

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



Звіт
з лабораторної роботи № 7
з дисципліни: «Кросплатформенні засоби програмування» на
тему: «Параметризоване програмування»

Виконав:
студент групи КІ-36
Музика М
Прийняв: доцент
кафедри ЕОМ
Іванов Ю. С.

Львів – 2022

Мета роботи: оволодіти навиками параметризованого програмування мовою Java.

Завдання (варіант №8)

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 фрагменту згенерованої документації.
4. Дати відповідь на контрольні запитання.
 - Дайте визначення терміну «параметризоване програмування».
 - Розкрийте синтаксис визначення простого параметризованого класу.
 - Розкрийте синтаксис створення об'єкту параметризованого класу.
 - Розкрийте синтаксис визначення параметризованого методу.
 - Розкрийте синтаксис виклику параметризованого методу.
 - Яку роль відіграє встановлення обмежень для змінних типів?
 - Як встановити обмеження для змінних типів?
 - Розкрийте правила спадкування параметризованих типів.
 - Яке призначення підстановочних типів?
 - Застосування підстановочних типів.

Індивідуальне завдання: Коробка

Текст програми

BoxDriver.java

```
package KI36.Muzyka.Lab7;

import java.util.ArrayList; import
java.util.Objects;

import static java.lang.System.out;

public class BoxDriver
{
    /**
     * Main method the entry point to start program
     * @param args
     */    public static void
main(String[] args)
    {
        Box<? super Item> bigBox = new Box<Item>();

        Sweet oreoCrumbs = new Sweet("Oreo Crushed Cookie Crumbs (No
Creme)", 410);

        bigBox.addItem(new Tools("Calipers", 210, "200MM"));
bigBox.addItem(new Tools("Roulette", 100, "7.5 м * 25 мм"));
bigBox.addItem(new Tools("Chisel", 180, "CR-V 22 мм"));
bigBox.addItem(new Book("English Grammar in Use Intermediate",
100, " Antoine de Saint-Exupery", "Art" ));
        bigBox.addItem(new Sweet("Oreo Small Crushed Cookie", 400));
bigBox.addItem(new Sweet("Toblerone White Chocolate Large Bar",
360));

        bigBox.addItem(oreoCrumbs);
        out.println("-----");

        var min = bigBox.findMin();
        out.println("The element with smallest weight is : ");
min.printInfo();

        out.println("\n====Printing=all=item=in=the==box====\n");
bigBox.printAllArray();

        bigBox.deleteDataByObject(oreoCrumbs);
        out.println("\n====Printing=all=item=in=the==box====\n");
bigBox.printAllArray();
```

```

        bigBox.deleteDataByIndex(1);
out.println("\n===Printing=all=item=in=the==box===\n");
bigBox.printAllArray();

    }
}

class Box <T extends Item>
{
    private ArrayList<T> arr;

    /**
    * Constructor default
    */
    public Box() {arr = new
ArrayList<>();}

    /**
    * Method to find items with minimal weight
    * @return Object with minimal elements
    */
    public T findMin()
    {
        if(!arr.isEmpty())
        {
            T min = arr.get(0);
for(var i: arr)
            {
                if(i.compareTo(min) < 0)
min = i;
            }
return min;
        }
        return null;
    }

    /**
    * Method to add item to box
    * @param item Item that will be added
    */
    public void
addItem(T item)
    {
        out.println("-----");
arr.add(item);
        out.println("Item added # " + arr.size() + " :");
item.printInfo();
    }

    /**
    * Method to pop item from box by object

```

```

* @param item Item that will be deleted
    */    public void
deleteDataByObject(T item)
    {
        if(arr.removeIf(arr -> Objects.equals(item, arr)))
        {
            out.println("The following item deleted: " );
            item.printInfo();
        }else
        {
            out.println("Item cannot be deleted: " );
            item.printInfo();
        }
    }

    /**
    * Method to pop item from box by index
    * @param index Index of item that will be deleted
    */    public void
deleteDataByIndex(int index)
    {
        if(index > 0 && index <= arr.size())
        {
            out.println("The following item deleted ");
            arr.remove(index-1).printInfo();
        }
        else
        {
            out.println("Item cannot be deleted: " );
            arr.get(index).printInfo();
        }
    }

    /**
    * Method to view all item in the box
    */    public void
printAllArray()
    {
        int cnt =
0;
        for(var a :
arr)
        {
            if(!arr.isEmpty())
            {
                out.println("Item # " + (++cnt));
                out.println("-----");
                a.printInfo();
                out.println("-----");
            }else
            {
                out.println("Box is empty");
            }
        }
    }

```

```

    }
}
}
}

```

```

interface Item extends Comparable<Item>
{
    /**
    * Method to get weight of item
    * @return The weight of item
    */    public int
    getWeight();

    /**
    * Method to get info of item
    */    public void
    printInfo();
}

```

```

class Book implements Item
{
    private String bookName;
private String genre;    private
String author;    private int
    bookWeight;

    public String getBookName() {
return bookName;
    }

    public void setBookName(String bookName) {
this.bookName = bookName;
    }

    public String getGenre() {
return genre;
    }

    public void setGenre(String genre) {
this.genre = genre;
    }

    public String getAuthor() {
return author;
    }

    public void setAuthor(String author) {
this.author = author;
    }

    public int getBookWeight() {
return bookWeight;
    }

    public void setBookWeight(int bookWeight) {
this.bookWeight = bookWeight;
    }
}

```

```

    }

    /**
    * Construcnor for initialize item
    * @param bookName The book name
    * @param bookWeight The book weight
    * @param author The book author
    * @param genre The book genre
    */
    Book(String bookName, int bookWeight, String author, String genre)
    {
        this.bookName = bookName;
    this.bookWeight = bookWeight;
    this.author = author;        this.genre
    = genre;
    }

    @Override
    public int getWeight() {return bookWeight;}

    @Override
    public void printInfo()
    {
        out.println
            ("Name: " + bookName +
            "\nAuthor: " + author +
            "\nGenre: " + genre +
            "\nWeigh: " + bookWeight
            );
    }

    @Override
    public int compareTo(Item item)
    {
        Integer cmp = bookWeight;
        return cmp.compareTo(item.getWeight());
    }
}
class Tools implements Item
{
    String nameTools;

    String description;
    int weightTools;

    /**
    * Constructor to initialize Tools
    * @param nameTools The tools name
    * @param weightTools The weight tools
    * @param description The tools description

```

```

        */    public Tools(String nameTools, int weightTools, String
description)
    {
        this.nameTools = nameTools;
this.weightTools = weightTools;
this.description = description;
    }

    @Override
    public int getWeight()
    {
        return weightTools;
    }

    @Override
    public void printInfo()
    {
        out.println
        (
            "Name: " + nameTools +
            "\nDescription: " + description +
            "\nWeigh: " + weightTools
        );
    }

    @Override
    public int compareTo(Item item)
    {
        Integer cmp = weightTools;
        return cmp.compareTo(item.getWeight());
    }
}
class Sweet implements Item
{
    String nameSweets;

    int weightSweet;

    /**
    * Constructor to initialize Sweets
    * @param nameSweets The sweet name
    * @param weightSweet The sweet weight
    */    public Sweet(String nameSweets, int
weightSweet)
    {
        this.nameSweets = nameSweets;
this.weightSweet = weightSweet;
    }
}

```

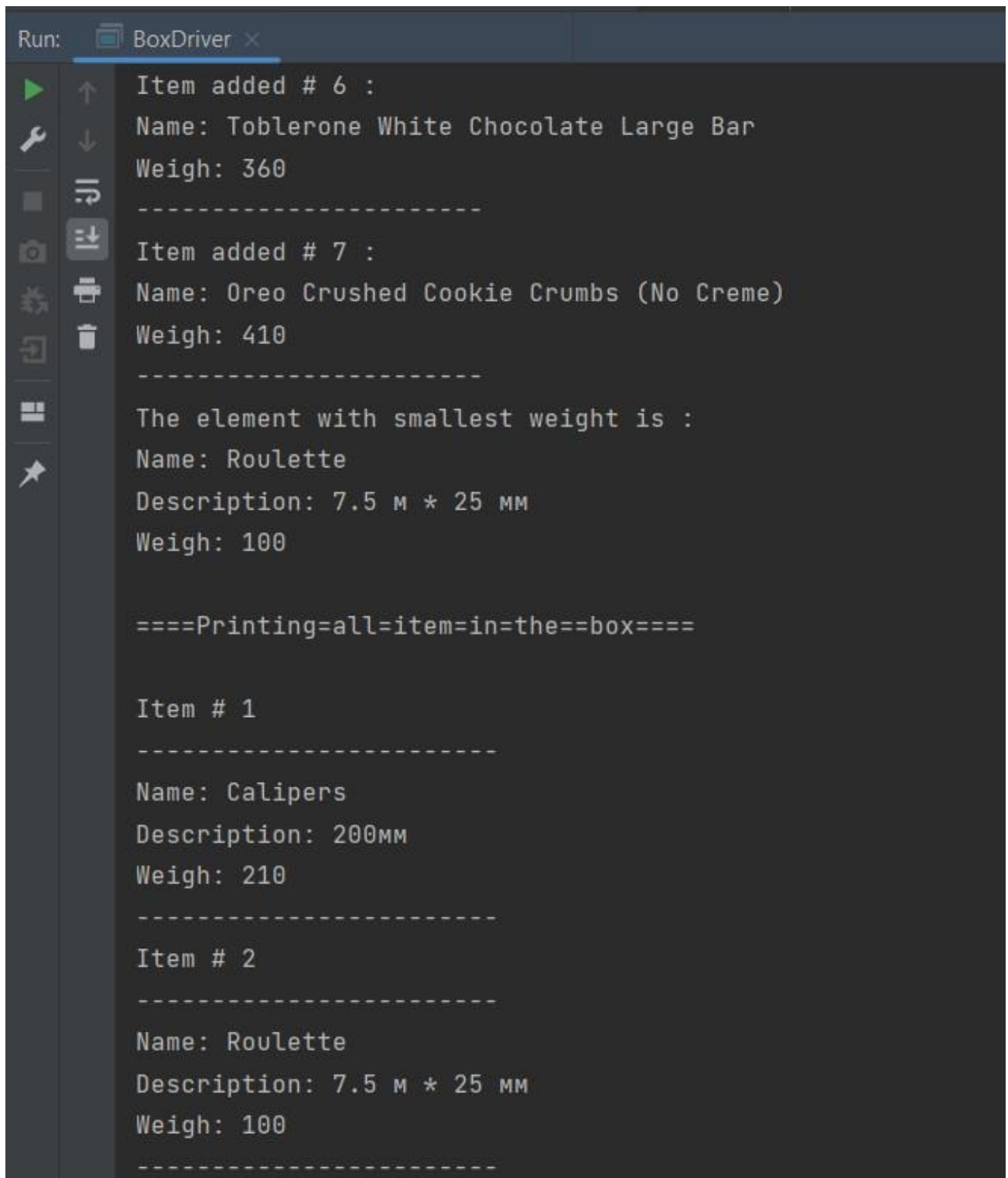


```
        @Override        public
int getWeight() {
return weightSweet;
}

@Override
public void printInfo()
{
    out.println
        ("Name: " + nameSweets +
         "\nWeigh: " + weightSweet
        );
}

@Override
public int compareTo(Item item)
{
    Integer cmp = weightSweet;
return cmp.compareTo(item.getWeight());
}
}
```

Результат виконання програми



```
Run: BoxDriver x
Item added # 6 :
Name: Toblerone White Chocolate Large Bar
Weigh: 360
-----
Item added # 7 :
Name: Oreo Crushed Cookie Crumbs (No Creme)
Weigh: 410
-----
The element with smallest weight is :
Name: Roulette
Description: 7.5 м * 25 мм
Weigh: 100

====Printing=all=item=in=the==box====

Item # 1
-----
Name: Calipers
Description: 200мм
Weigh: 210
-----
Item # 2
-----
Name: Roulette
Description: 7.5 м * 25 мм
Weigh: 100
-----
```

Рис.1.Фрагмент із результату виконання програми

Відповіді на контрольні запитання

1. Дайте визначення терміну «параметризоване програмування».

Параметризоване програмування - це такий підхід до опису даних і алгоритмів, який дозволяє їх використовувати з різними типами даних без зміни їх опису.

2. Розкрийте синтаксис визначення простого параметризованого класу.

```
[public] class НазваКласу <параметризованийТип {,параметризованийТип}>
{...}
```

3. Розкрийте синтаксис створення об'єкту параметризованого класу.

```
НазваКласу < перелікТипів > = new НазваКласу < перелікТипів > (параметри);
```

4. Розкрийте синтаксис визначення параметризованого методу.

```
Модифікатори <параметризованийТип {,параметризованийТип}> типПовернення
назваМетоду(параметри);
```

5. Розкрийте синтаксис виклику параметризованого методу.

```
Модифікатори <параметризованийТип {,параметризованийТип}> типПовернення
назваМетоду(параметри);
```

6. Яку роль відіграє встановлення обмежень для змінних типів?

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.

7. Як встановити обмеження для змінних типів?

Модифікатори <параметризований тип extends обмежуючийТип {& обмежуючий тип}
{, параметризований тип extends обмежуючийТип {& обмежуючий тип} }>
типПовернення назваМетоду(параметри);

8. Розкрийте правила спадкування параметризованих типів.

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.
2. 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
3. Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи.

9. Яке призначення підстановочних типів?

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

10.Застосування підстановочних типів.

Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

Висновок

Під час виконання даної лабораторної роботи я оволодів навиками параметризованого програмування мовою Java. Дізнався про Generic(и), тобто це параметризовані типи, за допомогою яких можна оголошувати класи, методи та інтерфейси, де тип даних заданий у вигляді параметру. Навчився параметризовувати класи та методи, встановлювати для змінних типів обмеження та створювати підстановочні типи.