Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

З лабораторної роботи №7

Варіант – 6

З дисципліни: «Кросплатформні засоби програмування»

На тему: «Параметризоване програмування»

Виконав: ст. гр. КІ-35

Павлик С. В.

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Львів 2022

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

**Завдання(Варіант 6)**

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:

1. Як встановити обмеження для змінних типів?
2. Яке призначення підстановочних типів?

**Варіант завдання: 6. Шафа.**

**Код програми:**

**Файл Main.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\* Main class realize driver for the application

\* @author Serhii Pavlyk

\* @version 1.0

\*/

public class Main {

/\*\*

\* Method main start point to the program

\* @param args arguments to pass to the start point

\*/

public static void main(String[] args)

{

Wardrobe<? super Clothes> wardrobe = new Wardrobe<Clothes>();

wardrobe.AddData(new T\_Short(44,"red","sun",10.4));

wardrobe.AddData(new Jacket(52, "black", "skull", 20.9));

wardrobe.AddData(new Pants(112, "black", "jeans", 16.9));

wardrobe.AddData(new T\_Short(43,"blue","flower",12.4));

System.out.println("The cheapest cloth in the Wardrobe:");

wardrobe.findMin().printInformation();

wardrobe.DeleteData("sun");

System.out.println("The cheapest cloth in the Wardrobe:");

wardrobe.findMin().printInformation();

}

}

**Файл actionsWithClothes.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\*

\* interface actionsWith clotheses realize what we can do

\* with our classes which interpret the clothes

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public interface actionsWithClothes extends Comparable<actionsWithClothes>

{

/\*\*

\* Method getColor() return the color of the clothes

\* @return color of the clothes

\*/

String getColor();

/\*\*

\* Method getSize() returns the size of the clothes

\* @return color of the clothes

\*/

int getSize();

/\*\*

\* Method getPrice() returns the price of the clothes

\* @return price of the clothes

\*/

double getPrice();

/\*\*

\* Method printInformation() print the information of the clothes

\*/

void printInformation();

/\*\*

\* Method getName() retuns the combination of the specific and color of the clothes

\* @return generated name of the clothes

\*/

String getName();

**}**

**Файл Clothes.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\* class Clothes realize abstract model of clothes

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public abstract class Clothes implements actionsWithClothes

{

/\*\*

\* Constructor of class Cloth

\* @param size size of Cloth

\* @param color color of Cloth

\* @param price price of Cloth

\*/

public Clothes(int size, String color, double price)

{

this.size = size;

this.color = color;

this.price = price;

}

private int size ;

private String color;

private double price;

/\*\*

\* Method returns size of clothes

\*/

@Override

public int getSize() {

return size;

}

/\*\*

\* Method returns color of clothes

\*/

@Override

public String getColor() {

return color;

}

/\*\*

\* Method returns price of clothes

\*/

@Override

public double getPrice() {

return price;

}

}

**Файл Jacket.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\* class Jacket realize model of Jacket

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public class Jacket extends Clothes

{

private String weave;

/\*\*

\* Method getName() generates the name of Jacket based on its weave and color

\* @return name of Jacket

\*/

public String getName() {

return weave + getColor();

}

/\*\*

\* Constructor of class Jacket

\* @param size size of Jaket

\* @param color color of Jacket

\* @param weave weave of Jacket

\* @param price price of Jacket

\*/

public Jacket(int size, String color, String weave, double price)

{

super(size,color,price);

this.weave = weave;

}

/\*\*

\* Method printInformation() print all information about Jacket

\*/

@Override

public void printInformation() {

System.out.println("Jacket:\nColor: " + getColor() +"\nSize: " + getSize()

+"\nWeave: " + weave +"\nPrice: " + getPrice() );

}

/\*\*

\* Method compareTo() compare price of Jacket with price of another cloth

\* @param another element of cloth with extends class Cloth

\* @return result of comparing

\*/

@Override

public int compareTo(actionsWithClothes another) {

Double a = this.getPrice();

return a.compareTo(another.getPrice());

}

}

**Файл Pants.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\* class Pants realize model of pants

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public class Pants extends Clothes

{

private String material;

/\*\*

\* Method getName() generates the name of Jacket based on its material and color

\* @return name of Jacket

\*/

public String getName() {

return material + getColor();

}

/\*\*

\* Constructor of class Pants

\* @param size size of Pants

\* @param color color of Pants

\* @param material material of Pants

\* @param price price of Pants

\*/

public Pants(int size, String color, String material, double price)

{

super(size,color,price);

this.material = material;

}

/\*\*

\* Method printInformation() print all information about Pants

\*/

@Override

public void printInformation() {

System.out.println("Pants:\nColor: " + getColor() +"\nSize: " + getSize()

+"\nMaterial: " + material +"\nPrice: " + getPrice() );

}

/\*\*

\* Method compareTo() compare price of Pants with price of another clothes

\* @param another element of clothes which extends class Clothes

\* @return result of comparing

\*/

@Override

public int compareTo(actionsWithClothes another) {

Double a = this.getPrice();

return a.compareTo(another.getPrice());

}

}

**Файл T\_Short.java**

/\*\*

\* LAB\_07 package

\*/

package LAB\_07;

/\*\*

\* class T\_Short realize model of t\_short

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public class T\_Short extends Clothes

{

private String weave;

/\*\*

\* Method getName() generates the name of T\_Short based on its weave and color

\* @return name of Jacket

\*/

public String getName() {

return weave + getColor();

}

/\*\*

\* Constructor of class T\_Short

\* @param size size of T\_Short

\* @param color color of T\_Short

\* @param weave weave of T\_Short

\* @param price price of T\_Short

\*/

public T\_Short(int size, String color, String weave, double price)

{

super(size,color,price);

this.weave = weave;

}

/\*\*

\* Method printInformation() print all information about t\_short

\*/

@Override

public void printInformation() {

System.out.println("T-Short:\nColor: " + getColor() +"\nSize: " + getSize()

+"\nWeave: " + weave +"\nPrice: " + getPrice() );

}

/\*\*

\* Method compareTo() compare price of t\_short with price of another clothes

\* @param another element of clothes which extends class Clothes

\* @return result of comparing

\*/

@Override

public int compareTo(actionsWithClothes another) {

Double a = this.getPrice();

return a.compareTo(another.getPrice());

}

}

**Файл Wardrobe.java**

package LAB\_07;

import java.util.ArrayList;

/\*\*

\* Generic class Wardrobe realize model of Wardrobe.

\* @param <T> class can be created only by types which extends class Clothes

\* @author Pavlyk Serhii

\* @version 1.0

\*/

public class Wardrobe <T extends Clothes>

{

private ArrayList<T> arrayList;

/\*\*

\* Constructor of class Wardrobe

\*/

public Wardrobe()

{

arrayList = new ArrayList<T>();

}

/\*\*

\* Method findMin() finds cloth with minimal price

\* @return the cheapest clothes

\*/

public T findMin()

{

if (!arrayList.isEmpty())

{

T min = arrayList.get(0);

for (int i=1; i< arrayList.size(); i++)

{

if ( arrayList.get(i).compareTo(min) < 0 )

min = arrayList.get(i);

}

return min;

}

return null;

}

/\*\*

\* Method AddData() adds instances of classes that extends from

\* the Clothes class to the collection arrayList

\* @param data instances of classes that extend from the Clothes class

\*/

public void AddData(T data)

{

arrayList.add(data);

System.out.print("Element added: ");

data.printInformation();

}

/\*\*

\* Method DeleteData() deletes instances of classes that extend

\* from the Clothes class from colection arrayList of Wardrobe

\* @param key the key clue which identifies the

\* instances of classes that extend from the Clothes class

\*/

public void DeleteData(String key)

{

int index = 0 ;

for (int j = 0; j < arrayList.size(); j++)

{

if (arrayList.get(j).getName().equals(key + arrayList.get(j).getColor()))

index = j;

}

System.out.println("Element deleted:");

arrayList.get(index).printInformation();

arrayList.remove(index);

if (!arrayList.isEmpty())

{

System.out.println("The following things remained in the Wardrobe:");

for (int j = 0; j < arrayList.size(); j++)

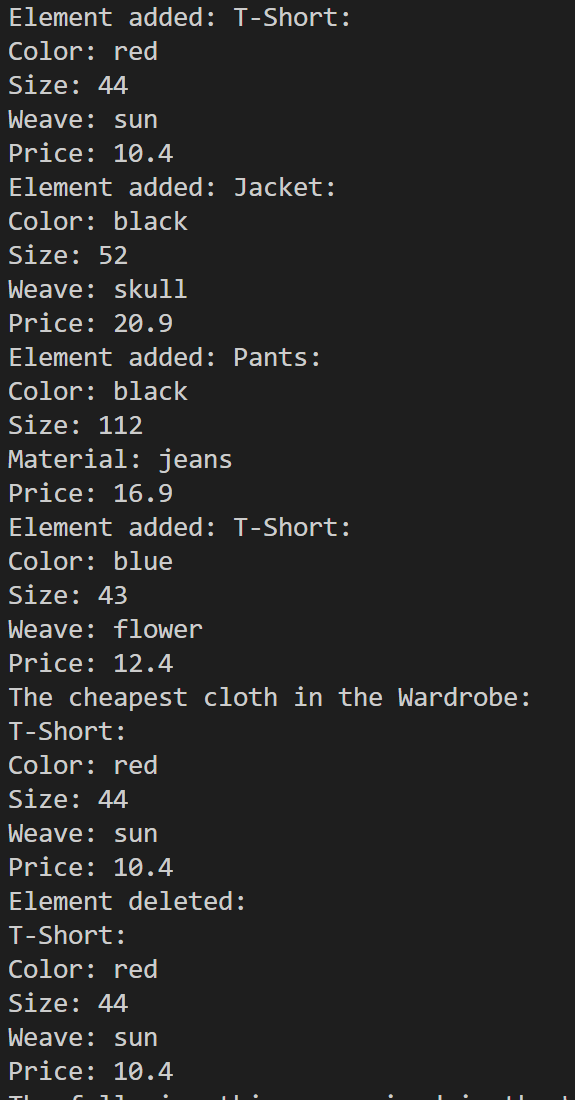
arrayList.get(j).printInformation();

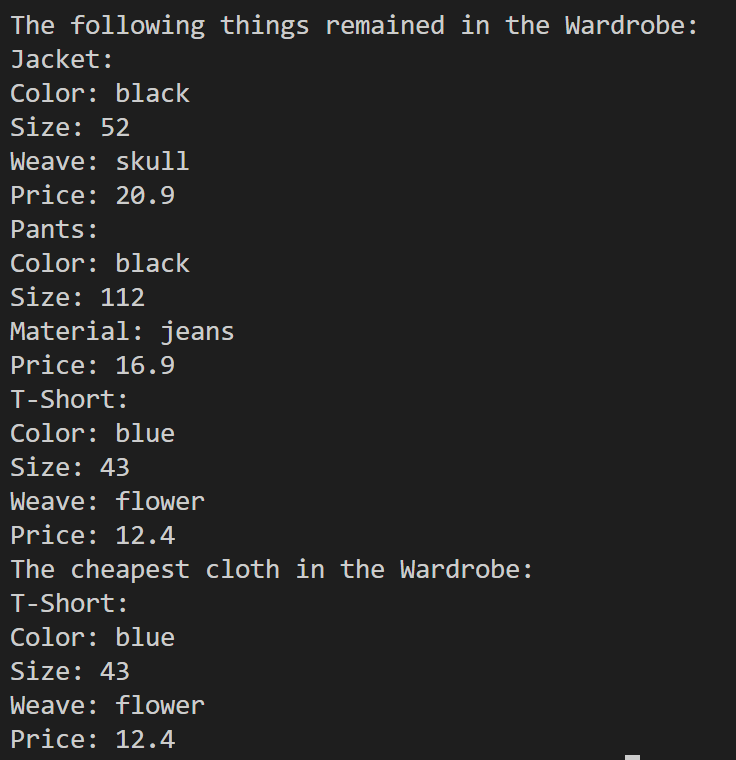
}

}

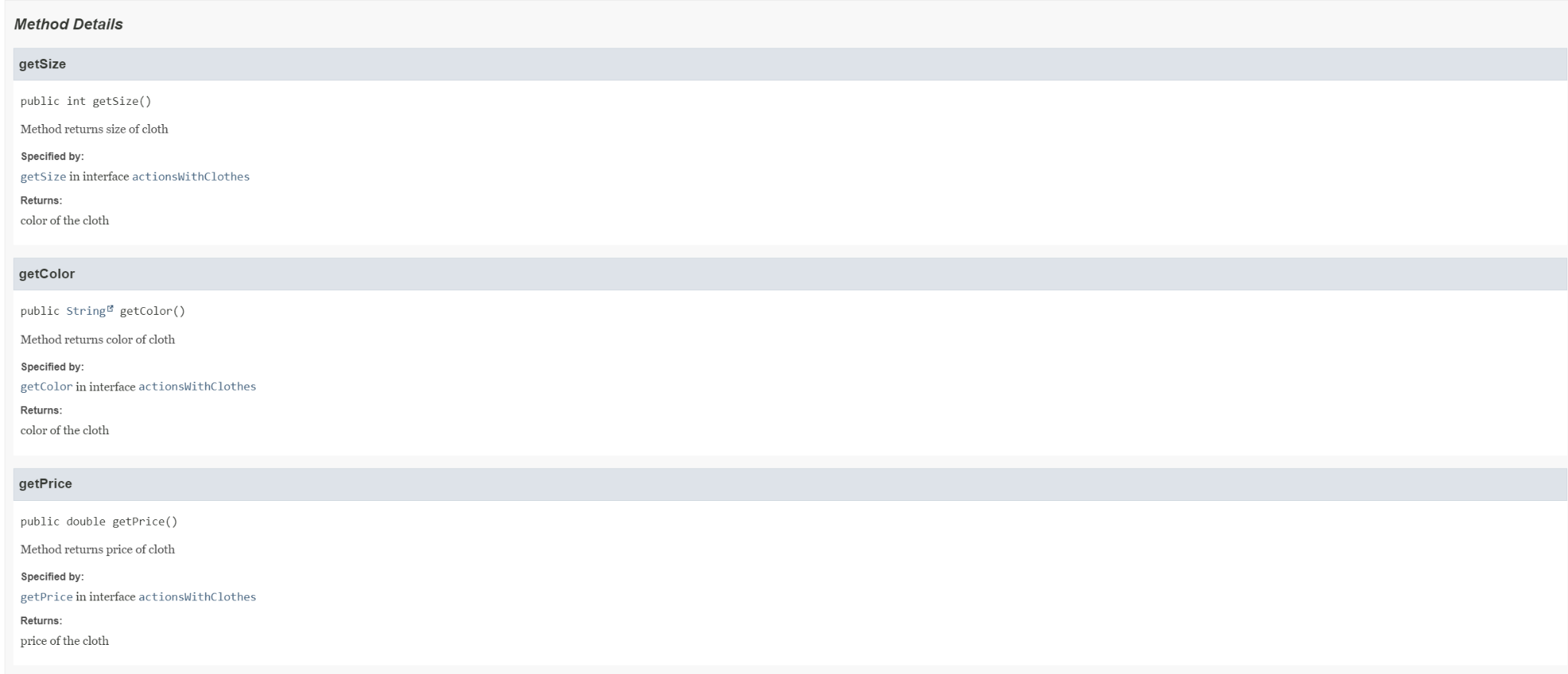
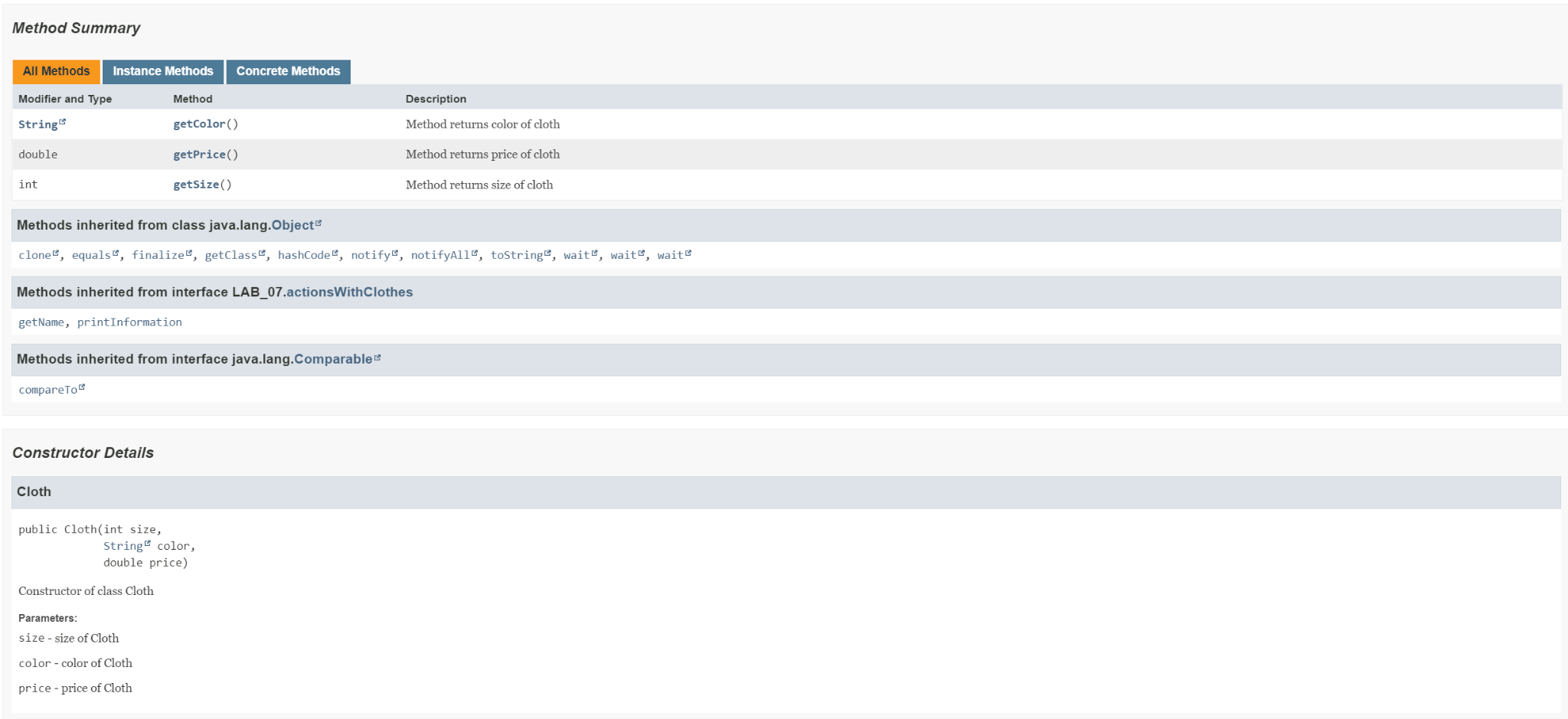
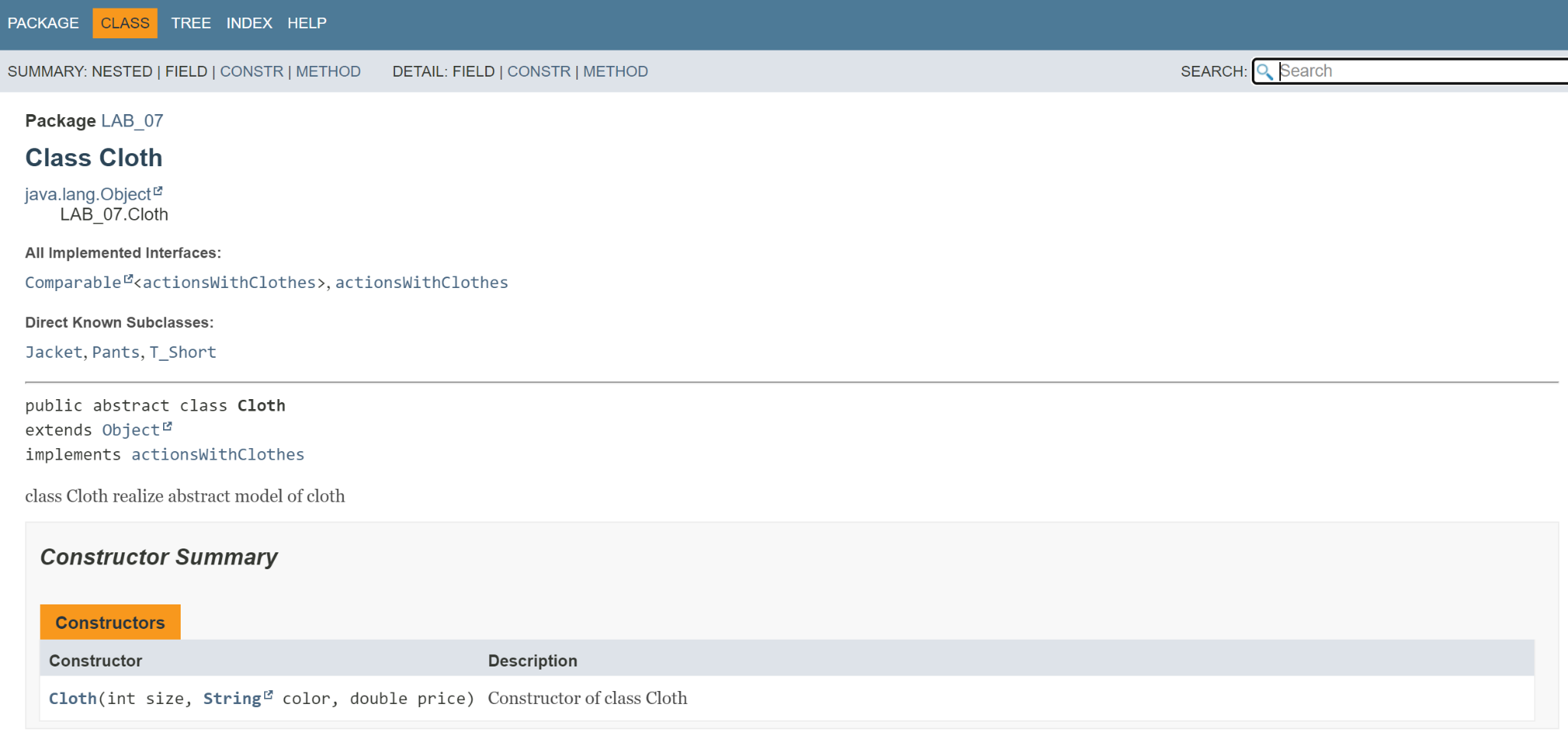
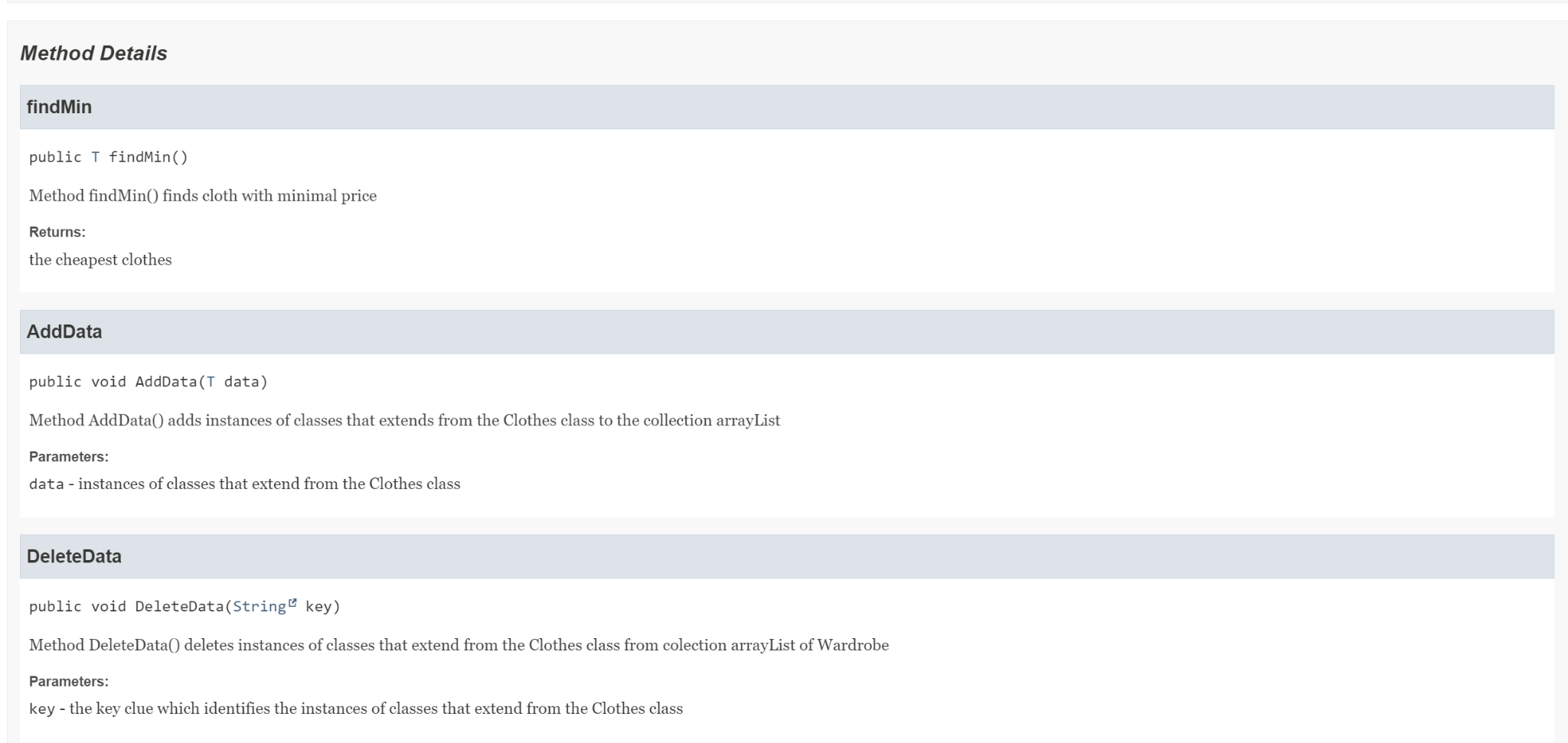
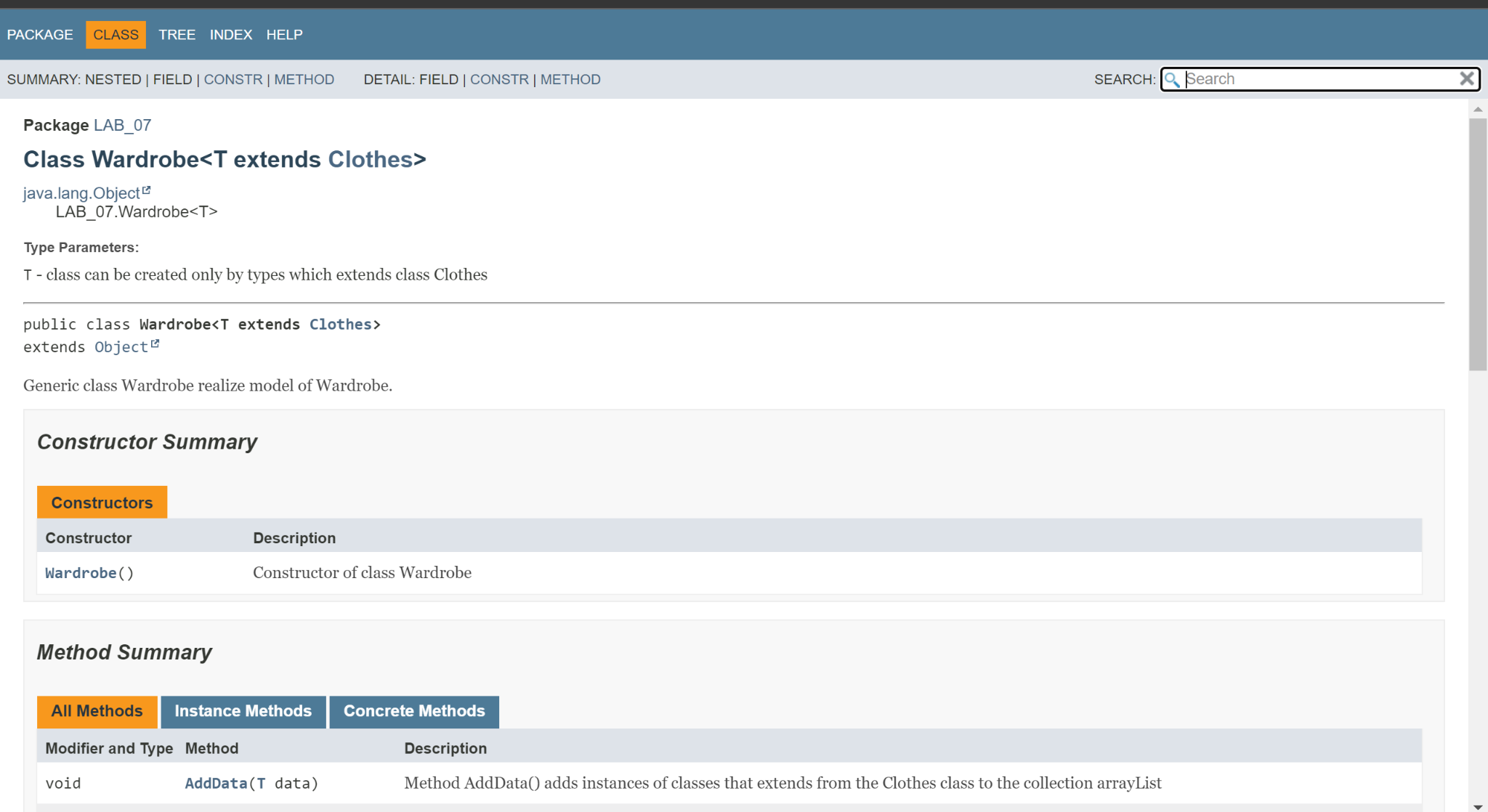
}

**Результати роботи програми:**

****

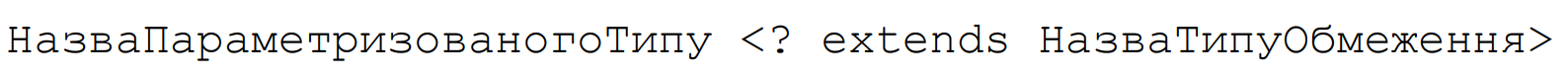
****

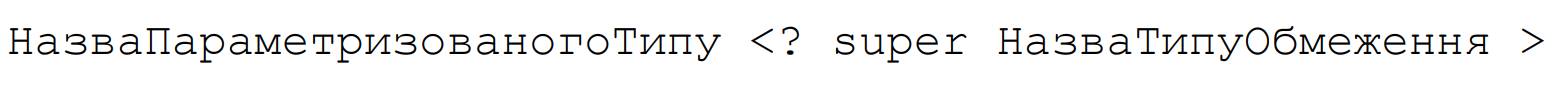
**Фрагмент згенерованої документації**

****

**Відповіді на контрольні запитання**

1. Синтаксис встановлення обмеження підтипу для підстановок має такий вигляд:



Синтаксис встановлення обмеження супертипу для підстановок має такий вигляд: 

1. Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї 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.

**Висновок:** Я оволодів навиками параметризованого програмування мовою Java.