Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



Звіт

з лабораторної роботи № 7

з дисципліни: «Кросплатформні засоби програмування»

на тему: «Параметризоване програмування»

**Виконав:**

студент групи КІ-35

Скіпар І.В.

**Прийняв:**

доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2022

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

**Завдання (варіант № 10)**

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 містить 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. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання:
5. Дайте визначення терміну «параметризоване програмування».
6. Як встановити обмеження для змінних типів?

**Текст програми**

PenCase.java

package KI35.Skipar.Lab7;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* Class <code>PenCase</code> implement pen-case  
 \** ***@param*** <*T*> *type of stationery  
 \*/*public class PenCase <T extends IStationery> {  
 private final ArrayList<T> stationery;  
 private static int *count*=0;  
  
 */\*\*  
 \* Default constructor  
 \*/* public PenCase() {  
 stationery = new ArrayList<T>();  
 }  
  
 */\*\*  
 \* Find element with minimal write length  
 \** ***@return*** *element with minimal write length  
 \*/* public T findMin()  
 {  
 if (!stationery.isEmpty())  
 {  
 T min = stationery.get(0);  
 for (T n : stationery) {  
 if (n.compareTo(min) < 0) min = n;  
 }  
 return min;  
 }  
 return null;  
 }  
  
 */\*\*  
 \* Add new stationery to pen-case  
 \** ***@param*** *data type of stationery  
 \*/* public void addStationery(T data) {  
 stationery.add(data);  
 System.*out*.print("Stationery added: ");  
 data.print();  
 ++*count*;  
 }  
  
 */\*\*  
 \* Delete element  
 \** ***@param*** *i index of stationery  
 \*/* public void deleteStationery(int i) {  
 stationery.remove(i);  
 --*count*;  
 }  
  
 */\*\*  
 \* Get count of stationery in pen-case  
 \** ***@return*** *count of stationery  
 \*/* public int getCount()  
 {  
 return *count*;  
 }  
}

IStationery.java

package KI35.Skipar.Lab7;  
  
*/\*\*  
 \* Interface <code>IStationery</code> implement stationery  
 \*/*public interface IStationery extends Comparable<IStationery>{  
 */\*\*  
 \* Get write length  
 \** ***@return*** *write length  
 \*/* int getWriteLength();  
  
 */\*\*  
 \* Print info  
 \*/* void print();  
}

Pen.java

package KI35.Skipar.Lab7;  
  
*/\*\*  
 \* Class <code>Pen</code> implement pen  
 \*/*public class Pen implements IStationery{  
  
 private String brand;  
 private final int writeLength;  
 private String type;  
  
 */\*\*  
 \* Constructor with parameters  
 \** ***@param*** *brand brand  
 \** ***@param*** *writeLength write length  
 \** ***@param*** *type type of pen  
 \*/* public Pen(String brand, int writeLength, String type) {  
 this.brand = brand;  
 this.writeLength = writeLength;  
 this.type = type;  
 }  
  
 */\*\*  
 \* Get brand  
 \** ***@return*** *brand  
 \*/* public String getBrand() {  
 return brand;  
 }  
  
 */\*\*  
 \* Set brand  
 \** ***@param*** *brand pen brand  
 \*/* public void setBrand(String brand) {  
 this.brand = brand;  
 }  
  
 */\*\*  
 \* Get type  
 \** ***@return*** *type  
 \*/* public String getType() {  
 return type;  
 }  
  
 */\*\*  
 \* Set type  
 \** ***@param*** *type type  
 \*/* public void setType(String type) {  
 this.type = type;  
 }  
  
 */\*\*  
 \* Get write length  
 \** ***@return*** *write length  
 \*/* @Override  
 public int getWriteLength() {  
 return writeLength;  
 }  
  
 */\*\*  
 \* Print info about pen  
 \*/* @Override  
 public void print() {  
 System.*out*.print("Pen brand: " + brand + ", Write length: " +  
 writeLength + ", Type: " + type + "\n");  
 }  
  
 */\*\*  
 \* Compare 2 pen  
 \** ***@param*** *o the object to be compared.  
 \** ***@return*** *the value 0 if this writes length is equal to the argument write length;  
 \* a value less than 0 if this writes length is numerically less than the argument writes length;  
 \* and a value greater than 0 if this writes length is numerically greater than the argument writes length  
 \*/* @Override  
 public int compareTo(IStationery o) {  
 Integer wl = writeLength;  
 return wl.compareTo(o.getWriteLength());  
 }  
}

Pencil.java

package KI35.Skipar.Lab7;  
  
*/\*\*  
 \* Class <code>Pencil</code> implement pencil  
 \*/*public class Pencil implements IStationery{  
  
 private String brand;  
 private final int writeLength;  
 private String hardness;  
  
 */\*\*  
 \* Constructor with parameters  
 \** ***@param*** *brand brand  
 \** ***@param*** *writeLength write length  
 \** ***@param*** *hardness hardness type  
 \*/* public Pencil(String brand, int writeLength, String hardness) {  
 this.brand = brand;  
 this.writeLength = writeLength;  
 this.hardness = hardness;  
 }  
  
 */\*\*  
 \* Get brand  
 \** ***@return*** *brand  
 \*/* public String getBrand() {  
 return brand;  
 }  
  
 */\*\*  
 \* Set brand  
 \** ***@param*** *brand brand  
 \*/* public void setBrand(String brand) {  
 this.brand = brand;  
 }  
  
 */\*\*  
 \* Get hardness  
 \** ***@return*** *hardness  
 \*/* public String getHardness() {  
 return hardness;  
 }  
  
 */\*\*  
 \* Set hardness  
 \** ***@param*** *hardness hardness  
 \*/* public void setHardness(String hardness) {  
 this.hardness = hardness;  
 }  
  
  
 */\*\*  
 \* Get write length  
 \** ***@return*** *write length  
 \*/* @Override  
 public int getWriteLength() {  
 return writeLength;  
 }  
  
 */\*\*  
 \* Print info about pencil  
 \*/* @Override  
 public void print() {  
 System.*out*.print("Pencil brand: " + brand + ", Write length: " +  
 writeLength + ", Hardness: " + hardness + "\n");  
 }  
  
 */\*\*  
 \* Compare 2 pencil  
 \** ***@param*** *o the object to be compared.  
 \** ***@return*** *\* the value 0 if this writes length is equal to the argument write length;  
 \* a value less than 0 if this writes length is numerically less than the argument writes length;  
 \* and a value greater than 0 if this writes length is numerically greater than the argument writes length  
 \*/* @Override  
 public int compareTo(IStationery o) {  
 Integer wl = writeLength;  
 return wl.compareTo(o.getWriteLength());  
 }  
}

PenCaseApp.java

import KI35.Skipar.Lab7.\*;  
  
public class PenCaseApp {  
  
 public static void main(String[] args) {  
  
 PenCase<? super IStationery> penCase = new PenCase<IStationery>();  
 penCase.addStationery(new Pencil("Sanford", 800, "H1"));  
 penCase.addStationery(new Pen("Skipar", 900, "Ballpoint"));  
 penCase.addStationery(new Pencil("Gilbert", 901, "B2"));  
 penCase.addStationery(new Pen("Parker", 1000, "Gel"));  
 IStationery res = penCase.findMin();  
 System.*out*.print("\nMin: ");  
 res.print();  
 System.*out*.print("\n Count of stationery: "+penCase.getCount());  
 }  
  
}

**Результат виконання програми**

Консоль

Stationery added: Pencil brand: Sanford, Write length: 800, Hardness: H1

Stationery added: Pen brand: Skipar, Write length: 900, Type: Ballpoint

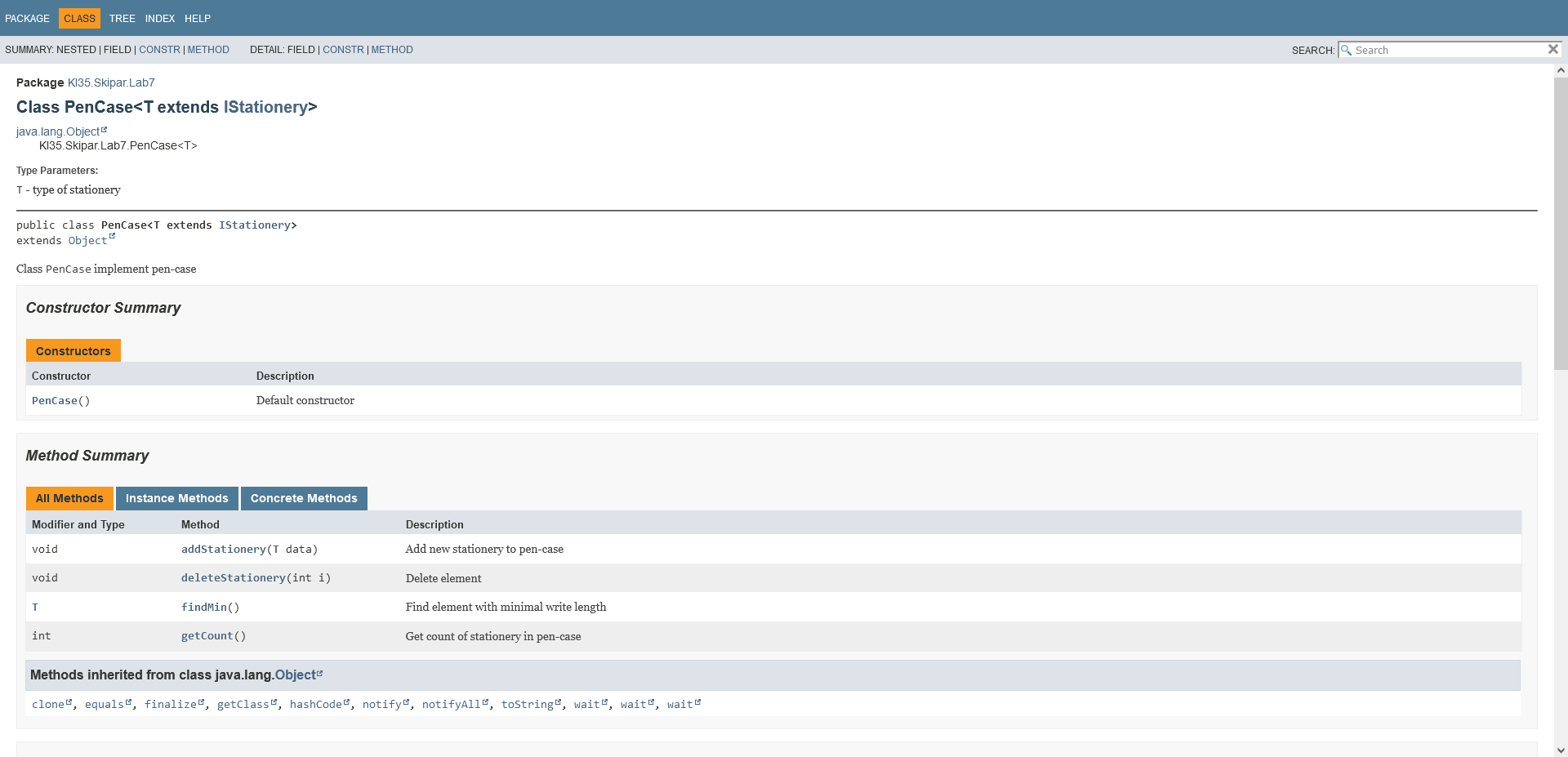
Stationery added: Pencil brand: Gilbert, Write length: 901, Hardness: B2

Stationery added: Pen brand: Parker, Write length: 1000, Type: Gel

Min: Pencil brand: Sanford, Write length: 800, Hardness: H1

Count of stationery: 4

**Фрагмент згенерованої документації**



**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. Обмеження для змінних типів можна поставити за допомогою ключового слова extends.

<параметризований тип extends обмежуючий Тип {& обмежуючий тип}>

**Висновок**

В результаті виконання лабораторної роботи я навчився працювати з параметризованими методами та класами в мові Java.