Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Fall 2019-2020

Homework 1 - Formula 1 Simulator

Due date: 06.11.2019, Wednesday, 23:55

1 Introduction

The objective of this assignment is to learn the basics of object oriented design and Java programming language. In this homework you will create a UML Class Diagram for a Formula 1 race simulator and implement your design. In the simulation race cars will go around a track a predetermined amount of laps. The track will consist of turns and straights with different parameters (e.g. turn direction). As the cars go around the track, their tires will degrade and they will start to slow down and at some point they may have to perform a pit-stop to change their tires. At the end of the race, the winning team and a timing table for all cars will be printed on screen. While implementing this homework you should keep the object oriented design principles in mind. In this homework you will not implement a main function, instead we will use the classes you created to run the simulation.

2 Simulation Classes and Details

Your implementation should consist of following classes:

1. **TrackFeature:** This class is used to define a characteristic of a track feature, such as a turn. There are three types of track features you need to implement: `HighSpeedTurn`, `LowSpeedTurn` and `Straight`. Each track feature will contain the following attributes:

   – `int featureNo`: Used to determine the order of track features in a track.

   – `TurnDirection turnDirection`: Used to determine whether a track acually creates a loop. Values of this field will be provided with an Enum.

   – `double distance`: Length of the track feature, it will be used to determine how much time cars spend at this feature.

   – `double roughness`: Used to calculate the impact of the feature on the degradation of tires.

   A track feature's constructor will take above parameters in above order. That is, a new track feature can be created as follows:

   ```
   HighSpeedTurn hst = new HighSpeedTurn(4, TurnDirection.LEFT, 420.0, 16);
   ```

2. **Track:** A track will have a name, and it will consist of `TrackFeautes`. Also the directions that the cars loop around a track will be specified with a boolean value. Overall `Track` will contain the following attributes:

  - `String trackName`: Name of the track.
  - `ArrayList<TrackFeature> featureList`: A list of features that the track consists of.
  - `boolean isClockwise`: Direction of the track.

  Moreover the `Track` will provide the following methods:

  - `public int getTrackLength`: Returns the length of the `featureList` list.
  - `public void addFeature(TrackFeature f)`: With this method the user should be able to append features to the `featureList`.
  - `public TrackFeature getNextFeature()`: Returns the next track feature from the `featureList` in a cyclic manner. On the first call to this method, the first element of `featureList` should be returned. After the last item is returned, the next call should return the first element again.
  - `public boolean isValidTrack()`: Returns a boolean stating the validity of the track. A valid track starts and ends with a `Straight`. And the count of left and right turns should differ exactly by 4 depending on the direction of the track. (in a clockwise track #right_turns = #left_turns + 4)

3. **Tire:** Tires are one of the most important parts of a Formula 1 car and they come in 3 different types: `SoftTire`, `MediumTire`, `HardTire`. Each tire type has different speeds (at zero degradation) and degrade differently. Degradation affects the speed of the tire. `Tire` class will have the following fields:

  - `double speed`: Current speed of the tire.
  - `double degradation`: How much wear the tire has.

  `Tire` class should also implement the following method:
  `public void tick(TrackFeature f)`: With this method a tire will update its `speed` and `degradation` at each tick of the simulation.

4. **Team:** Each team will have the following attributes:

  - `String name`: Name of the team
  - `String[] teamColors`: An array of color names representing the team colors.
  - `ArrayList<Car> carList`: List of cars the team has. A team can have 2 cars at most .

5. **Car:** A car will have the following:

  - `int carNo`: Car number, used to uniquely identify cars.
  - `String driverName`: Name of the driver driving that car.
  - `double totalTime`: Total time the car has spent on the track.
  - `Tire tire`: The tire the car currently has.

  Additionally `Car` should have the following methods:

  - `public void tick(TrackFeature f)`: For each track feature the track has, this method will be called. It needs to update the `totalTime`, and then call the `currentTire.tick(f)`.

6. **Session:** This class is the glue that brings all parts of the simulation together. It holds the following:

  - `Track track`: The track the race will be simulated on.
  - `ArrayList<Team> teamList`: A list of teams (with cars) joining to the race.
  - `int totalLaps`: Number of laps a race consists of. Cars will go around the track `<totalLaps>` number of times.

  It should also provide 3 methods:

- `public void simulate()`: This method simulates the race until it's over. This method should check the validity of the track before starting the simulation and if the track is not valid it should return without simulating the race.
- `public void printTimingTable()`: Prints the driver names and their times in String format. The first element of the table should be the fastest driver and the last element should be the slowest.
- `public void printWinnerTeam()`: Prints a String with the name and colors of the winning team.

3 Class Behaviour and Default Values

3.1 Tire

The tire will degrade at each feature of the track. Moreover the amount it degrades depends on tire type (soft, medium, hard), feature type (high speed turn, low speed turn, straight) and the feature's roughness value. Tire degradation value should start from 0 and increase at each `tick` method call by the following amount.

$$degradation+ = feature\_type\_multplier * feature\_roughness * tire\_type\_multiplier$$

Additionally, the speed of the tire should decrese at each `tick` method call by the following amount.

$$speed- = min(75, degradation) * 0.25$$

Also when the tire speed drops below 100 it should not decrease more. For example, when the tire speed is 103 and after a tick its speed drops to 98, it should stay 98. In other words, a tire shouldn't lose speed when its speed is already below 100.
Multipliers for different features and tires are given in the below table.

Feature Type Multipliers	
HighSpeedTurn	1.55
LowSpeedTurn	1.3
Straight	1.0

Tire Type Multipliers	
SoftTire	1.2
MediumTire	1.1
HardTire	1.0

Default speed values for tires are:

- `SoftTire`: 350
- `MediumTire`: 310
- `HardTire`: 275

3.2 Car

At each tick cars will go through a single track feature. Time spent at a track feature is calculated by dividing the feature's distance by the car's tire's current speed and adding a random value between 0 and 1:

$$time\_spent = feature\_distance/tire_speed() + Random(0, 1.0)$$

As cars go around the track their tires will degrade and when the tire degradation value goes over 70, cars will perform a pit stop. At the pit stop cars will put on brand new tires, but they will change their tire types according to the following table. The pit stop will also cost them 25 seconds.

Old Tire	New Tire
`SoftTire`	`MediumTire`
`MediumTire`	`SoftTire`
`HardTire`	`SoftTire`

Tire degradation calculation should be performed after a car clears a feature. That is, first Car's tick method should be called, then Tire's tick method should be called.

4 Input & Output

During the simulation, your classes will print information to `stdout`. This section explains the outputs and their formats.

1. Before simulating the race you should confirm the validity of the track and print it.
   For valid tracks:
   `Track is valid.Strating simulation on <track_name> for <number_of_laps> laps.`
   For invalid tracks:
   `Track is invalid.Simulation aborted!`

2. After simulation ends, you should print the winning team and their colors.
   `Team <team_name> wins.<team_color_1> [[, <team_color_2>,<team_color_3>, ...]  and <team_color_n>] flags are waving everywhere.`
   Example 1:
   `Team Ferrari wins.Red and Yellow flags are waving everywhere.`
   Example 2:
   `Team Mercedes wins.Silver flags are waving everywhere.`
   Example 3:
   `Team Williams wins.Red, Blue and White flags are waving everywhere.`
   Note: Pay attention to the use of commas and the word ”and”.

3. After the simulation ends, and after printing the winning team and their colors you will print the timing table. This table will contain the driver name, car number and the time it took to complete the race for each car. Drivers should be sorted from fastest to slowest. Milliseconds in the timings should be rounded to three decimal places.
   `<driver_name>(<car_no>):  <hrs>:<mins>:<secs>.<ms>`
   Example:
   ```
   Bottas(77):  00:53:33.766
   Leclerc(16):  00:53:36.367
   Vettel(5):  00:53:40.219
   Hamilton(44):  00:53:54.534
   ```

2 test case files, a parser for them and a main method will be provided to you in order to make it easier for you to check your output.

5 UML Class Diagram

You are required to provide a UML Class Diagram of all the classes (even the ones provided) and the relations between them. Your UML Class Diagram can be either as a pdf or a png file.

6 Specifications

- The codes must be in Java.
- You should be able to grasp the design of the system from homework text and design your solution in this manner.You are also allowed to add your own classes, add fields and methods to the given and your ownclasses.
- Yo must code your classes based on the object oriented principles covered in class
- Your codes will be evaluated with both black and white box testing. For black box texting, makesure you print your outputs in the correct format. White box testing will be done to determine whether you have correctly applied object oriented principles to your solution.

- Put all your source codes under package of your metu username `eXXXXXX` under a Source folder, excluding the parser file and the file containing the main method.
- Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. Thisincludes friends, previous homeworks, or the Internet. The violators will be punished according tothe department regulations.
- Follow the course page on ODTUClass for any updates and clarifications. Please ask your questionson ODTUClass instead of e-mailing if the question does not contain code or solution

7 Submission

Submission will be done via ODTUClass. You will submit a single tar file called "hw1.tar.gz" that contain all your source code in Source folder and your UML Diagram in UML folder. Your folders should look like this:

```
|-- Source
|   |-- Car.java
|   |-- HardTire.java
|   |-- HighSpeedTurn.java
|   |-- LowSpeedTurn.java
|   |-- MediumTire.java
|   |-- Session.java
|   |-- SoftTire.java
|   |-- Straight.java
|   |-- Team.java
|   |-- Tire.java
|   |-- Track.java
|   |-- TrackFeature.java
|   `-- TurnDirection.java
|-- UML
|   `-- UML_class_diagram.pdf
```

All your source codes should be able to compile with following commands.

```
 > tar -xf hw1.tar.gz
 > cd Source\student_username
 > javac *.java
```