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

PROGRAMMING ASSIGNMENT 7

Need for Speed

Objective

The goal of this assignment is to have you apply the practices of inheritance and object-oriented programming.

Description

Now that you have written your game to work with two types of cars, you are asked to rewrite your solution in a manner that will allow future programmers to add as many expansion pack vehicles as they wish with ease.

Duplicate your original project and rename the copy firstname_lastnamePA7. Then, make the changes listed below.

1. Introduce a Base Abstract Class

Car: The Car class is an *abstract class* representing a general race contestant.

- `public Car()` — You may add as many constructors as you like for the Car class.

Note: Be smart about what methods you migrate to the Car class. How could you make your life easier if you needed to implement more car classes in the future?

2. Add an additional car class inheriting from Car

SportsCar: Each SportsCar has a preset speed and strength. The SportsCar is allowed to have a speed between **45-20** and a strength level between **1-3**. A generic SportsCar (if you pass in no parameters to the constructor) gets a default speed of **30** and a strength of **2**.

3. Modify the subclasses

Modify the previous classes to use the Car abstract class as a base class.

Make the classes RaceCar, FormulaOne and SportsCar inherit fields and methods from Car. Additionally, subclasses *override* any methods or constructors that need to be changed for their specific car subclass.

4. Change the Remaining Classes

Modify the remaining classes that use three car types. You may also find yourself rewriting or even deleting near duplicated code that you added in Part B of this assignment. This is highly encouraged.

You may end up with more concise code in the classes RaceTrack, Pitstop, FinishLine, because they can handle any Car, and each subclass of Car is responsible for its specific behavior.

Implementation Details

The constructors of the following classes need to be edited to support the Car abstract class.

RaceTrack Class

- `public void setCars()` — sets the cars that will participate in your race. No more than 10 cars will participate in a single race.

PitStop Class

- `public void enterPitStop(Car car)` — This method adds a Car into your pitstop. Because you are now using an abstract class, this method should not be overloaded.

FinishLine Class

- `public void enterFinishLine(Car car)` — This method places a Car into the FinishLine - removing it from the race. This method should not be overloaded.

SportsCar Class

- `public SportsCar()` —have one generic constructor that sets your speed to 30 and strength to 2.

- `public SportsCar(int speed, int strength)` —have a constructor that sets the car's speed to any value between 45 and 20, and strength to any value between 1 and 3. If a value enters a number outside of those bounds, reset the value to the closest bound.

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

- `public String toString()` — This method returns a string in the form "TypeOfCarSpeed/Strength" (i.e. for generic SportsCar, return "SportsCar30/2"

Calculating 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
+ 200 per SportsCar entered in the race

Expected Output

The expected output should not change. This is because introducing the base class is behavior-preserving refactoring, which improves the structure of the code without changing its behavior. If you notice some changes in the expected output, check that your method overriding is working as you expect it to work.

Output Styling (Important):

In order to simplify your solution, you will be given a logger, **TrackLoggerC**, which will take care of printing to the 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.

If you want to be printing in PitStop, FinishLine make sure that you pass the same instance of the logger you use in the RaceTrack to those classes. ALL events in a race should be recorded in the same logger (i.e. PitStop, FinishLine should not have their own loggers).

Important

You should NOT import any libraries in your code. The only data structure that you are allowed to use is 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.