Міністерство освіти і науки України Національний університет «Львівська політехніка»



Звіт

до лабораторної роботи №7

З дисципліни: «Кросплатформенні засоби програмування» На тему: «Параметризоване програмування»

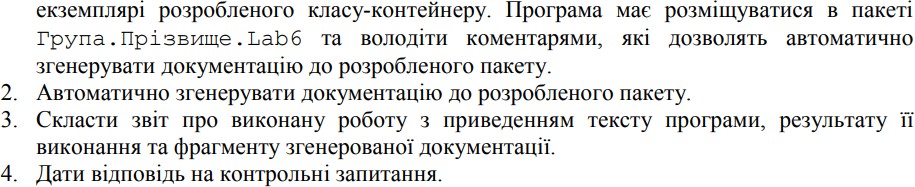
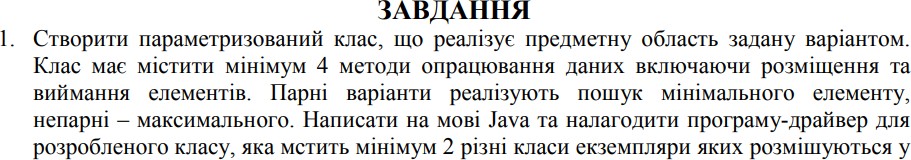
Варіант 15

Виконав: Ст. групи КІ-36 Постоюк В.Р.

Прийняв: к.т.н., доцент Іванов Ю.С.

Львів 2022

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



# ВАРІАНТ ЗАВДАННЯ



**Код роботи**

*Main.java*

import KI34.Olkhovyk.Lab7.\*; public class Main {

public static void main(String[] args) {

Carriege<? super Filling> carriege = new Carriege<Filling>(); carriege.AttachCarriege(new Cargo("CargoType", 5000, 6000));

carriege.AttachCarriege(new Cargo("CargoType2", 4000, 6000));

carriege.AttachCarriege(new Passengers(60, 56, 80)); Filling res = carriege.findMax(); System.out.print("The biggest weiht is: "); System.out.println(res.getAllInfo());

}

}

*Cargo.java*

package KI34.Olkhovyk.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 AttachCarriege(T carriege) { list.add(carriege); this.numOfCarriege++; System.out.print("Carriege added: ");

System.out.println(carriege.getAllInfo());

}

public int getNumOfCarriege(){ return this.numOfCarriege;

}

public void DetachCarriege(T carriege) { list.remove(carriege); this.numOfCarriege--;

}

}

*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() {

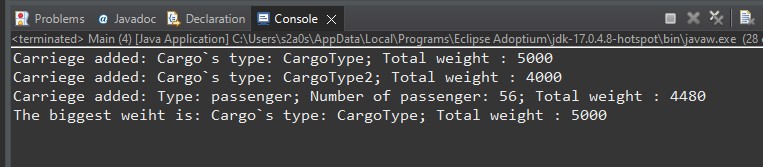
return "Type: passenger; Number of passenger: " + this.passengers + "; Total weight : " + this.capacity;

}

}

# Результат виконання програми

* Консоль



# Відповіді на контрольні запитання:

1. Параметризоване програмування є аналогом шаблонів у С++. 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 {

…

}

1. GenericClass <String, Integer> obj = new GenericClass<String, Integer> ();
2. (НазваКласу|НазваОб’єкту).[<Переліт типів>] НазваМетоду(параметри);
3. Модифікатори<параметризованийТип{,параметризованийТип}>типПоверне ння назваМетоду(параметри);
4. 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.
5. Синтаксис оголошення параметризованого методу з обмеженнями типів: Модифікатори <параметризований тип extends обмежуючийТип {& обмежуючий тип} {, параметризований тип extends обмежуючийТип {& обмежуючий тип} } > типПовернення назваМетоду(параметри);
6. 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.
   1. 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.

* 1. 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.

1. – 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***:* я оволодів навиками параметризованого програмування мовою Java.