code> implements Chair

\*/

}

class Bookcase implements Data{

private int shelves;

private int weight;

/\*\*

\* Constructor

\* **@param** <code>shelves\_t</code> number of shelves

\* **@param** <code>weight\_t</code> Bookcase weight

\*/

public Bookcase(int shelves\_t,int weight\_t){

shelves = shelves\_t;

weight= weight\_t;

}

/\*\*

\* Method return number of Bookcase shelves

\* **@return** shelves

\*/

public int getShelves(){

return shelves;

}

/\*\*

\* Method set number of Bookcase shelves

\* **@param** <code>shelves\_t</code> the number of Bookcase shelves

\*/

public void setShelves(int shelves\_t){

shelves=shelves\_t;

}

/\*\*

\* Method return weight of Bookcase

\* **@return** Weight

\*/

public int getWeight(){

return weight;

}

/\*\*

\* Method set Bookcase weight

\* **@param** <code>weight\_t</code> the Bookcase weight

\*/

public void setWeight(int weight\_t){

weight = weight\_t;

}

/\*\*

\* Method compare objects type Data

\* **@param** <code>p</code> the weight of object

\*/

public int compareTo(Data p){

Integer h = weight;

return h.compareTo(p.getWeight());

}

/\*\*

\* Method print information about Bookcase

\*/

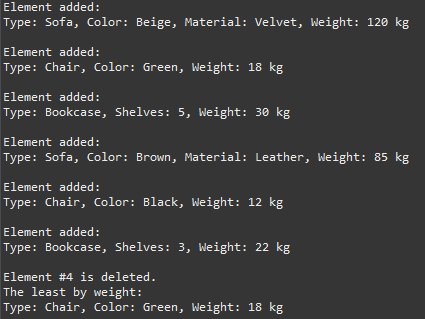
public void print(){

System.***out***.println("Type: Bookcase, Shelves: " + shelves + ", Weight: " + weight + " kg\n");

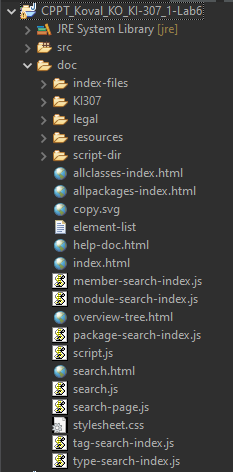
}

}

**Результат:**

****

**Документація:**

****

**Відповіді на контрольні запитання:**

1. \*\*Визначення терміну «параметризоване програмування»:\*\*

- Параметризоване програмування (або програмування з параметрами) - це підхід, при якому код може бути написаний для роботи з будь-яким типом даних. У Java це часто реалізується за допомогою параметризованих класів і методів.

2. \*\*Синтаксис визначення простого параметризованого класу:\*\*

```java

public class MyGenericClass<T> {

// тіло класу

}

```

3. \*\*Синтаксис створення об’єкту параметризованого класу:\*\*

```java

MyGenericClass<Integer> myObject = new MyGenericClass<>();

```

4. \*\*Синтаксис визначення параметризованого методу:\*\*

```java

public <T> void myGenericMethod(T parameter) {

// тіло методу

}

```

5. \*\*Синтаксис виклику параметризованого методу:\*\*

```java

myGenericMethod("Hello");

```

6. \*\*Роль встановлення обмежень для змінних типів:\*\*

- Встановлення обмежень дозволяє зменшити або визначити, які типи можуть бути використані як параметри для параметризованих класів чи методів.

7. \*\*Як встановити обмеження для змінних типів:\*\*

- Використовуючи ключове слово `extends` для обмеження типу до певного класу або інтерфейсу.

```java

public class MyGenericClass<T extends Number> {

// тіло класу

}

```

8. \*\*Правила спадкування параметризованих типів:\*\*

- Параметризований клас не може успадковуватися від параметризованого класу із відмінними параметрами.

- Параметризований клас може успадковуватися від параметризованого класу з тими самими параметрами.

9. \*\*Призначення підстановочних типів:\*\*

- Підстановочні типи використовуються для забезпечення безпеки типів під час використання параметризованих класів чи методів, дозволяючи компілятору визначати і перевіряти типи.

10. \*\*Застосування підстановочних типів:\*\*

- Забезпечення безпеки типів у колекціях (наприклад, `List<String>`).

- Забезпечення безпеки типів у параметризованих методах та конструкторах.

**Висновок:** на даній лабораторній роботі я оволоділа навиками параметризованого програмування мовою Java