Національний університет «Львівська Політехніка»

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра електронних обчислювальних машин



Звіт

Про виконання лабораторної роботи №6

З дисципліни «Кросплатформлені засоби програмування»

**Виконав:**

студент групи КІ-305

Ключко Д.С.

**Перевірив:**

Доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Тема:** Параметризоване програмування.

**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. 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.

**Завдання згідно варіанту:**

**Варіант: 12**



**Виконання:**

**Код програми:**

package lab6Klyuchko305;

/\*\*

\* Class TruckApp realize program to lab work 6

\*

\* @author Klyuchko Dmytro

\* @version 1.1

\* @since version 1.0

\*/

import java.util.\*;

import java.io.\*;

public class TruckApp {

public static void main(String[] args) {

Truck <? super Specification> truck = new Truck <Specification>();

TechnicalSpecifications technicalSpec = new TechnicalSpecifications("Scania", 6, 2, 6.56, 310, "v8");

truck.AddSpecification(technicalSpec);

TechnicalSpecifications technicalSpec1 = new TechnicalSpecifications("DAF", 8, 4, 5.78, 235, "v6");

truck.AddSpecification(technicalSpec1);

technicalSpec.Wheelformula();

System.out.println("Повна вага: " + technicalSpec.FullWeight() + " т");

ParametersforDistance parametersDist = new ParametersforDistance(100,60,345);

truck.AddSpecification(parametersDist);

System.out.println("Час витрачений на подолання дистанції: " + parametersDist.Time()+" год");

System.out.println("Середня витрата пального: " + parametersDist.Fuelconsumption()+" л");

ParametersforDistance parametersDist1 = new ParametersforDistance(247,76,428);

truck.AddSpecification(parametersDist1);

System.out.println("Час витрачений на подолання дистанції: " + parametersDist1.Time()+" год");

System.out.println("Середня витрата пального: " + parametersDist1.Fuelconsumption() + " л");

technicalSpec1.Wheelformula();

System.out.println("Повна вага: " + technicalSpec1.FullWeight() +" т");

Specification res = truck.findMin();

System.out.println("\n");

System.out.println("Мінімальне значення:" + "\n");

res.print();

}

}

/\*\*

\* The Truck class manages a collection of objects that implement the Specification interface.

\* It allows adding specifications, finding the minimum tank capacity, and deleting data from the collection.

\* @param <T> The type parameter constrained to implement the Specification interface.

\*/

class Truck <T extends Specification>{

private ArrayList<T> arr;

/\*\*

\* Constructor to initialize the ArrayList of specifications.

\*/

public Truck()

{

arr = new ArrayList<T>();

}

/\*\*

\* Finds the specification with the minimum tank capacity in the collection.

\* @return The specification with the minimum tank capacity or null if the collection is empty.

\*/

public T findMin()

{

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;

}

/\*\*

\* Adds a new specification to the collection.

\* Prints a confirmation message and the added specification details.

\* @param specification The specification to add.

\*/

public void AddSpecification(T specification)

{

arr.add(specification);

System.out.println("\n"+"Елемент додано: ");

specification.print();

}

/\*\*

\* Deletes a specification from the collection based on its index.

\* @param i The index of the specification to remove.

\*/

public void DeleteData(int i)

{

arr.remove(i);

}

}

/\*\*

\* The Specification interface defines methods that specifications must implement.

\* It requires implementing classes to provide tank capacity information and printing functionality.

\*/

interface Specification extends Comparable<Specification> {

/\*\*

\* Gets the tank capacity of a specification.

\* @return The tank capacity as an integer.

\*/

int getTankCapacity();

/\*\*

\* Prints details of a specification.

\*/

void print();

}

/\*\*

\* The TechnicalSpecifications class represents technical details of a truck and implements the Specification interface.

\*/

class TechnicalSpecifications implements Specification

{

// Attributes representing various technical details of a truck

private String truckbrand;

private int wheels;

private int occasion;

private double weight;

private int tankcapacity;

private String engine;

/\*\*

\* Constructor to initialize TechnicalSpecifications with truck details.

\* @param pTruckbrand The brand of the truck.

\* @param pWheels The number of wheels.

\* @param pOccasion The number of occasion (drive) wheels.

\* @param pWeight The weight of the truck.

\* @param pTankcapacity The tank capacity of the truck.

\* @param pEngine The type of engine.

\*/

public TechnicalSpecifications(String pTruckbrand, int pWheels, int pOccasion, double pWeight, int pTankcapacity, String pEngine ) {

truckbrand = pTruckbrand;

wheels = pWheels;

occasion = pOccasion;

weight = pWeight;

tankcapacity = pTankcapacity;

engine = pEngine;

}

// Getters and setters for the attributes

// ...

public String getTruckbrand()

{

return truckbrand;

}

public void setTruckbrand(String Truckbrand)

{

truckbrand = Truckbrand;

}

public int getWheels()

{

return wheels;

}

public void setWheels(int Wheels)

{

wheels = Wheels;

}

public int getOccasion()

{

return occasion;

}

public void setOccasion(int Occasion)

{

occasion = Occasion;

}

public double getWeight()

{

return weight;

}

public void setWeight(double Weight )

{

weight = Weight;

}

public int getTankCapacity()

{

return tankcapacity;

}

public void setTankCapacity(int TankCapacity)

{

tankcapacity = TankCapacity;

}

public String getEngine()

{

return engine;

}

public void setEngine(String Engine)

{

engine = Engine;

}

public void Wheelformula()

{

System.out.println("Колісна формула: "+wheels + "x" + occasion);

}

/\*\*

\* Method to calculate the full weight of the truck.

\* @return The calculated full weight of the truck.

\*/

public double FullWeight()

{

double temp = (tankcapacity \* 0.83) / 1000;

return weight + temp;

}

/\*\*

\* Compares the tank capacity of this truck specification to another Specification object.

\* @param p The Specification object to compare.

\* @return An integer representing the comparison result.

\*/

public int compareTo(Specification p)

{

Integer s = tankcapacity;

return s.compareTo(p.getTankCapacity());

}

/\*\*

\* Prints the technical details of the truck.

\*/

public void print()

{

System.out.println("Марка вантажівки: " + truckbrand);

System.out.println("Кількість колес: " + wheels);

System.out.println("Кількість колес приводу:" + occasion);

System.out.println("Вага: " + weight + " т");

System.out.println("Ємність бака: "+ tankcapacity + " л");

System.out.println("Двигун: "+ engine);

}

}

/\*\*

\* The ParametersforDistance class represents parameters related to a truck's journey and implements the Specification interface.

\*/

class ParametersforDistance implements Specification

{

// Attributes representing journey parameters

private int distance;

private int speed;

private int tankcapacity ;

/\*\*

\* Constructor to initialize ParametersforDistance with journey parameters.

\* @param pDistance The distance of the journey.

\* @param pSpeed The speed of the truck.

\* @param pTankCapacity The tank capacity of the truck.

\*/

public ParametersforDistance(int pDistance, int pSpeed, int pTankCapacity)

{

distance = pDistance;

speed = pSpeed;

tankcapacity = pTankCapacity;

}

// Getters and setters for the attributes

// ...

public int getDistance() {

return distance;

}

public void setDistance(int Distance) {

distance = Distance;

}

public int getSpeed()

{

return speed;

}

public void setSpeed(int Speed) {

speed = Speed;

}

public int getTankCapacity()

{

return tankcapacity;

}

public void setTankCapacity(int TankCapacity)

{

tankcapacity = TankCapacity;

}

/\*\*

\* Calculates the time taken for the journey.

\* @return The calculated time taken for the journey.

\*/

public double Time()

{

return distance / speed;

}

/\*\*

\* Calculates the fuel consumption for the journey.

\* @return The calculated fuel consumption for the journey.

\*/

public double Fuelconsumption() {

double temp = ((double)distance / tankcapacity);

return temp;

}

/\*\*

\* Compares the tank capacity of this journey specification to another Specification object.

\* @param p The Specification object to compare.

\* @return An integer representing the comparison result.

\*/

public int compareTo(Specification p)

{

Integer s = tankcapacity;

return s.compareTo(p.getTankCapacity());

}

/\*\*

\* Prints the journey parameters.

\*/

public void print()

{

System.out.println("Відстань поїздки: "+ distance +" км");

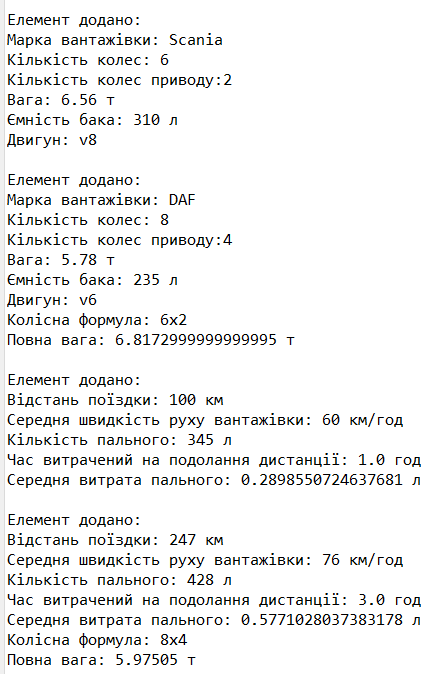
System.out.println("Середня швидкість руху вантажівки: "+ speed +" км/год");

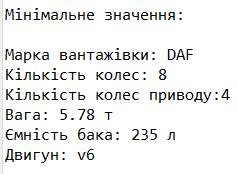
System.out.println("Кількість пального: "+ tankcapacity + " л");

}

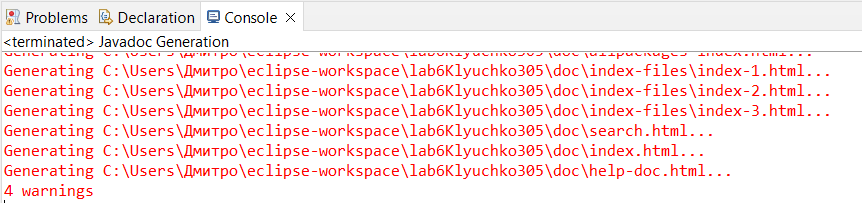
}

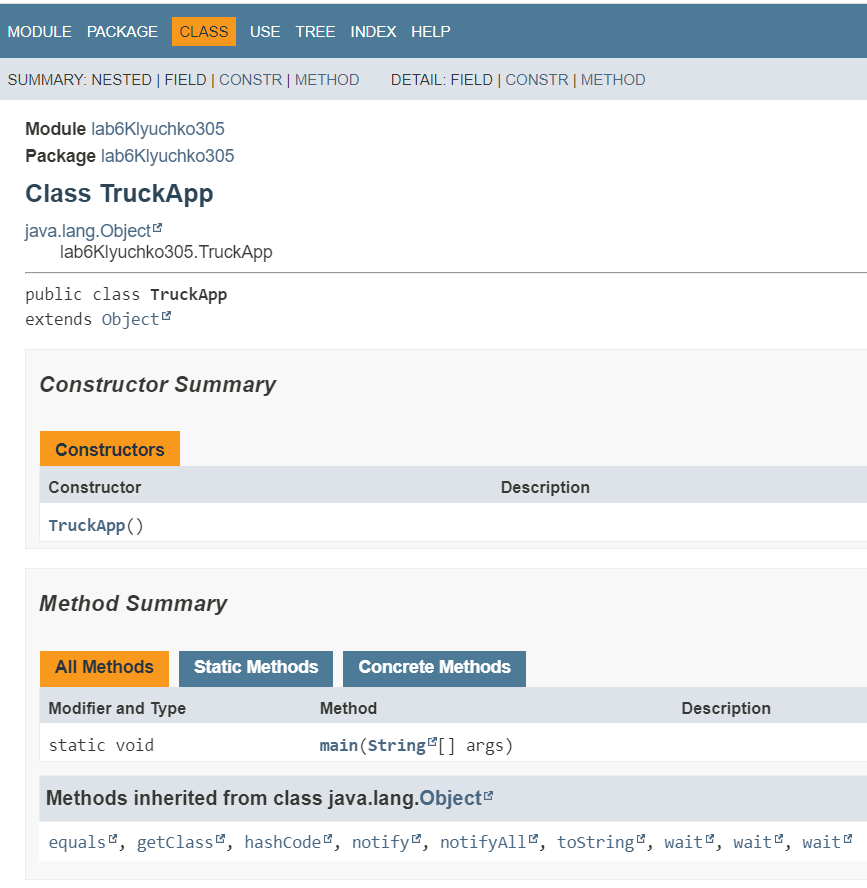
**Результат роботи програми у консолі та файлі:**

****

****

**Фрагмент генерації документації:**

****

****

**КОНТРОЛЬНІ ПИТАННЯ**

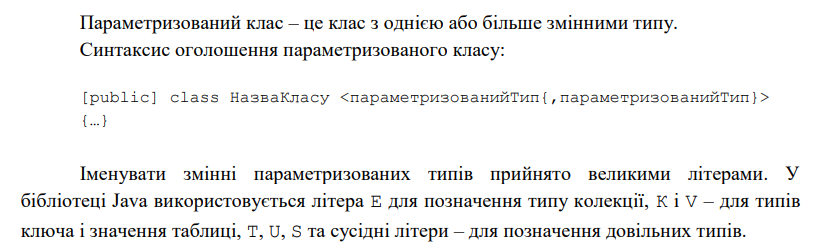
1. Дайте визначення терміну «параметризоване програмування».

**Відповідь:**

Параметризоване програмування є аналогом шаблонів у С++. 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.

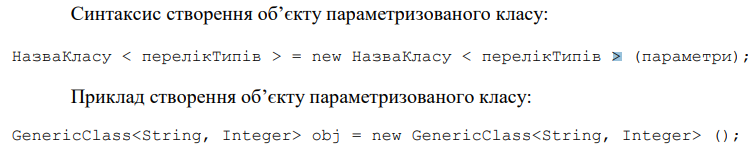
1. Розкрийте синтаксис визначення простого параметризованого класу.

**Відповідь:**

****

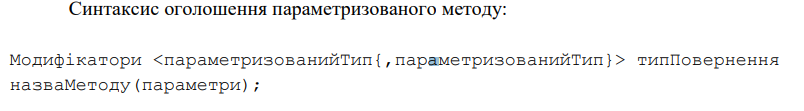
1. Розкрийте синтаксис створення об’єкту параметризованого класу.

**Відповідь:**

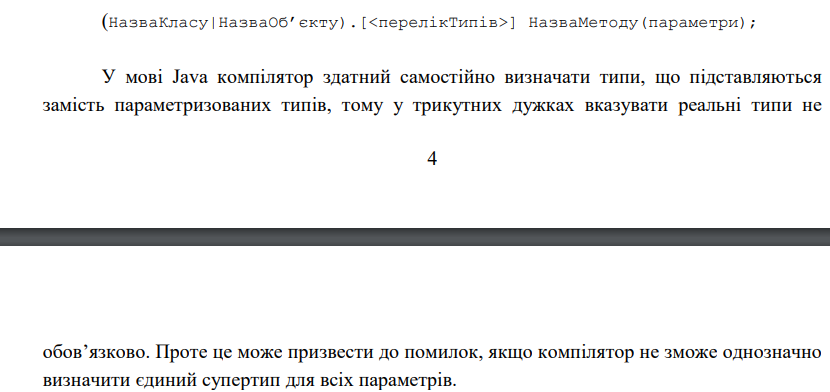
****

1. Розкрийте синтаксис визначення параметризованого методу.

**Відповідь:**

****

1. Розкрийте синтаксис виклику параметризованого методу.**Відповідь:**

****

1. Яку роль відіграє встановлення обмежень для змінних типів?

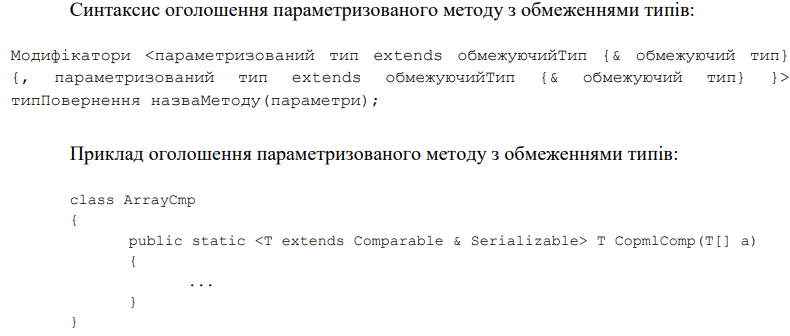
**Відповідь:**

Встановлення обмежень для змінних типів грає важливу роль у програмуванні з декількох причин:

1. Безпека даних: Обмеження можуть забезпечити захист від некоректного використання даних та неправильного доступу до них. Встановлення обмежень може допомогти уникнути помилок, пов'язаних з некоректними значеннями змінних.
2. Ефективність та оптимізація: Встановлення обмежень може допомогти оптимізувати використання пам'яті та обчислювальних ресурсів. Наприклад, використання коротших типів даних (наприклад, byte замість int) для зберігання значень, які можуть бути в межах діапазону byte, може зменшити обсяг пам'яті, потрібний для зберігання цих даних.
3. Читабельність та розуміння коду: Обмеження можуть слугувати як документація для програмістів, які використовують ваш код, надаючи інформацію про очікувані значення та діапазони, що можуть бути використані для конкретних змінних.
4. Підвищення стабільності та надійності програм: Перевірка обмежень може допомогти уникнути виникнення помилок, таких як переповнення пам'яті, витік пам'яті або інші види помилок, пов'язаних з некоректними значеннями змінних.
5. Як встановити обмеження для змінних типів?

**Відповідь:**

Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово extends і вказати один суперклас, або довільну кількість інтерфейсів (через знак &), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова extends.



1. Розкрийте правила спадкування параметризованих типів.

**Відповідь:**

У Java параметризовані типи (generics) дозволяють створювати класи та методи, які можна параметризувати типами даних. Коли ми працюємо зі спадкуванням параметризованих типів, виникають певні правила:

1. Підтримка підтипів: Параметризовані типи підтримують підтипи. Це означає, що якщо B є підтипом A, то MyClass<B> є підтипом MyClass<A>.
2. Не можна створювати інстанції типу з параметром: Не можна створити новий об'єкт параметризованого типу напряму, наприклад, new T() або new MyClass<T>(). Це обмеження пояснюється відсутністю відомостей про тип T на етапі компіляції.
3. Спільне використання методів і поля: При спадкуванні параметризованого класу або інтерфейсу всі методи і поля базового класу також доступні в похідних класах.
4. Передача параметризованих типів в конструктори: Підклас може передавати параметризовані типи до конструктора свого суперкласу.
5. Спеціалізація параметризованого типу: В деяких випадках можливе спеціалізування (розширення) параметризованого типу. Наприклад, MyClass<T> може бути розширено класом MyClassSpecialized<T, U>.
6. Перевірка типів за під час компіляції: Параметризовані типи перевіряються за типом під час компіляції, що забезпечує безпеку типів.

ризованих типів.

1. Яке призначення підстановочних типів?

**Відповідь:**

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.

1. Застосування підстановочних типів.

**Відповідь:**

Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;

2. обмеження супертипу;

3. необмежені підстановки.

**Висновок:** на цій лабораторній роботі я оволодів навиками параметризованого програмування мовою Java. Створив параметризований клас, який працює з двома класами, екземпляри якого розміщуються у списку цього класу в програмі-драйвері.