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*
Олесько Б. А.
Прийняв:
доцент кафедри ЕОМ
Іванов Ю. С.

Львів – 2023

Мета: оволодіти навиками параметризованого програмування мовою Java.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Параметризоване програмування є аналогом шаблонів у C++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів. Користувачів параметризованого програмування можна поділити на 3 рівні кваліфікації:

1. ті, що користуються готовими класами;

2. ті, що користуються готовими класами і вміють виправляти помилки, що виникають при їх використанні;

3. ті, що пишуть власні параметризовані класи.

Для успішного застосування параметризованого програмування слід навчитися розуміти помилки, що генерує середовище при компіляції програми, що можуть стосуватися, наприклад, неоднозначності визначення спільного суперкласу для всіх переданих об’єктів. З іншої сторони необхідно передбачити захист від присвоєння об’єктів параметризованого класу, що містять об’єкти підкласу об’єктам параметризованого класу, що містять об’єкти суперкласу і дозволити зворотні присвоєння. Для вирішення цієї проблеми у мові Java введено так звані підстановочні типи. Це далеко не всі «підводні камені», що виникають при застосуванні параметризованого програмування.

Варіант 11 y=ctg(x)/tg(x)

```java
package KI306.Olesko.lab6;

public interface Item extends Comparable<Item>{
            /**
             * Get thePrice of the item.
             *
             * @return The Price of the item as a double value.
             */
            public double getPrice();
            public void  get();
            /**
             * Perform an action Price the item.
             */
            public void buy();

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

```java
package KI306.Olesko.lab6;

/**
```

```java
 * Головний клас програми, який демонструє використання класу ShoppingCenter
 * для зберігання та роботи з предметами (Item) у торговому центрі.
 */
public class MainApp {

    /**
     * Головний метод програми, де демонструється використання класу ShoppingCenter
     * та робота з предметами.
     *
     * @param args Аргументи командного рядка (не використовуються в цій програмі).
     */
    public static void main(String[] args) {
        // Створення екземпляра класу ShoppingCenter з можливістю зберігання предметів типу
Item
        ShoppingCenter<? super Item> test0 = new ShoppingCenter<>();
        // Додавання різних предметів у торговий центр
        test0.putItem(new test2("Список Шинлера", 100.83));
        test0.putItem(new test2("Atomic Habits", 2.95));
        test0.putItem(new test2("Atomic Habits", 2.95));
        test0.putItem(new test2("Atomic Habits", 2.95));
        test0.putItem(new test2("Atomic Habits", 2.95));
        System.out.println(test1.nn);
        System.out.println(test2.m);
        // Отримання предмету за індексом та виклик методів print та buy
        //Item item = test0.getItem(2);
        //item.print();

        //item = test0.getItem(3);
        //item.buy();

        // Отримання найдорощого предмету та виведення інформації про нього
        Item max = test0.getMax();
        System.out.println("\nThe price item in the bedside table is:");
        max.print();
    }
}
```

```java
package KI306.Olesko.lab6;

import java.util.ArrayList;

/**
 * Клас ShoppingCenter представляє торговий центр для зберігання та роботи з предметами (Item).
 *
 * @param <T> Тип предметів, які можна зберігати у торговому центрі.
 */
public class ShoppingCenter<T extends Item> {

  private ArrayList<T> array;

  /**
   * Конструктор класу ShoppingCenter, який ініціалізує пустий список для зберігання предметів.
   */
  public ShoppingCenter() {
    array = new ArrayList<T>();
  }

  /**
   * Додає предмет до торгового центру та виводить повідомлення про додавання предмету.
   *
   * @param item Предмет, який додається до торгового центру.
   */
  public void putItem(T item) {
    array.add(item);
    //System.out.print("Item added: ");
```

```
        item.print();
    }

    /**
     * Отримує предмет з торгового центру за індексом.
     *
     * @param i Індекс предмету, який потрібно отримати.
     * @return Предмет, який знаходиться за вказаним індексом.
     */
    public T getItem(int i) {
        return array.get(i);
    }

    /**
     * Знаходить та повертає найважчий предмет у торговому центру.
     *
     * @return Найважчий предмет у торговому центру або null, якщо список порожній.
     */
    public T getMax() {
        if (!array.isEmpty()) {
            T max = array.get(0);
            for (int i = 1; i < array.size(); i++) {
                if (array.get(i).compareTo(max) > 0) {
                    max = array.get(i);
                }
            }
            return max;
        }
        return null;
    }

    /**
     * Використовує предмет у торговому центру за індексом та викликає метод buy() цього предмету.
     *
     * @param i Індекс предмету, який потрібно використати.
     */
    public void useItem(int i) {
        array.get(i).buy();
    }
    public void get(T item) {
                item.print();
                        // TODO Auto-generated method stub
                //System.out.println("test2=");
                }
}
```

```
package KI306.Olesko.lab6;

/**
 * Клас `test1` представляє предмет типу Item з визначеними назвою та ціною.
 */
public class test1 implements Item {
    private String name;
    private double price;
    public static int nn = 0;

    /**
     * Конструктор класу `test1`, який ініціалізує назву та ціну предмету.
     *
     * @param n Назва предмету.
     * @param p Ціна предмету.
     */
    public test1(String n, double p) {
```

```
        this.name = n;
        this.price = p;
        nn+=1;

    }

    /**
     * Порівнює цей предмет з іншим предметом за ціною.
     *
     * @param item Інший предмет для порівняння.
     * @return Результат порівняння за ціною.
     */
    @Override
    public int compareTo(Item item) {
        Double p = price;
        return p.compareTo(item.getPrice());
    }

    /**
     * Отримує ціну цього предмету.
     *
     * @return Ціна предмету.
     */
    @Override
    public double getPrice() {
        return price;
    }

    /**
     * Виконує дію покупки для цього предмету та виводить повідомлення про ціну предмету.
     */
    @Override
    public void buy() {
        System.out.println("test 1 price: " + price);
    }

    /**
     * Виводить інформацію про цей предмет, включаючи назву та ціну.
     */
    @Override
    public void print() {

            // System.out.println(""+nn);
    }
            @Override
            public void get() {
                    // TODO Auto-generated method stub
                    // System.out.println("test1="+nn);
            }
}
```

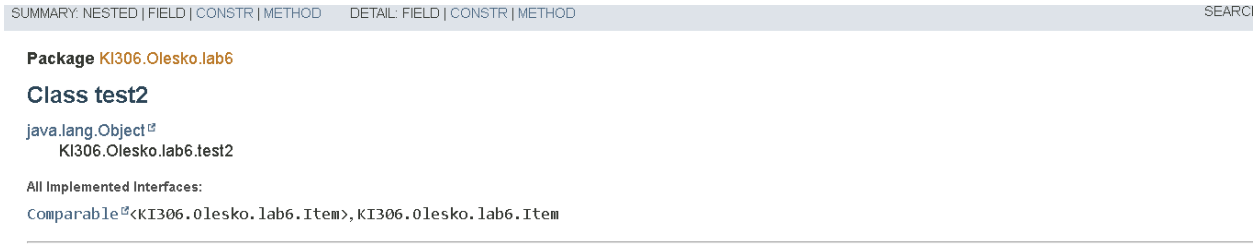
```
package KI306.Olesko.lab6;

/**
 * Клас `test2` представляє предмет типу Item з визначеними назвою та ціною.
 */
public class test2 implements Item {
    private String name;
    private double price;
    public static int m = 0;
;
    /**
     * Конструктор класу `test2`, який ініціалізує назву та ціну предмету.
```

```
     *
     * @param n Назва предмету.
     * @param p Ціна предмету.
     */
    public test2(String n, double p) {
        this.name = n;
        this.price = p;
        m+=1;
    }

    /**
     * Порівнює цей предмет з іншим предметом. Ця реалізація завжди повертає 0, оскільки порівняння за ціною не визначене.
     *
     * @param Інший предмет для порівняння.
     * @return Результат порівняння (завжди 0).
     */
    @Override
    public int compareTo(Item o) {
        return 0;
    }

    /**
     * Отримує ціну цього предмету.
     *
     * @return Ціна предмету.
     */
    @Override
    public double getPrice() {
        return price;
    }

    /**
     * Виконує дію покупки для цього предмету та виводить повідомлення про ціну предмету.
     */
    @Override
    public void buy() {
        System.out.println("test 2 price: " + price);
    }

    /**
     * Виводить інформацію про цей предмет, включаючи назву та ціну.
     */
    @Override
    public void print() {

        //System.out.println(""+n1);
    }
            @Override
            public void get() {
                    // TODO Auto-generated method stub
                     //System.out.println("test2="+m);
            }
}
```

Результат

```
0
5

The price item in the bedside table is:
```

Документація



Відповіді на контрольні запитання

1. Параметризоване програмування - це підхід до програмування, при якому можна створювати загальні класи або методи, які можна налаштовувати для роботи з різними типами даних.
2. class MyClass<T>
   {
   // Внутрішні члени класу
   }
3. MyClass<int> myObject = new MyClass<int>();
4. public void MyMethod<T>(T value)
   {
   // Тіло методу
   }
5. myObject.MyMethod(5);
6. Встановлення обмежень для змінних типів дозволяє обмежити типи, які можна використовувати як параметри при параметризації класів або методів.
7. public class MyClass<T> where T : SomeBaseClass
   {
   // Внутрішні члени класу
   }
8. Правила спадкування параметризованих типів визначають, як параметризовані класи або методи успадковують типові властивості і методи від базових класів або інтерфейсів.

9. Підстановочні типи використовуються для заміни конкретного типу в параметризованих класах або методах на загальний тип.

10. Застосування підстановочних типів може полегшити створення загальних компонентів, які працюють з різними типами даних, зробити код більш читабельним і підтримуваним, а також забезпечити безпеку типів під час компіляції.

Висновок

На даній лабораторній роботі я навчився працювати з кросплатформенні засоби програмування.