Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра «Електронних обчислювальних машин»

Звіт
з лабораторної роботи № 6
з дисципліни: «Кросплатформенні засоби програмування»
на тему: «ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ»

Виконав:
студент групи КІ-306
Хмільовський С. Р.
Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2023

Мета роботи: оволодіти навиками параметризованого програмування мовою Java.

Завдання (варіант № 22)

1. Створити параметризований клас, що реалізує предметну область задану варіантом **(Валіза)**. 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.

Вихідний код програми:

Файл SuitCaseApp.java

```
//package KI306.Khmilovskiy.Lab6;

/**
 * The main class for testing the SuitCase implementation.
 */
public class SuitCaseApp {
    /**
     * The main entry point for the application.
     * @param args Command line arguments.
     */
        public static void main(String[] args) {
            SuitCase<? super Item> suitcase = new SuitCase<>();

            System.out.println();
            suitcase.addItem(new Clothing("Shirt","Burberry", 599, 35));
            suitcase.addItem(new Clothing("Pants", "Nike", 199, 32));
```

```
            suitcase.addItem(new Book("\'The Case-Book of Sherlock Holmes\'",
"Detective", 2021, 11));
            suitcase.addItem(new Book("\'The Great Gatsby\'", "Fiction", 1925,
10));
            suitcase.addItem(new Book("\'To Kill a Mockingbird\'", "Fiction",
1960, 8));

            suitcase.removeItem(4);

            System.out.print("\nContents of SuitCase: \n");
            suitcase.printContents();

            Item minItem = suitcase.findMin();
            System.out.print("\nThe smallest item in the SuitCase is: ");
            minItem.print();
        }
    }
```

Файл SuitCase.java

```
//package KI306.Khmilovskiy.Lab6;
import java.util.ArrayList;

/**
 * A generic class representing a suitcase that can hold items of type T.
 * @param <T> The type of items that the suitcase can hold.
 */
class SuitCase<T extends Item> {
    private ArrayList<T> items;

    /**
     * Constructs a new SuitCase object.
     */
    public SuitCase() {
        items = new ArrayList<>();
    }

    /**
     * Finds the item with the minimum size in the suitcase.
     * @return The item with the minimum size, or null if the suitcase is empty.
     */
    public T findMin() {
        if (!items.isEmpty()) {
            T min = items.get(0);
            for (int i = 1; i < items.size(); i++) {
                if (items.get(i).compareTo(min) < 0)
                    min = items.get(i);
            }
            return min;
        }
        return null;
    }

    /**
     * Adds an item to the suitcase.
     * @param item The item to be added.
     */
    public void addItem(T item) {
        items.add(item);
        System.out.print("Item added: ");
        item.print();
    }

    /**
     * Removes an item from the suitcase at the specified index.
```

```
     * @param i The index of the item to be removed.
     */
    public void removeItem(int i) {
        if (i >= 0 && i < items.size()) {
            items.remove(i);
            System.out.println("Item removed at index " + i);
        } else {
            System.out.println("Invalid index. Cannot remove item.");
        }
    }

    /**
     * Prints the contents of the suitcase.
     */
    public void printContents() {
        if (!items.isEmpty()) {
            for (T item : items) {
                item.print();
            }
        } else {
            System.out.println("SuitCase is empty. No items available.");
        }
    }
}

/**
 * A class representing clothing items.
 */
class Clothing implements Item {
    private String clothingType;
    private String clothingBrand;
    private double clothingCost;
    private int clothingSize;

    /**
     * Constructs a new Clothing object.
     *
     * @param cType   The type of clothing.
     * @param cBrand  The brand of clothing.
     * @param cCost   The cost of clothing.
     * @param cSize   The size of clothing.
     */
    public Clothing(String cType, String cBrand, double cCost, int cSize) {
        clothingType = cType;
        clothingBrand = cBrand;
        clothingCost = cCost;
        clothingSize = cSize;
    }

    // SET + GET [TYPE]
    public String getClothingType() {
        return clothingType;
    }

    public void setClothingType(String type) {
        clothingType = type;
    }

    // SET + GET [BRAND]
    public String getClothingBrand() {
        return clothingBrand;
    }

    public void setClothingBrand(String brand) {
        clothingBrand = brand;
    }
```

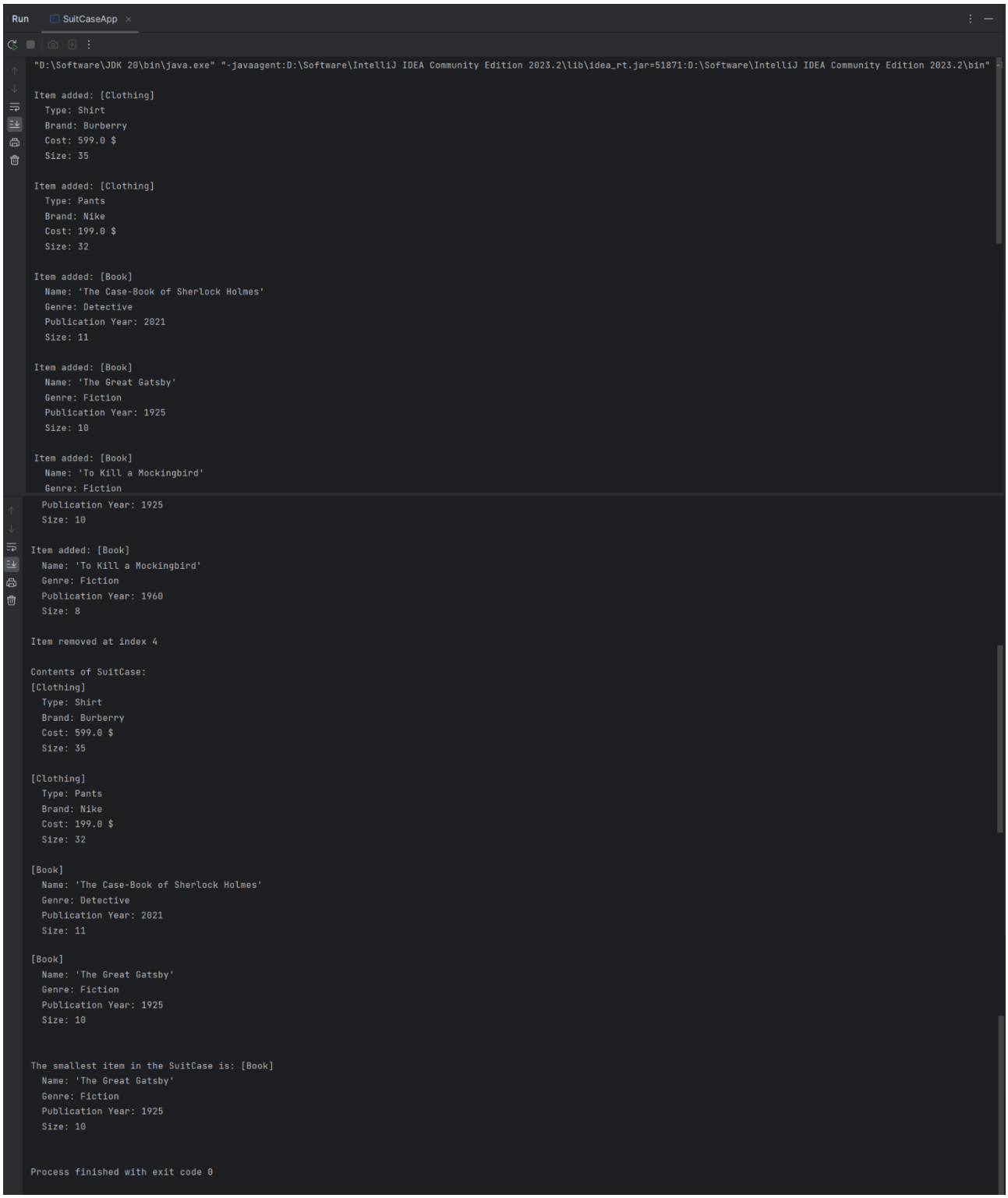
```
    // SET + GET [COST]
    public double getClothingCost() {
        return clothingCost;
    }

    public void setClothingCost(double cost) {
        clothingCost = cost;
    }

    // SET [SIZE]
    public void setClothingSize(int size) {
        clothingSize = size;
    }

    // Implementing methods from Item interface:
    public int getSize() {
        return clothingSize;
    }

    public int compareTo(Item item) {
        Integer s = clothingSize;
        return s.compareTo(item.getSize());
    }

    public void print() {
        System.out.println("[Clothing]");
        System.out.println("  Type: " + clothingType);
        System.out.println("  Brand: " + clothingBrand);
        System.out.println("  Cost: " + clothingCost + " $");
        System.out.println("  Size: " + clothingSize);
        System.out.println();
    }
}

/**
 * A class representing books.
 */
class Book implements Item {
    private String bookName;
    private String bookGenre;
    private int bookPublicationYear;
    private int bookSize;

    /**
     * Constructs a new Book object.
     *
     * @param bName              The name of the book.
     * @param bGenre             The genre of the book.
     * @param bPublicationYear   The publication year of the book.
     * @param bSize              The size of the book.
     */
    public Book(String bName, String bGenre, int bPublicationYear, int bSize) {
        bookName = bName;
        bookGenre = bGenre;
        bookPublicationYear = bPublicationYear;
        bookSize = bSize;
    }

    // SET + GET [NAME]
    public String getBookName() {
        return bookName;
    }
    public void setBookName(String name) {
        bookName = name;
    }
```

```
    // SET + GET [GENRE]
    public String getBookGenre() {
        return bookGenre;
    }
    public void setBookGenre(String genre) {
        bookGenre = genre;
    }

    // SET + GET [YEAR]
    public int getBookPublicationYear() {
        return bookPublicationYear;
    }
    public void setBookPublicationYear(int year) {
        bookPublicationYear = year;
    }

    // SET [SIZE]
    public void SetBookSize(int n) {
        bookSize = n;
    }

    // Implementing methods from Item interface:
    public int getSize() {
        return bookSize;
    }
    public int compareTo(Item item) {
        Integer s = bookSize;
        return s.compareTo(item.getSize());
    }
    public void print() {
        System.out.println("[Book]");
        System.out.println("  Name: " + bookName);
        System.out.println("  Genre: " + bookGenre);
        System.out.println("  Publication Year: " + bookPublicationYear);
        System.out.println("  Size: " + bookSize);
        System.out.println();
    }
}
```

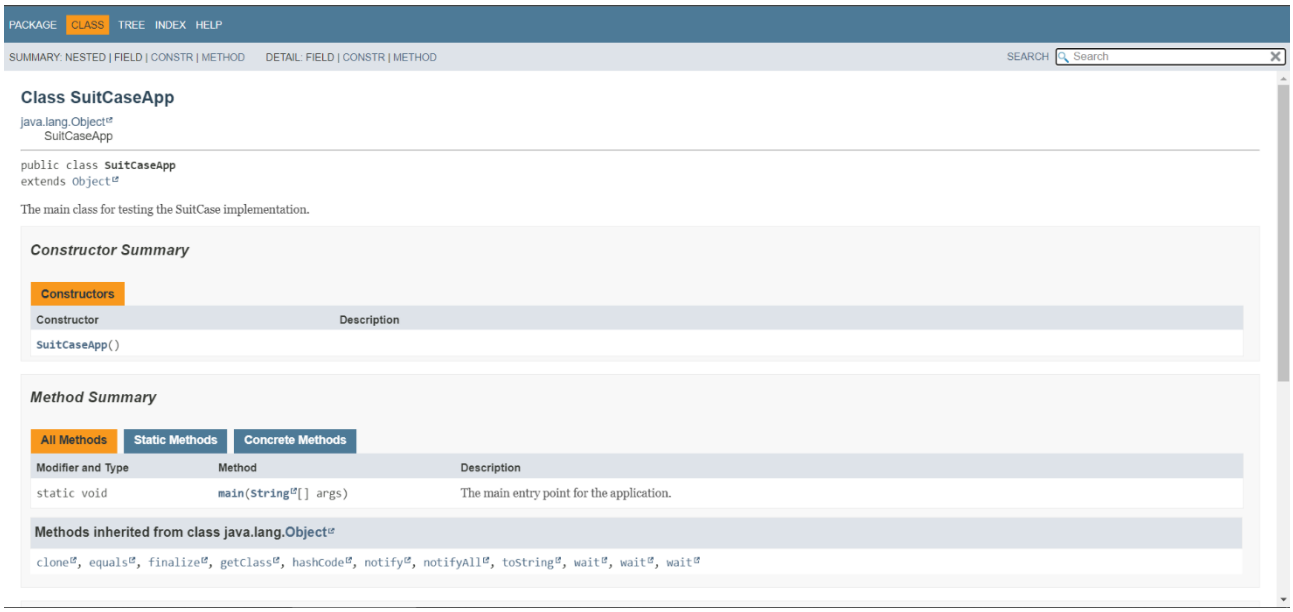
Файл Item.java

```
//package KI306.Khmilovskiy.Lab6;
/**
 * An interface representing an item.
 */
interface Item extends Comparable<Item> {
    /**
     * Gets the size of the item.
     * @return The size of the item.
     */
    int getSize();

    //int compareTo(Item item);

    /**
     * Prints information about the item.
     */
    void print();
}
```

Результат виконання програми:



Фрагмент згенерованої документації JavaDoc:



Відповіді на контрольні запитання:

1. Дайте визначення терміну «параметризоване програмування».

Параметризоване програмування (або generics) - це механізм мов програмування, який дозволяє створювати код, який може працювати з різними типами даних. Воно дозволяє створювати універсальні абстракції, які можуть бути використані для різних типів, забезпечуючи при цьому безпеку типів.

2. Розкрийте синтаксис визначення простого параметризованого класу.

```
public class MyGenericClass<T> {
    // код класу
}
```

3. Розкрийте синтаксис створення об’єкту параметризованого класу.

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

4. Розкрийте синтаксис визначення параметризованого методу.

```
public <T> void myGenericMethod(T parameter) {
    // код методу
}
```

5. Розкрийте синтаксис виклику параметризованого методу.

```
myGenericMethod("Hello");
```

6. Яку роль відіграє встановлення обмежень для змінних типів?

Встановлення обмежень для змінних типів (generics constraints) дозволяє обмежити можливі типи, які можна використовувати в параметризованих типах, забезпечуючи при цьому певний функціонал або властивості для цих типів.

7. Як встановити обмеження для змінних типів?

```
public class MyGenericClass<T extends SomeClass> {
    // код класу
}
```

8. Розкрийте правила спадкування параметризованих типів.

Параметризовані типи спадковуються аналогічно непараметризованим типам. При цьому враховується також параметр типу.

9. Яке призначення підстановочних типів?

Підстановочні типи (wildcards) використовуються для створення більш гнучких методів та класів, які можуть працювати з різними типами. Два основних типи підстановочних типів: ? extends T (upper-bounded wildcard) і ? super T (lower-bounded wildcard).

10. Застосування підстановочних типів.

- `? extends T`: Використовується для доступу до об'єктів з типами, які є підтипами T.
- `? super T`: Використовується для передачі об'єктів з типами, які є супертипами T.

Висновок:

На даній лабораторній роботі я ознайомився з використанням параметризованого програмування. Створив клас який реалізує предметну область «валіза». Та розробив клас драйвер який показує роботу параметризованого класу-контейнера.