Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Звіт

до лабораторної роботи №7

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ»

Виконав:

Студент групи КІ-34

Сенета Б.Р

Прийняв:

Іванов Ю.С.

Львів 2022

Мета: оволодіти навиками параметризованого програмування мовою Java

Виконання роботи

ЗАВДАННЯ

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.

Завдання:

ВАРІАНТИ ЗАВДАННЬ

1. Масив
2. Однозв’язний список
3. Побутовий пакет
4. Конвеєр
5. Двозв’язний список
6. Шафа
7. Трюм корабля
8. Коробка
9. Споруда
10. Пенал
11. Торговий тентр
12. Грузова машина
13. Словник (тип даних)
14. Поличка
15. Вагон
16. Земельна ділянка
17. Кошик
18. Відсік для зброї
19. Сейф
20. Стек
21. Бак для сміття
22. Валіза
23. Хлів
24. Коробка для інструментів

Код програми:

```
package bodyanich7;

import bodyanich7.SafeInfo;

public class Jewelry implements SafeInfo {

	private  String mark;
	private  int Weight;
	private  double Volume;
	private  String Type;

    public Jewelry(String mark, int Weight, double Volume, String Type) {
	  this.mark = mark;
	  this.Weight = Weight;
	  this.Volume = Volume;
	  this.Type = Type;
    }

	  public String getMark() {
		return mark;
	  }

	  public String getSafeType() {
		return Type;
	  }

	  @Override
	  public int compareTo(SafeInfo o) {
		Integer i = Weight;
	    return i.compareTo(o.Weight());
	  }

	  @Override
	  public int Weight() {
		return Weight;
	  }

	  @Override
	  public double Volume() {
		return Volume;
	  }

	  @Override
	  public void print() {
		System.out.println(getMark() + " is safe, it's weight is -> " + Weight() +
				" pounds " + "this safe has volume -> " + Volume() + "sm^3.
This is safe for " + getSafeType() + ".\n" );

	  }
}

------------
package bodyanich7;

import bodyanich7.SafeInfo;

public class WeaponSafe implements SafeInfo {

	private  String mark;
	private  int Weight;
	private  double Volume;
	private  String Type;
```

```
	public WeaponSafe(String mark, int Weight, double Volume, String Type) {
		this.mark = mark;
	this.Weight = Weight;
	this.Volume = Volume;
	this.Type = Type;
	}

	public String getMark() {
		return mark;
	}

	public String getSafeType() {
		return Type;
	}


	@Override
	public int compareTo(SafeInfo s) {
		 Integer i = Weight;
		  return i.compareTo(s.Weight());
	}

	@Override
	public int Weight() {
		return Weight;
	}

	@Override
	public double Volume() {
		return Volume;
	}

	@Override
	public void print() {
		System.out.println(getMark() + " is safe, it's weight is -> " + Weight() +
				" pounds " + "this safe has volume -> " + Volume() + "sm^3.
This is safe for " + getSafeType() + "." + "\n");

	}

}

---------------

package bodyanich7;

import java.util.ArrayList;


public class Safe<T extends SafeInfo> {
	private ArrayList<T> arr;

    public Safe() {
        arr = new ArrayList<T>();
    }

//find minimal material weight
    public T findMinWeight() {
        if (!arr.isEmpty()) {
            T min = arr.get(0);
            for (int i = 1; i < arr.size(); i++) {
                if (arr.get(i).compareTo(min) < 0)
                    min = arr.get(i);
```

```
            }
            return min;
        }
        return null;
    }

    public void WriteData(T data) {
        arr.add(data);
        System.out.println("Element added:");
        data.print();
    }

    public void DeleteData(int x) {
        arr.remove(x);
    }
}
```

```
package bodyanich7;

public interface SafeInfo extends Comparable<SafeInfo> {
       public int Weight();
       public double Volume();
       public void print();
}
```

```
package bodyanich7;

import bodyanich7.Safe;
import bodyanich7.SafeInfo;
import bodyanich7.WeaponSafe;
import bodyanich7.Jewelry;

public class SafeAllInfo {

       public static void main(String[] args) {
              Safe<? super SafeInfo> safe = new Safe<SafeInfo>();
              safe.WriteData(new WeaponSafe("Burgas", 55, 344.2, "For Pistols"));
              safe.WriteData(new WeaponSafe("Liverty", 123, 621.4, "For Rifles"));
              safe.WriteData(new Jewelry("Targo", 10, 100.2, "circlets"));
              safe.WriteData(new Jewelry("Robyr", 15, 150.0, "pendants"));
              SafeInfo smallestweight = safe.findMinWeight();
        System.out.print("\t\t\t\tMININMAL WEIGHT IS IN ELEMENT OF MY PROJECT: \n");
        smallestweight.print();

       }

}
```

Результат роботи програми:

```
Element added:
Burgas is safe, it's weight is -> 55 pounds this safe has volume -> 344.2sm^3. This is safe for For Pistols.

Element added:
Liverty is safe, it's weight is -> 123 pounds this safe has volume -> 621.4sm^3. This is safe for For Rifles.

Element added:
Targo is safe, it's weight is -> 10 pounds this safe has volume -> 100.2sm^3. This is safe for circlets.

Element added:
Robyr is safe, it's weight is -> 15 pounds this safe has volume -> 150.0sm^3. This is safe for pendants.

                              MININMAL WEIGHT IS IN ELEMENT OF MY PROJECT:
Targo is safe, it's weight is -> 10 pounds this safe has volume -> 100.2sm^3. This is safe for circlets.
```

Відповіді на КЗ

1. Параметризоване програмування є аналогом шаблонів у C++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів.
2. Параметризований клас – це клас з однією або більше змінними типу. Синтаксис оголошення параметризованого класу:
   [public] class НазваКласу {
   …
   }
3. GenericClass <String, Integer> obj = new GenericClass<String, Integer> ();
4. (НазваКласу|НазваОб’єкту).[<Переліт типів>] НазваМетоду(параметри);
5. Модифікатори<параметризованийТип{,параметризованийТип}>типПовернення назваМетоду(параметри);
6. Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, 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. Синтаксис оголошення параметризованого методу з обмеженнями типів: Модифікатори <параметризований тип extends обмежуючийТип {& обмежуючий тип} {, параметризований тип extends обмежуючийТип {& обмежуючий тип} } > типПовернення назваМетоду(параметри);

8. 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. Проте у цьому випадку можна зробити помилки, які генеруватимуть виключення на етапі виконання програми.

   3. Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи. В цьому відношенні вони не відрізняються від звичайних класів.

   Наприклад, ArrayList<T> реалізує інтерфейс List<T>. Це значить, що ArrayList<SubClass> можна перетворити у List<SubClass>. Але ArrayList<SubClass> це не ArrayList<SupClass> і не List<SupClass>, де SubClass – підклас суперкласу SupClass.
9. – 10. 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. Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

Висновок: оволодів навиками параметризованого програмування мовою Java.