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-304

Мох М. П.

**Прийняв:**

Олексів М.В.

**Львів – 2023**

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

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.

**Силка на GitHub:** https://github.com/MarianMoh/LabsMohKI-304

**Варіант 12:** Грузова машина

Main.java

package KI304.Moh.Lab6;  
  
*/\*\*  
 \* The {@code Main} class demonstrates the usage of the {@code Truck}, {@code FoodTruck}, and {@code DangerousTruck} classes.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-27  
 \*/*public class Main {  
  
 */\*\*  
 \* The main method demonstrates the creation and manipulation of FoodTruck and DangerousTruck instances using the Truck class.  
 \*  
 \* @param args The command line arguments (not used in this example).  
 \*/* public static void main(String[] args) {  
  
 // Example with FoodTruck  
  
 Truck<FoodTruck> foodTrucks = new Truck<>();  
 foodTrucks.addVehicle(new FoodTruck(5, true));  
 foodTrucks.addVehicle(new FoodTruck(10, false));  
 foodTrucks.addVehicle(new FoodTruck(7, true));  
  
 Vehicle maxVehicle = foodTrucks.findMaxCarryingCapacity();  
 Vehicle minVehicle = foodTrucks.findMinCarryingCapacity();  
  
 System.*out*.println("\n--FoodTruck with min carrying capacity--");  
 maxVehicle.print();  
  
 System.*out*.println("\n--FoodTruck with max carrying capacity--");  
 minVehicle.print();  
  
 // Example with DangerousTruck  
  
 Truck<DangerousTruck> dangerousTrucks = new Truck<>();  
 DangerousTruck deletedTruck = new DangerousTruck(9, 2);  
  
 dangerousTrucks.addVehicle(new DangerousTruck(8, 5));  
 dangerousTrucks.addVehicle(deletedTruck);  
 dangerousTrucks.addVehicle(new DangerousTruck(5, 4));  
  
 dangerousTrucks.removeVehicle(deletedTruck);  
  
 Vehicle maxVehicle2 = dangerousTrucks.findMaxCarryingCapacity();  
 Vehicle minVehicle2 = dangerousTrucks.findMinCarryingCapacity();  
  
 System.*out*.println("\n\n--DangerousTruck with min carrying capacity--");  
 maxVehicle2.print();  
  
 System.*out*.println("\n--DangerousTruck with max carrying capacity--");  
 minVehicle2.print();  
 }  
}

Vehicle.java

package KI304.Moh.Lab6;  
  
*/\*\*  
 \* The {@code Vehicle} interface represents a generic vehicle that can be used for transportation.  
 \* It extends the {@code Comparable} interface to enable comparison between vehicles based on their carrying capacity.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-27  
 \*/*public interface Vehicle extends Comparable<Vehicle> {  
  
 */\*\*  
 \* Gets the carrying capacity of the vehicle.  
 \*  
 \* @return The carrying capacity of the vehicle.  
 \*/* float getCarryingCapacity();  
  
 */\*\*  
 \* Prints information about the vehicle.  
 \* The specific details of the printed information depend on the implementation of this method in concrete classes.  
 \*/* void print();  
}

Truck.java

package KI304.Moh.Lab6;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* The {@code Truck} class represents a truck that can carry vehicles.  
 \* It is parameterized with a type {@code T} that extends the {@code Vehicle} interface,  
 \* indicating the type of vehicles that can be carried by the truck.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-27  
 \*  
 \* @param* <*T*> *The type of vehicles that can be carried by the truck, must extend the {@code Vehicle} interface.  
 \*/*public class Truck<T extends Vehicle> {  
  
 */\*\*  
 \* The list of vehicles carried by the truck.  
 \*/* private List<T> arr;  
  
 */\*\*  
 \* Constructs a new instance of the {@code Truck} class with an empty list of vehicles.  
 \*/* public Truck() {  
 this.arr = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Adds a vehicle to the list of vehicles carried by the truck.  
 \*  
 \* @param vehicle The vehicle to be added to the truck.  
 \*/* public void addVehicle(T vehicle) {  
 arr.add(vehicle);  
 }  
  
 */\*\*  
 \* Removes a vehicle from the list of vehicles carried by the truck.  
 \*  
 \* @param vehicle The vehicle to be removed from the truck.  
 \*/* public void removeVehicle(T vehicle) {  
 arr.remove(vehicle);  
 }  
  
 */\*\*  
 \* Finds and returns the vehicle with the minimum carrying capacity among the vehicles carried by the truck.  
 \*  
 \* @return The vehicle with the minimum carrying capacity, or {@code null} if the list is empty.  
 \*/* public T findMinCarryingCapacity() {  
 if(arr != null) {  
 T min = arr.get(0);  
 for(int i = 1; i < arr.size(); i++) {  
 if(min.compareTo(arr.get(i)) > 0){  
 min = arr.get(i);  
 }  
 }  
 return min;  
 } else {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Finds and returns the vehicle with the maximum carrying capacity among the vehicles carried by the truck.  
 \*  
 \* @return The vehicle with the maximum carrying capacity, or {@code null} if the list is empty.  
 \*/* public T findMaxCarryingCapacity() {  
 if(arr != null) {  
 T max = arr.get(0);  
 for(int i = 1; i < arr.size(); i++) {  
 if(max.compareTo(arr.get(i)) < 0){  
 max = arr.get(i);  
 }  
 }  
 return max;  
 } else {  
 return null;  
 }  
 }  
}

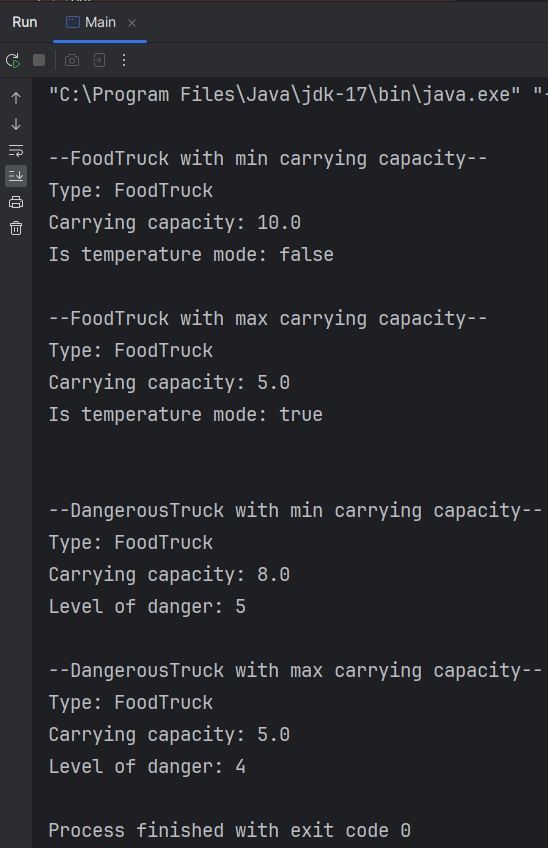
FoodTruck.java

package KI304.Moh.Lab6;  
  
*/\*\*  
 \* The {@code FoodTruck} class represents a specialized type of vehicle designed for transporting food.  
 \* It implements the {@code Vehicle} interface and provides information about its carrying capacity and temperature mode.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-27  
 \*/*public class FoodTruck implements Vehicle {  
  
 */\*\*  
 \* The carrying capacity of the food truck.  
 \*/* private float carryingCapacity;  
  
 */\*\*  
 \* A flag indicating whether the food truck is equipped with a temperature control system.  
 \*/* private boolean isTemperatureMode;  
  
 */\*\*  
 \* Constructs a new instance of the {@code FoodTruck} class with the specified carrying capacity and temperature mode.  
 \*  
 \* @param carryingCapacity The carrying capacity of the food truck.  
 \* @param isTemperatureMode {@code true} if the food truck has temperature control, {@code false} otherwise.  
 \*/* public FoodTruck(float carryingCapacity, boolean isTemperatureMode) {  
 this.carryingCapacity = carryingCapacity;  
 this.isTemperatureMode = isTemperatureMode;  
 }  
  
 */\*\*  
 \* Gets the carrying capacity of the food truck.  
 \*  
 \* @return The carrying capacity of the food truck.  
 \*/* @Override  
 public float getCarryingCapacity() {  
 return carryingCapacity;  
 }  
  
 */\*\*  
 \* Prints information about the food truck, including its type, carrying capacity, and temperature mode.  
 \*/* @Override  
 public void print() {  
 System.*out*.println("Type: FoodTruck"  
 + "\nCarrying capacity: " + carryingCapacity  
 + "\nIs temperature mode: " + isTemperatureMode);  
 }  
  
 */\*\*  
 \* Compares the carrying capacity of the food truck with that of another vehicle.  
 \*  
 \* @param vehicle Another vehicle to compare carrying capacities.  
 \* @return A negative integer, zero, or a positive integer if the carrying capacity of this truck  
 \* is less than, equal to, or greater than the carrying capacity of the specified vehicle.  
 \*/* @Override  
 public int compareTo(Vehicle vehicle) {  
 Float d = carryingCapacity;  
 return d.compareTo(vehicle.getCarryingCapacity());  
 }  
}

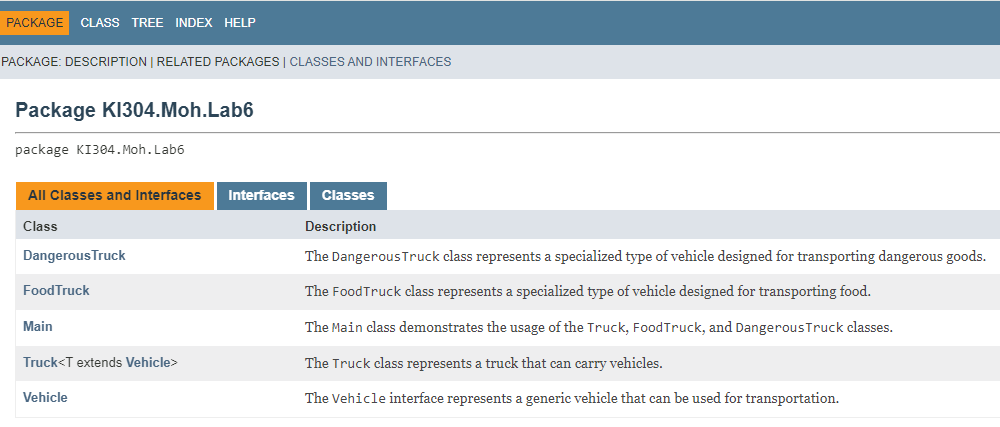
DangerousTruck.java

package KI304.Moh.Lab6;  
  
*/\*\*  
 \* The {@code DangerousTruck} class represents a specialized type of vehicle designed for transporting dangerous goods.  
 \* It implements the {@code Vehicle} interface and provides information about its carrying capacity and level of danger.  
 \*  
 \* @author Marian Moh  
 \* @version 1.0  
 \* @since 2023-11-27  
 \*/*public class DangerousTruck implements Vehicle {  
  
 */\*\*  
 \* The carrying capacity of the dangerous goods truck.  
 \*/* private float carryingCapacity;  
  
 */\*\*  
 \* The level of danger associated with the transported goods.  
 \* Rating scale from 1 to 5.  
 \*/* private int levelOfDanger;  
  
 */\*\*  
 \* Constructs a new instance of the {@code DangerousTruck} class with the specified carrying capacity and level of danger.  
 \*  
 \* @param carryingCapacity The carrying capacity of the dangerous goods truck.  
 \* @param levelOfDanger The level of danger associated with the transported goods.  
 \*/* public DangerousTruck(float carryingCapacity, int levelOfDanger) {  
 this.carryingCapacity = carryingCapacity;  
 this.levelOfDanger = levelOfDanger;  
 }  
  
 */\*\*  
 \* Gets the carrying capacity of the dangerous goods truck.  
 \*  
 \* @return The carrying capacity of the dangerous goods truck.  
 \*/* @Override  
 public float getCarryingCapacity() {  
 return carryingCapacity;  
 }  
  
 */\*\*  
 \* Prints information about the dangerous goods truck, including its type, carrying capacity, and level of danger.  
 \*/* @Override  
 public void print() {  
 System.*out*.println("Type: FoodTruck"  
 + "\nCarrying capacity: " + carryingCapacity  
 + "\nLevel of danger: " + levelOfDanger);  
 }  
  
 */\*\*  
 \* Compares the carrying capacity of the dangerous goods truck with that of another vehicle.  
 \*  
 \* @param vehicle Another vehicle to compare carrying capacities.  
 \* @return A negative integer, zero, or a positive integer if the carrying capacity of this truck  
 \* is less than, equal to, or greater than the carrying capacity of the specified vehicle.  
 \*/* @Override  
 public int compareTo(Vehicle vehicle) {  
 Float d = carryingCapacity;  
 return d.compareTo(vehicle.getCarryingCapacity());  
 }  
}

**Результат виконання програми:**



**Фрагмент згенерованої документації:**



**Відповіді на контрольні запитання:**

1. Дайте визначення терміну «параметризоване програмування».

Параметризоване програмування - це підхід до програмування, де класи та методи можуть приймати типи даних як параметри, що дозволяє створювати більш загальні та змінні структури даних та методи.

1. Розкрийте синтаксис визначення простого параметризованого класу.

class MyGenericClass<T> {

// Код класу з використанням параметра T

}

1. Розкрийте синтаксис створення об’єкту параметризованого класу.

MyGenericClass<Integer> obj = new MyGenericClass<>();

1. Розкрийте синтаксис визначення параметризованого методу.

<T> void myGenericMethod(T value) {

// Код методу з використанням параметра T

}

1. Розкрийте синтаксис виклику параметризованого методу.

MyClass.<Integer>myGenericMethod(42); // Виклик методу з параметром Integer

1. Яку роль відіграє встановлення обмежень для змінних типів?

Встановлення обмежень для змінних типів дозволяє обмежити допустимий тип даних, який можна використовувати як параметр типу.

1. Як встановити обмеження для змінних типів?

<T extends Number> void myMethod(T value) { /\* ... \*/ }

1. Розкрийте правила спадкування параметризованих типів.

Правила спадкування параметризованих типів включають здатність успадковувати параметризовані класи та методи з інтерфейсів або суперкласів та можливість використовувати абстрактні типи даних.

1. Яке призначення підстановочних типів?

Підстановочні типи (wildcards) використовуються для створення більш загальних методів та класів, які можуть приймати об'єкти різних типів.

1. Застосування підстановочних типів.

Застосування підстановочних типів включає використання ? для прийняття об'єктів різних типів, таких як <?> для довільного типу або <? extends T> для обмеження типу до підкласу T або <? super T> для обмеження типу до суперкласу T.

**Висновок**

Ознайомився з використанням параметризованого програмування. Створив клас який реалізує предметну область «множина». Та розробив клас драйвер який показує роботу параметризованого класу контейнера.