

Міністерство освіти і науки України Національний
університет «Львівська політехніка»



Звіт

до лабораторної роботи №7

З дисципліни: «Кросплатформенні засоби програмування»

На тему: «Параметризоване програмування»

Варіант 15

Виконав:
Ст. групи КІ-34
Ольховик О.С.

Прийняв:
к.т.н., доцент
Іванов Ю.С.

Львів 2022

Мета: оволодіти навиками параметризованого програмування мовою Java

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.

ВАРІАНТ ЗАВДАННЯ

15. Вагон

Код роботи

Main.java

```
import KI34.Olkhovik.Lab7.*;

public class Main {

    public static void main(String[] args) {
        Carriage<? super Filling> carriage = new Carriage<Filling>();
        carriage.AttachCarriage(new Cargo("CargoType", 5000, 6000));
        carriage.AttachCarriage(new Cargo("CargoType2", 4000, 6000));
        carriage.AttachCarriage(new Passengers(60, 56, 80));
        Filling res = carriage.findMax();
        System.out.print("The biggest weight is: ");
        System.out.println(res.getAllInfo());
    }
}
```

Cargo.java

```
package KI34.Olkhovik.Lab7;

public class Cargo implements Filling {

    private int carryingCapacity;
    private int capacity;
    private String CargoType;

    public Cargo(String CargoType, int capacity, int carryingCapacity) {
```

```

        this.CargoType = CargoType;
        this.carryingCapacity = carryingCapacity;
        if(capacity > this.carryingCapacity)
            this.capacity = this.carryingCapacity;
        else
            this.capacity = capacity;
    }

    public String GetCargoType() {
        return this.CargoType;
    }

    @Override
    public int compareTo(Filling o) {
        Integer s = capacity;
        return s.compareTo(o.getCapacity());
    }

    @Override
    public int getCapacity() {
        return this.capacity;
    }

    @Override
    public String getAllInfo() {
        return "Cargo`s type: " + this.CargoType + "; Total weight : " +
this.capacity ;
    }
}

```

Carriege.java

```

package KI34.Olkhovyk.Lab7;

import java.util.ArrayList;

public class Carriege<T extends Filling> {
    private ArrayList<T> list;
    private int numOfCarriege;

    public Carriege() {
        list = new ArrayList<T>();
        this.numOfCarriege = 0;
    }

    public T findMax() {
        if (!list.isEmpty()) {
            T max = list.get(0);
            for (int i = 1; i < list.size(); i++) {

```

```

        if (list.get(i).compareTo(max) > 0)
            max = list.get(i);
    }
    return max;
}
return null;
}

public void AttachCarriage(T carriage) {
    list.add(carriage);
    this.numOfCarriage++;
    System.out.print("Carriage added: ");
    System.out.println(carriage.getAllInfo());
}

public int getNumOfCarriage(){
    return this.numOfCarriage;
}

public void DetachCarriage(T carriage) {
    list.remove(carriage);
    this.numOfCarriage--;
}
}

```

Filling.java

```

package KI34.Olkhovyk.Lab7;

public interface Filling extends Comparable<Filling> {
    int getCapacity();

    String getAllInfo();
}

```

Passenger.java

```

package KI34.Olkhovyk.Lab7;

public class Passengers implements Filling {

    private int numOfSits;
    private int passengers;
    private int capacity;
    private int averageWeight;

    public Passengers(int numOfSits, int passengers, int averageWeight) {
        this.numOfSits = numOfSits;
        if (this.passengers > this.numOfSits)
            this.passengers = this.numOfSits;
        else

```

```

        this.passengers = passengers;
        this.averageWeight = averageWeight;

        this.capacity = this.averageWeight * this.passengers;
    }

    @Override
    public int compareTo(Filling o) {
        Integer s = capacity;
        return s.compareTo(o.getCapacity());
    }

    @Override
    public int getCapacity() {
        return this.capacity;
    }

    public double getNumOfPassengers() {
        return this.passengers;
    }

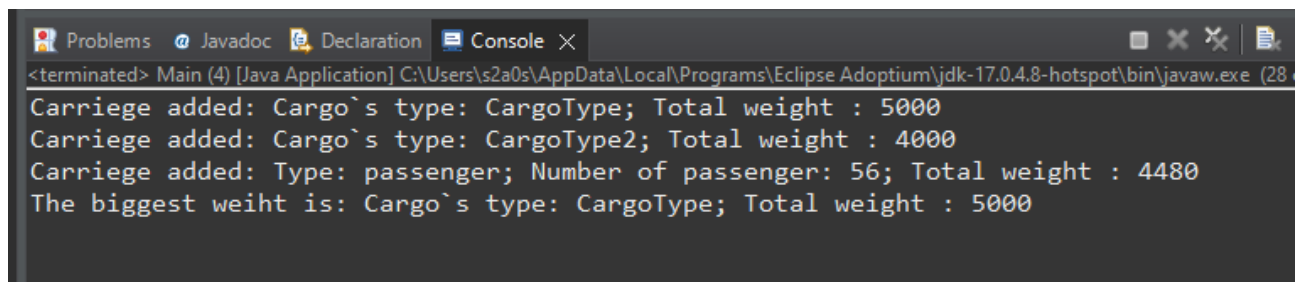
    @Override
    public String getAllInfo() {

        return "Type: passenger; Number of passenger: " + this.passengers + ";
Total weight : " + this.capacity;
    }
}

```

Результат виконання програми

- Консоль



```

<terminated> Main (4) [Java Application] C:\Users\s2a0s\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.4.8-hotspot\bin\javaw.exe (28
Carriege added: Cargo`s type: CargoType; Total weight : 5000
Carriege added: Cargo`s type: CargoType2; Total weight : 4000
Carriege added: Type: passenger; Number of passenger: 56; Total weight : 4480
The biggest weiht is: Cargo`s type: CargoType; Total weight : 5000

```

Відповіді на контрольні запитання:

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 утворенні реального типу з параметризованого типу, наприклад, у методі `main`.

Висновок: я оволодів навиками параметризованого програмування мовою Java.