Lab Assignment 6: Working with Inheritance, Interfaces and Packages (Part 2)

Course: Advanced Programming (EE423)
Institution: IGEE, Boumerdes University

December 3, 2022

1 Objectives

Within this lab, you will learn to define interfaces in Java and implement them.

2 Assignment

Create an interface called `Vehicle` as follows:

```
interface Vehicle {
 public static double MAX_TANK = 50;
 public abstract void move(double distance);
 public abstract double addFuel(double amount);
 public abstract void print();
 public default void honk(){ System.out.println("Ton Ton"); }
}
```

2.1 Question 1:

Create a class called `Car` that has the following attributes:

```
double fuel; // the amount of fuel left in the tank
double totalDistance; // the total distance covered by the car
double yield; // the number of kilometers a car can cover
              // per liter of fuel
```

2.2 Question 2:

Add a constructor to the class `Car`. The constructor takes one parameter called `yield`. The constructor should set the `fuel` and `totalDistance` to `0`; and set `yield` to the value passed to the constructor.

2.3 Question 3:

Make the class `Car` implement the interface `Vehicle` and implement all the required abstract methods. To make a class implement an interface add **implements Vehicle** as follows

```
class Car implements Vehicle {...}
```

- For the `move` method: it takes as input the distance the car should go, if it does not have enough fuel to run that distance, the car should not run at all and print the following message:

  ```
  "No enough fuel to move that distance"
  ```

  If it has enough fuel, the car should increase its `totalDistance` and also reduce the amount of fuel left in the tank (Use the variable `yield` to compute this) and print

  ```
  "The car moved "+distance+" KM and consumed "+usedFuel+"L of fuel"
  ```

- For the method `addFuel`, it takes as parameter the amount of fuel to fill in the tank and returns the amount of fuel in tank after it was filled. If this tank reaches its max limit (`MAX_TANK`) defined in the interface `Vehicle`, the car should not take more than that amount of fuel. The method `addFuel` should print a message like this:

  ```
  "Amount of fuel: 20 L "
  ```

  where, in this case, 20 is the amount of fuel in the tank after it was added.

- For the method `print`, it should print the information of the car:

  ```
  "Total distance: "+totalDistance+" Remaining fuel: "+fuel+" Yield: "+yield
  ```

2.4 Question 4:

Test you code

```
class TestCar {
 public static void main(String[] args) {
  Vehicle car = new Car(10);
  car.move(20);
  car.print();
  car.addFuel(100);
  car.honk();
  car.move(300);
  car.print();
 }
}
```

By now, if your code is correct, you should get the following output:

```
No enough fuel to move that distance
Total distance: 0.0 Remaining fuel: 0.0 Yield: 10.0
Amount of fuel: 50.0 L
Ton Ton
The car moved 300.0 KM and consumed 30.0L of fuel
Total distance: 300.0 Remaining fuel: 20.0 Yield: 10.0
```

2.5 Question 5:

Create an interface called `VehicleDiesel` that extends the interface `Vehicle`. Add the following abstract method to it: `double co2Emission();`

2.6 Question 6:

Add a static variable called `CO2_EMISSION_DIESEL` inside the interface `VehicleDiesel`. Set its value to `0.25`. This means each 1KM of `totalDistance` corresponds to $0.25M^3$ of CO_2 being emitted

2.7 Question 7:

Create a class called `CarDiesel`. It should have all the variables of the class `Car` and it must also implement the interface `VehicleDiesel`. The `CarDiesel` should use the `move` and `addFuel` methods of the class `Car`, but should implement a new `print` function, the output should be

```
"Total distance: "+totalDistance+" Remaining fuel: "+fuel+" Yield: "+yield
"Total co2-emission: " + co2Emission() + "M3"
```

2.8 Question 8:

Override the function `honk` so that it will print `"Diesel Ton Ton"` and not `"Ton Ton"`

2.9 Question 9:

Add two default methods called `start()` and `stop()`, one should print `"vehicle stared"` and the other should print `"vehicle stopped"`. These methods should be available for all types of vehicles (`Car` and `CarDiesel`).

2.10 Question 10:

Test your code

```
class TestCarDiesel {
 public static void main(String[] args) {
  VehicleDiesel carDiesel = new CarDiesel(20);
```

```
  carDiesel.start();
  carDiesel.move(100);
  carDiesel.addFuel(100);
  carDiesel.move(100);
  carDiesel.honk();
  carDiesel.print();
  carDiesel.stop();
  System.out.println("--------------");
  Vehicle car = new Car(10);
  car.start();
  car.addFuel(100);
  car.move(300);
  car.print();
  car.stop();
 }
}
```

By now, if your code is correct, you should get the following output:

```
Vehicle started
No enough fuel to move that distance
Amount of fuel: 50.0 L
The car moved 100.0 KM  and consumed 5.0L of fuel
Diesel Ton Ton
Total distance: 100.0 Remaining fuel: 45.0 Yield: 20.0
Total co2-emission: 25.0M3
Vehicle stopped
--------------
Vehicle started
Amount of fuel: 50.0 L
The car moved 300.0 KM  and consumed 30.0L of fuel
Total distance: 300.0 Remaining fuel: 20.0 Yield: 10.0
Vehicle stopped
```

3 Conclusion

Interfaces provide a way to define a certain behavior without knowing its current implementation. The class that do implement those interfaces will have to provide the actual code of each method.