Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

з лабораторної роботи №6

з дисципліни: “Кросплатформні засоби програмування”

на тему: “Параметризоване програмування”

Виконав:

ст. гр. КІ-305

Шевчук М. О.

Прийняв:

Іванов Ю.С.

Львів – 2023

**Мета роботи:** 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. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагмент згенерованої документації.
4. Дати відповідь на контрольні запитання.

**Варіант №28**



**Лістинг програми:**

**Lab6ShevchoukKI305.java**

import java.util.\*;  
  
*/\*\**  
 *\* Class Lab6ShevchoukKI305 realize example program to lab work 6*  
 *\**   
 *\* @author Shevchouk Maxim*  
 *\* @version 1.1*  
 *\* @since version 1.0*  
 *\*/*  
public class Lab6ShevchoukKI305 {  
 public static void main(String[] args) {  
 Сabinet<? super Mezzanine> Сabinet = new Сabinet<Mezzanine>();  
 Сabinet.AddMezzanine(new Book("To Kill a Mockingbird", 200, 440));  
 Сabinet.AddMezzanine(new Utensils("Cup", 170));  
 Сabinet.AddMezzanine(new Utensils("Plate", 400));  
 Сabinet.AddMezzanine(new Book("1984", 405, 550));  
 Mezzanine res1 = Сabinet.findMin();  
 System.*out*.print("The smallest object in the mezzanine has: \n");  
 res1.print();  
 Mezzanine res2 = Сabinet.findMax();  
 System.*out*.print("The biggest object in the mezzanine has: \n");  
 res2.print();  
 }  
}  
  
  
*/\*\**  
 *\* Interface <code>Mezzanine</code> implements Mezzanine*  
 *\*/*  
interface Mezzanine extends Comparable<Mezzanine> {  
 public int getSize();  
  
 public void print();  
}  
  
*/\*\**  
 *\* Parameterized class <code>Сabinet</code> implements Сabinet*  
 *\*/*  
class Сabinet<T extends Mezzanine> { *// село*  
private ArrayList<T> arr;  
  
 */\*\**  
 *\* Constructor*  
 *\*/*  
public Сabinet() {  
 arr = new ArrayList<T>();  
 }  
  
 */\*\**  
 *\* Method find object with the smallest size*  
 *\*/*  
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;  
 }  
  
 */\*\**  
 *\* Method find object with the biggest size*  
 *\*/*  
public T findMax() {  
 if (!arr.isEmpty()) {  
 T max = arr.get(0);  
 for (int i = 1; i < arr.size(); i++) {  
 if (arr.get(i).compareTo(max) > 0)  
 max = arr.get(i);  
 }  
 return max;  
 }  
 return null;  
 }  
  
 */\*\**  
 *\* Method add new object to Сabinet*  
 *\**   
 *\* @param <code>Mezzanine</code> The class value*  
 *\*/*  
public void AddMezzanine(T Mezzanine) {  
 arr.add(Mezzanine);  
 System.*out*.print("Element added: ");  
 Mezzanine.print();  
 }  
  
 */\*\**  
 *\* Method delete land plot from Сabinet*  
 *\**   
 *\* @param <code>i</code> The index of element of array in Сabinet*  
 *\*/*  
public void DeleteMezzanine(int i) {  
 arr.remove(i);  
 }  
}  
  
  
  
  
  
  
class Book implements Mezzanine { *// книга*  
private String BookName;  
 private int BookPage;  
 private int BookSize;  
  
 */\*\**  
 *\* Constructor*  
 *\**   
 *\* @param <code>hFName</code> The Book name type*  
 *\* @param <code>hPage</code> The count of page in the Book*  
 *\* @param <code>hSize</code> The size of Book*  
 *\*/*  
public Book(String hName, int hPage, int hSize) {  
 BookName = hName;  
 BookPage = hPage;  
 BookSize = hSize;  
 }  
  
 */\*\**  
 *\* Method return Book name*  
 *\**   
 *\* @return BookFloors*  
 *\*/*  
public String getBookName() {  
 return BookName;  
 }  
  
 */\*\**  
 *\* Method set Book name for Book*  
 *\**   
 *\* @param <code>name</code> The Book name*  
 *\*/*  
public void setBookName(String name) {  
 BookName = name;  
 }  
  
 */\*\**  
 *\* Method return count of page in the Book*  
 *\**   
 *\* @return page*  
 *\*/*  
public int getPage() {  
 return BookPage;  
 }  
  
 */\*\**  
 *\* Method set count page in the Book*  
 *\**   
 *\* @param <code>n</code> The page*  
 *\*/*  
public void setMilkYield(int n) {  
 BookPage = n;  
 }  
  
 */\*\**  
 *\* Method return size of Book*  
 *\**   
 *\* @return size*  
 *\*/*  
public int getSize() {  
 return BookSize;  
 }  
  
 */\*\**  
 *\* Method compare objects type Mezzanine*  
 *\**   
 *\* @param <code>p</code> The size of object*  
 *\* @return num (0, 1, -1)*  
 *\*/*  
public int compareTo(Mezzanine p) {  
 Integer s = BookSize;  
 return s.compareTo(p.getSize());  
 }  
  
 */\*\**  
 *\* Method print information about Book*  
 *\*/*  
public void print() {  
 System.*out*.print("Book: " + BookName + ", Page: " + BookPage +  
 ", Book size: " + BookSize + ";\n");  
 }  
}  
  
  
  
  
  
  
  
  
  
  
  
class Utensils implements Mezzanine { *// кінь*  
private String UtensilsName;  
 private int UtensilsSize;  
  
 */\*\**  
 *\* Constructor*  
 *\**   
 *\* @param <code>sName</code> The name of Utensils*  
 *\* @param <code>ssize</code> The size of Utensils*  
 *\*/*  
public Utensils(String sName, int ssize) {  
 UtensilsName = sName;  
 UtensilsSize = ssize;  
 }  
  
 */\*\**  
 *\* Method return name of Utensils*  
 *\**   
 *\* @return UtensilsName*  
 *\*/*  
public String getName() {  
 return UtensilsName;  
 }  
  
 */\*\**  
 *\* Method set name of Utensils*  
 *\**   
 *\* @param <code>n</code> The UtensilsName*  
 *\*/*  
public void setName(String name) {  
 UtensilsName = name;  
 }  
  
 */\*\**  
 *\* Method return size of Utensils*  
 *\**   
 *\* @return size*  
 *\*/*  
public int getSize() {  
 return UtensilsSize;  
 }  
  
 */\*\**  
 *\* Method set size of Utensils*  
 *\**   
 *\* @param <code>n</code> The Utensilssize*  
 *\*/*  
public void setsize(int n) {  
 UtensilsSize = n;  
 }  
  
 */\*\**  
 *\* Method compare objects type Mezzanine*  
 *\**   
 *\* @param <code>p</code> The size of object*  
 *\* @return num (0, 1, -1)*  
 *\*/*  
public int compareTo(Mezzanine p) {  
 Integer s = UtensilsSize;  
 return s.compareTo(p.getSize());  
 }  
  
 */\*\**  
 *\* Method print information about Utensils*  
 *\*/*  
public void print() {  
 System.*out*.print("Utensils Name: " + UtensilsName + ",Utensils size: " + UtensilsSize + ";\n");  
 }  
}

**Результат виконання програми:**

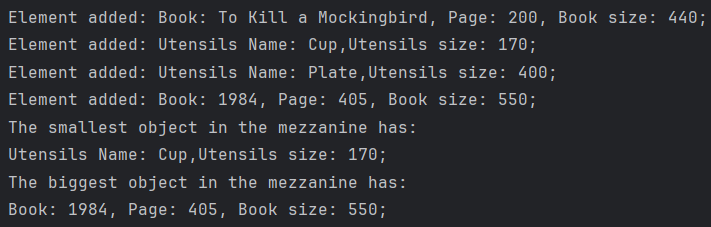
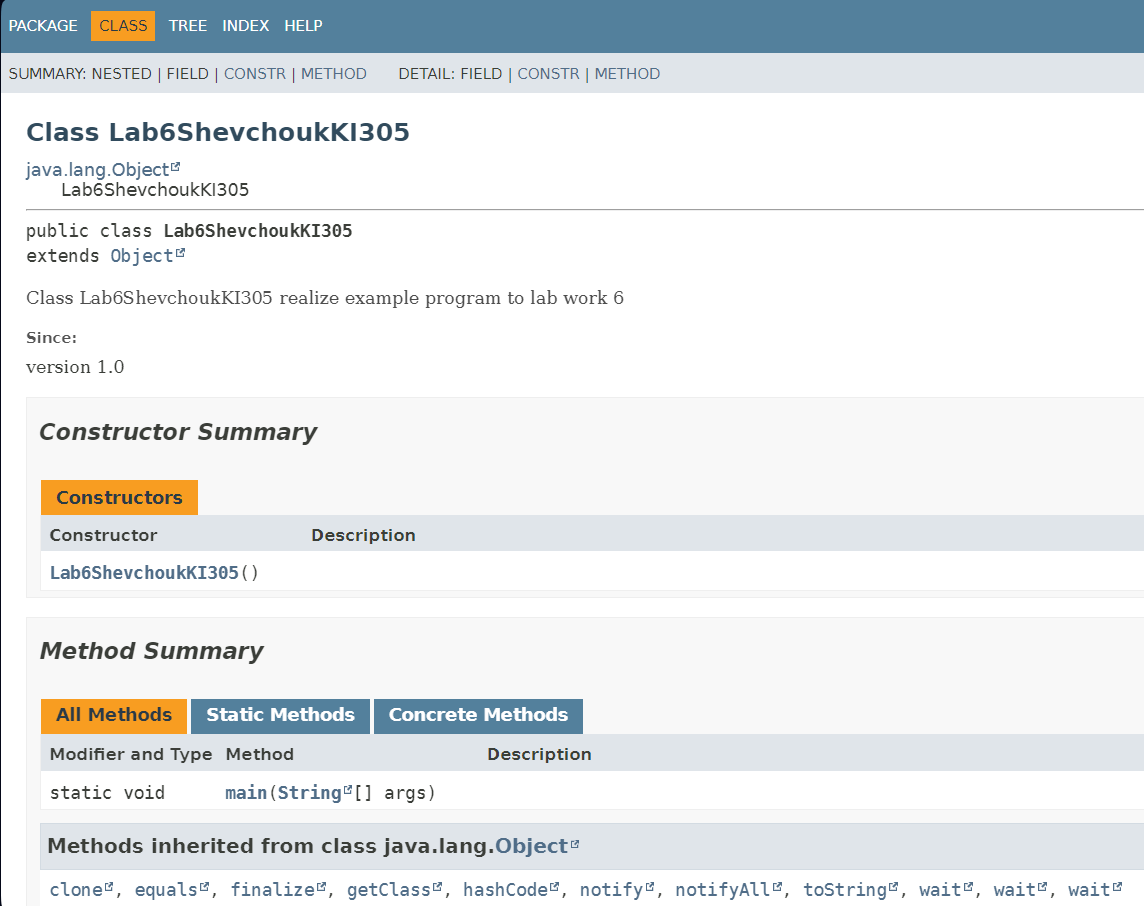


Рис.1.Результат виконання програми

**Згенерована документація**



**Відповіді на контрольні запитання:**

1. Дайте визначення терміну «параметризоване програмування».

*Параметризоване програмування*  це підхід в програмуванні, що дозволяє створювати загальні абстракції (такі як класи, функції або методи), які можуть працювати з будь-яким типом даних, або з декількома типами даних, без прив'язки до конкретного типу.

1. Розкрийте синтаксис визначення простого параметризованого класу.

class MyClass<T> {

// Код класу, де T - це параметр типу

}

1. Розкрийте синтаксис створення об’єкту параметризованого класу.

MyClass<int> obj = new MyClass<int>();

1. Розкрийте синтаксис визначення параметризованого методу.

void MyMethod<T>(T param) {

// Код методу, де T - це параметр типу

}

1. Розкрийте синтаксис виклику параметризованого методу.

MyMethod<int>(5); // Приклад виклику методу з типом даних int

1. Яку роль відіграє встановлення обмежень для змінних типів?

Роль встановлення обмежень для змінних типів  це забезпечення того, що параметр типу відповідає певним критеріям (наприклад, що він реалізує певний інтерфейс або має певний конструктор).

1. Як встановити обмеження для змінних типів?

Java обмеження для змінних типів можна встановити за допомогою ключового слова *extends* або *super* в контексті параметризованих класів і методів. Використання ключового слова extends дозволяє обмежити тип параметра відповідно до його успадкування, тоді як ключове слово super використовується для обмеження типу параметра його батьківським типом. Наприклад:

class MyClass<T extends SomeClass> {

// Код класу, де T - це параметр типу, який успадковує SomeClass

}

1. Розкрийте правила спадкування параметризованих типів.

Параметризовані класи можуть успадковувати інші класи, але при цьому потрібно передати параметри типу базовому класу.

1. Яке призначення підстановочних типів?

*Призначення підстановочних типів * це можливість використовувати параметризовані типи в загальних контекстах, не вказуючи конкретний тип даних.

1. Застосування підстановочних типів.

Застосування підстановочних типів допомагає підтримувати гнучкість і повторне використання коду, дозволяючи використовувати загальні класи та методи для різних типів даних, не змінюючи їхнього внутрішнього коду.

**Висновок:**На даній лабораторній роботі я оволодів навиками параметризованого програмування мовою Java.