Formula 1 Management System

Masterclass in Java

Makar Lavrov
Computer Science Student
makar.lavrov.1@iliauni.edu.ge

Description

Create Formula 1 Management System (FMS) in Java. FMS is widely used software. It can be any complexity. Our example is basic one, which have the following features:

1. paddock for the cars
2. ability to add the car in the paddock
3. ability to remove the car from the paddock
4. ability to print all available cars in the paddock

FMS structure

We will need the following classes for the software:

1. Car – the book itself.
2. FMS – formula 1 management system.
3. FMSTester – the tester class. This class will be used to test our management system.

Class Car	
String	team
String	driver

Class FMS	
List <Book>	pit
void addCar(Car)	
boolean removeCar(Car)	
Void printPit()	

Class Car

The class `Car` should have several fields, including `team` and `driver`. This class can be implemented in the following way:

```
package library;

public class Car {
    private String team, driver;


    public String getTeam() {
        return team;
    }

    public void setTeam(String team) {
        this.team = team;
    }

    public String getDriver() {
        return driver;
```

```
    }

    public void setDriver(String driver) {
        this.driver = driver;
    }

}
```

Pay attention to the setters and getters of the fields. In general, all the fields are private (unless the special requirements are stated) and the access functions are implemented such as setters and getters.

Read about `toString()` function and implement it fo r`Car` class.

Class FMS

The formula 1 management system should have an inner structure for storing cars. The management system should have methods for adding the new cars and removing the cars from paddock. It should have the ability to print the entire pit wall content when needed. The class can be implemented in the following way:

```
import java.util.ArrayList;
import java.util.List;

public class FMS {

    // Mapping with Book and the number of this book in the library
    private List<Car> pit = new ArrayList<>();

    // adds the cars to the paddock
    public void addCar(Car car) {
        pit.add(car);
    }

    // removes the car from the paddock
    public boolean removeCar(Car car) {

        boolean removed = false;

        for (int i = 0; i < pit.size(); i++) {
            Car  b = pit.get(i);
            if (b.getTeam().equals(car.getTeam()) && b.getDriver().equals(car.getDriver())) {
                pit.remove(i);
                removed = true;
                break;
            }
        }

        return removed;
    }

    public void printPit() {

        if (pit.isEmpty()) {
            System.out.println("The paddock is empty");
        } else {
            for (Car b: pit) {
                System.out.println("These cars are in the pit:");
                System.out.println(b.getDriver() + ", " + b.getTeam());
            }
        }
    }

}
```

Pay attention to the usage of the `ArrayList` class, `for` loops for the lists, `break` clause and the `String` object comparison. It is a good point to understand how `Interface` works. Usage of the `boolean` variables can also be observed in this example.

FMS Tester class

Now let's test our management system. First, create some books. Then create LLM and add those books to the library using the LLM. Then try to remove some of the books.

```
public class FMSTester {
    public static void main(String[] args) {
        Car p1 = new Car();
        p1.setTeam("Red Bull F1");
        p1.setDriver("Max Verstappen");


        Car p2 = new Car();
        p2.setTeam("Mclaren F1");
        s2.setSurname("Giorgadze");
        p2.setDriver("Lando Norris");









        FMS oms = new FMS();

        oms.addCar(p1);
        oms.addCar(p2);
        oms.removeCar(p1);



        oms.printPit();

    }

}
```

We print the state of the paddock to check if all the methods are working properly.

)

)