COMPUTER SCIENCE 12B (FALL, 2020)
ADVANCED PROGRAMMING TECHNIQUES

PROGRAMMING ASSIGNMENT 6

Need for Speed

Objective

The goal of this assignment is to provide practice with interactions between several objects, to interact with a logger class, and to start to learn how to test your code.

The implementation of this assignment should NOT use any inheritance.

Description

You have been hired by *Antonella Gaming* to write a race car simulation.

The objective of this simulation is to see the number of points that can be accumulated with a selection of race cars of different characteristics (speed and ability to handle collisions, etc.). One can run the game multiple times, see the number of points produced each time, and identify the selection of cars that may maximize the number of points. It is much more efficient to use a simulation instead of actual cars to evaluate these design tradeoffs.

Assume that each car has two adjustable characteristics- Strength and Speed - which determines a car's behavior during a race. There are tradeoffs: a car with high speed races faster but takes more damage during a collision, while a slower car takes less damage in a collision.

Once a car enters the race, it moves along the track in periods called "ticks." Each tick moves the car a fixed distance which is equal to its Speed stat.

Each lap is 100 units (a unit is a measure of simulated distance on the track). Once a car reaches 1000 units (10 laps of 100 units), it enters the finish line and exits the race.

Anytime two cars finish a tick at the same location, they collide and become damaged (remember that 2 cars in the same location on different laps still collide). Collision damage results in a penalty to your speed that is equal to your Strength level multiplied by five. (Cars do NOT take stacking damage penalty. This means that if a car is already damaged, it will NOT have a further reduction of speed in a second collision.)

- i.e. $NewSpeed = Speed - (Strength \cdot 5)$

The racetrack features a pitstop located at the 75th unit of every lap where a car enters each time it is damaged. A car that is in the pitstop is removed from the racetrack, misses two additional turns (ticks) after which it is no longer damaged.

Objects Overview

You will be coding an Object-Oriented solution, where you will need to implement the following classes:

RaceTrack: This class will hold your cars and racetrack components. The racetrack class will also run the race itself as a simulation game.

PitStop: The Pitstop is located at the 75th unit of the lap. Any car that is damaged and passes or lands on the PitStop during a tick enters the pitstop for 2 turns instead. When a car leaves the pitstop, it is no longer damaged. (Two cars leaving the pitstop simultaneously should NOT crash on the 75th unit mark - this is because a car leaving the pitstop should also move right away)

ex (only for RaceCar):

```
Tick 6
RaceCar55/3 has entered the pit stop.
Tick 7
RaceCar33/2 has been damaged.
Tick 8
RaceCar55/3 has exited the pit stop.
```

FinishLine: Cars that complete 10 laps (1000 units) enter the FinishLine and are removed from the race.

ex (only for RaceCar):

```
Tick 26
RaceCar55/3 has finished the race in place 1.
```

RaceCar: Each car has a preset speed and strength. The RaceCar is allowed to have a speed between **55-30** and a strength level between **2-4**. A generic RaceCar (if you pass in no parameters to the constructor) gets a default speed of **40** and a strength of **3**.

FormulaOne: Each FormulaOne has a preset speed and strength. The FormulaOne is allowed to have a speed between **70-30** and a strength level between **3-5**. A generic FormulaOne (if you pass in no parameters to the constructor) get a default speed of **50** and a strength of **4**.

SimulationDriver: This class has a main method to setup and run the race. It instantiates a Racetrack, creates and enters several cars in the race. Then you call your racetrack method to run the race.
We have provided this class for you.

Implementation Details

You may use additional methods in your classes in order to have an organized and modular solution.

Note: Feel free to pass as many parameters in each stared method (*) and constructor method as you see fit. Comment your code very well, as we will be evaluating your logic and your understanding of object-oriented practices.

RaceTrack Class

- `public Racetrack()` — Constructs a new racetrack object to hold the track components (cars, finishline, pitstop).

- `public void setCars(RaceCar[] racecar,FormulaOne[] formulaOnes)` — sets the cars that will participate in your race. No more than 10 cars will participate in a single race.

- `public void tick()` — This method runs one tick. A tick moves every car a set distance equal to its speed. If a car is damaged and passes the 75 unit mark during this tick, it enters the pitstop. Upon exiting the pitstop, it starts at the 75 unit mark and immediately moves according to its speed. After all the cars have moved their set distance, the Tick method should check for collisions and deal with them appropriately.

  ex:

  ```
  Tick 1
  Tick 2
  Tick 3
  ```

- `public void checkCollision()` — This method checks to see if any two cars are located at the same unit. If it finds two cars on the same unit, that is considered a collision, and both cars become damaged. You should only check for collisions after all cars have moved on a given tick. Cars that "pass" each other during a tick should NOT collide. Cars only collide if they end up in the same location after they have moved.

  ex (only for RaceCar):

  ```
  Tick 3
  RaceCar 55/3 has been damaged.
  RaceCar 40/3 has been damaged.
  Tick 4
  RaceCar50/2 has been damaged.
  Tick 5
  ```

  Note that if a car collides with an already damaged car, only the previously undamaged car gets damaged. (This occurs in example tick 4)

- `public void run()` — This method processes all the ticks until the race ends.

`public int calculateScore(int ticks)` — This method returns the games score.
When calculating the score for the race, the following formula should be used:
1000 - 20 per tick +..
+ 150 per RaceCar entered in the race
+ 100 per FormulaOne entered in the race

ex (only for RaceCar):

```
Tick 26
RaceCar40/3 has finished the race in place 3.
```

```
You scored 1300 points.
```

PitStop Class

`public void enterPitStop`***** — This method adds a car into your pitstop. You may want to overload this method to support multiple types of cars, for now.

Implement the exiting of cars from the PitStop however you think is best.

FinishLine Class

- `public void enterFinishLine`***** — This method places a car into the FinishLine - removing it from the race. You may want to overload this method to support multiple types of cars, for now.

- `public boolean finished()` – This method returns true when all cars entered in the race are now in the FinishLine

RaceCar Class

- `public RaceCar()` — The generic constructor that sets this RaceCar's speed to 40 and strength to 3

- `public RaceCar(int speed, int strength)` —have a constructor that sets this RaceCar's speed to any value between 55 and 30, and strength to any value between 2 and 4. If one or both of the values is above the maximum bound, then set the value to the maximum bound. If one or both of the input values are below the minimum bound, set the value to the minimum bound. (e.g.- if someone enters 60 for speed, set the RaceCar's speed to 55. If someone enters 1 for strength, set the RaceCar's strength to 2.)

- `public double getLocation()` — This method returns the car's current unit location. For example, a car on the 30th unit of lap 3 is at location 330.

- `public String toString()` — This method returns a string in the form "TypeOfCarSpeed/Strength" (i.e. for RaceCar with initial speed 40 and strength 3, return "RaceCar40/3")

FormulaOne Class

- `public FormulaOne()` — have one generic constructor that sets the FormulaOne's speed to 50 and strength to 4

- `public FormulaOne(int speed, int strength)` —have a constructor that sets the car's speed to any value between 70 and 30, and strength to any value between 3 and 5. If a value is entered outside of those bounds, you should place them on the closest bound.

- `public double getLocation()` — This method returns the car's current unit location.

- `public String toString()` — This method returns a string in the form "TypeOfCarSpeed/Strength" (i.e. for generic FormulaOne, return "FormulaOne50/4")

Slove?
Error.

Output Styling

In order to simplify your solution, you will be given a logger, **TrackLoggerB**, which will take care of printing to console for you. This logger will be used by the RaceTrack to record all events that occur during the race. This class will handle printing for you; therefore, you should not be printing in RaceTrack.java.

Unit Tests

In your skeleton you will find JUnit tests to run your code against. Make sure your output passes these unit tests in order to conform with our grading tests. For this assignment, you are required to write two simple unit tests, for testing your RaceCar constructor and enterPitStop.

Assignment Guidelines

You should NOT import any libraries in your code. You are allowed to use arrays; remember that you can have arrays of any type of object. The objects in the array can be instances of the specified class or null. Make sure you handle this correctly and avoid potential `NullPointerExceptions`. You should NOT use any Java collections, like `java.util.ArrayList`

If you think that you need extra classes, feel free to implement them. However, make sure to explain why you needed them and what advantages they add to your program.