

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

Гапонова Дарина

Прийняв:

доцент кафедри ЕОМ

Іванов Ю. С.

Мета роботи: оволодіти навиками параметризованого програмування мовою Java.

Завдання (варіант № 4)

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 елемента, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розміщуються 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 Main.java

```
package ki306.haponova.lab6;

/**
 * Class Main demonstrates generics work
 *
 * @author Haponova Darina
 */
public class Main {
    /**
     * Method driver
     * @param args
     */
    public static void main(String[] args) {
        Conveyor <? super Product> conveyor = new Conveyor<>();
        conveyor.putProduct(new Cellphone("Cellphone number 1", 200, true));
        conveyor.putProduct(new AudioPlayer("mp3 player dragon red" , 130,
false));
        conveyor.putProduct(new Cellphone("Cellphone3000" , 999, false));
        conveyor.putProduct(new AudioPlayer("King wave cosmos" , 300, false));

        Product product = conveyor.getProduct(0);
        product.print();

        product = conveyor.getProduct(2);
        product.checkIfIsDefect();
        conveyor.checkIfProductIsDefect(2);

        Product min = conveyor.getMin();
        System.out.println("\nThe cheapest product on the conveyor is: ");
        min.print();
    }
}
```

Файл Product.java

```
package ki306.haponova.lab6;

/**
 * The Product interface represents a product with a price that can be compared
 * and checked for defects.
 *
 * @author Haponova Darina
 */
public interface Product extends Comparable<Product> {
    /**
     * Gets the price of the product.
     *
     * @return The price of the product.
     */
    public int getPrice();

    /**
     * Checks if the product has any defects.
     */
    public void checkIfIsDefect();

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

Файл AudioPlayer.java

```
package ki306.haponova.lab6;

/**
 * The AudioPlayer class represents a audioPlayer product that implements the
 * Product interface.
 *
 * @author Haponova Darina
 */
public class AudioPlayer implements Product{
    private String name;
    private int price;
    private boolean isDefect;

    /**
     * Constructs an AudioPlayer object with a given name, price, and defect
     * status.
     *
     * @param name The name of the audio player.
     * @param price The price of the audio player.
     * @param isDefect True if the audio player has defects; otherwise, false.
     */
    public AudioPlayer(String name, int price, boolean isDefect) {
        this.name = name;
        this.price = price;
        this.isDefect = isDefect;
    }

    /**
     * Compares this audio player's price to another product's price.
     *
     */
}
```

```

    * @param product The product to compare to.
    * @return A negative integer, zero, or a positive integer if this product's
    price is less than, equal to, or greater than the other product's price.
    */
    public int compareTo(Product product) {
        Integer p = price;
        return p.compareTo(product.getPrice());
    }

    /**
     * Gets the price of the audio player.
     *
     * @return The price of the audio player.
     */
    @Override
    public int getPrice() {
        return price;
    }

    /**
     * Checks if the audio player has defects and prints a corresponding
     message.
     */
    @Override
    public void checkIfIsDefect() {
        if (isDefect) System.out.println("AudioPlayer " + name + " has a
defect");
        else System.out.println("AudioPlayer " + name + " has no defects");
    }

    /**
     * Prints detailed information about the audio player.
     */
    @Override
    public void print() {
        System.out.println("AudioPlayer{" +
            "name='" + name + '\'' +
            ", price=" + price +
            ", isDefect=" + isDefect +
            '}');
    }
}

```

Файл Cellphone.java

```

package ki306.haponova.lab6;

/**
 * The Cellphone class represents a cellphone product that implements the
 Product interface.
 *
 * @author Haponova Darina
 */
public class Cellphone implements Product{
    private String name;
    private int price;
    private boolean isDefect;

    /**
     * Constructs a Cellphone object with a given name, price, and defect
     status.
     *
     * @param name The name of the cellphone.
     * @param price The price of the cellphone.
     */
}

```

```

    * @param isDefect True if the cellphone has defects; otherwise, false.
    */
    public Cellphone(String name, int price, boolean isDefect) {
        this.name = name;
        this.price = price;
        this.isDefect = isDefect;
    }

    /**
     * Compares this cellphone's price to another product's price.
     *
     * @param product The product to compare to.
     * @return A negative integer, zero, or a positive integer if this product's
     price is less than, equal to, or greater than the other product's price.
     */
    public int compareTo(Product product) {
        Integer p = price;
        return p.compareTo(product.getPrice());
    }

    /**
     * Gets the price of the cellphone.
     *
     * @return The price of the cellphone.
     */
    @Override
    public int getPrice() {
        return price;
    }

    /**
     * Checks if the cellphone has defects and prints a corresponding message.
     */
    @Override
    public void checkIfIsDefect() {
        if (isDefect) System.out.println("Cellphone " + name + " has a defect");
        else System.out.println("Cellphone " + name + " has no defects");
    }

    /**
     * Prints detailed information about the cellphone.
     */
    @Override
    public void print() {
        System.out.println("Cellphone{" +
            "name='" + name + '\'' +
            ", price=" + price +
            ", isDefect=" + isDefect +
            '}');
    }
}

```

Файл Conveyor.java

```

package ki306.haponova.lab6;

import java.util.ArrayList;

/**
 * The Conveyor class represents a conveyor belt for handling products of type T
 (must implement the Product interface).
 *
 * @param <T> The type of product the conveyor can handle.
 *
 * @author Haponova Darina

```

```

*/
public class Conveyor<T extends Product> {
    private ArrayList<T> arr;

    /**
     * Constructs a Conveyor object with an empty product list.
     */
    public Conveyor() {
        arr = new ArrayList<T>();
    }

    /**
     * Adds a product to the conveyor belt.
     *
     * @param product The product to add.
     */
    public void putProduct(T product) {
        arr.add(product);
        System.out.print("Product added: ");
        product.print();
    }

    /**
     * Retrieves a product from the conveyor belt by its index.
     *
     * @param i The index of the product to retrieve.
     * @return The product at the specified index.
     */
    public T getProduct(int i) {
        return arr.get(i);
    }

    /**
     * Finds and returns the product with the minimum price on the conveyor
     belt.
     *
     * @return The product with the minimum price, or null if the conveyor is
     empty.
     */
    public T getMin() {
        if (!arr.isEmpty()) {
            T min = arr.get(0);
            for (T product: arr) {
                if (product.compareTo(min) < 0) {
                    min = product;
                }
            }
            return min;
        }
        else return null;
    }

    /**
     * Checks if a product on the conveyor belt at the specified index has
     defects and prints a corresponding message.
     *
     * @param i The index of the product to check for defects.
     */
    public void checkIfProductIsDefect(int i) {
        arr.get(i).checkIfIsDefect();
    }
}

```

Результат виконання програми

```
Run Main x
"\"C:\\Program Files\\Java\\jdk-17.0.3.1\\bin\\java.exe\" \"-javaagent:C:\\Program Files\\JetBrains\\Intelli
Product added: Cellphone{name='Cellphone number 1', price=200, isDefect=true}
Product added: AudioPlayer{name='mp3 player dragon red', price=130, isDefect=false}
Product added: Cellphone{name='Cellphone3000', price=999, isDefect=false}
Product added: AudioPlayer{name='King wave cosmos', price=300, isDefect=false}
Cellphone{name='Cellphone number 1', price=200, isDefect=true}
Cellphone Cellphone3000 has no defects
Cellphone Cellphone3000 has no defects

The cheapest product on the conveyor is:
AudioPlayer{name='mp3 player dragon red', price=130, isDefect=false}

Process finished with exit code 0
```

Фрагмент згенерованої документації

PACKAGE CLASS TREE INDEX HELP

PACKAGE DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES SEARCH Search

Package ki306.haponova.lab6

package ki306.haponova.lab6

All Classes and Interfaces	Interfaces	Classes
Class	Description	
AudioPlayer	The AudioPlayer class represents a audioPlayer product that implements the Product interface.	
Cellphone	The Cellphone class represents a cellphone product that implements the Product interface.	
Conveyor<T extends Product>	The Conveyor class represents a conveyor belt for handling products of type T (must implement the Product interface).	
Main	Class Main demonstrates generics work	
Product	The Product interface represents a product with a price that can be compared and checked for defects.	

Відповіді на контрольні запитання

1. Дайте визначення терміну «параметризоване програмування».
 - це підхід до програмування, що дозволяє створювати класи і методи, які можна використовувати з різними типами даних, надаючи більшу гнучкість і безпеку типів у програмах.
2. Розкрийте синтаксис визначення простого параметризованого класу.
 - ```
public class НазваКласу<параметризованийТип> {
 // Тіло класу
}
```
3. Розкрийте синтаксис створення об'єкту параметризованого класу.
  - ```
НазваКласу<перелікТипів> зміннаКласу = new  
НазваКласу<перелікТипів>(параметри);
```
4. Розкрийте синтаксис визначення параметризованого методу.
 - ```
public <параметризованийТип> типПовернення
назваМетоду(параметри) {
 // Тіло методу
}
```
5. Розкрийте синтаксис виклику параметризованого методу.
  - ```
(НазваКласу|НазваОб'єкту).<перелікТипів>назваМетоду(параметри);
```
6. Яку роль відіграє встановлення обмежень для змінних типів?
 - дозволяє заборонити використання деяких типів або вимагати, щоб тип підставлений за замовчуванням був підкласом або реалізував певний інтерфейс.
7. Як встановити обмеження для змінних типів?
 - за допомогою ключового слова `extends` для суперкласу або інтерфейсу, від яких має походити реальний тип.
8. Розкрийте правила спадкування параметризованих типів.
 - Всі класи, створені з параметризованого класу, незалежні один від одного.
 - Зазвичай немає залежності між класами, створеними з різними параметрами типів.
9. Яке призначення підстановочних типів?
 - використовуються для забезпечення безпеки типів при використанні параметризованих класів та методів. Вони дозволяють визначити, які типи можна використовувати замість параметризованих типів.
10. Застосування підстановочних типів.
 - `<?>` (unbounded wildcard) дозволяє читати об'єкти з колекції без змінення її.
 - `<? extends Тип>` (bounded wildcard) дозволяє читати об'єкти з колекції, але забороняє додавання в неї нових об'єктів.
 - `<? super Тип>` (lower bounded wildcard) дозволяє додавати об'єкти в колекцію, але забороняє їх читання.

Висновок

У ході виконання даної лабораторної роботи, я отримала важливі навички параметризованого програмування мовою Java. Ознайомилась з різними аспектами мови, такими як використання параметрів у методах, створення та використання класів та інтерфейсів.