Racing In IUT

Description

Let’s say we want to implement a single player racing game. Since it is single player we need to make a game where the player races against himself using a time tracker. Now we can choose a car, we can tune that chosen car, choose a track and at last race. We can also use the pit stop option to apply the Nitrous oxide system or NOS to the car.

Assumptions

So we are not actually making a game, we are going to build a skeleton of the game using print functions like we do in the design pattern lab. The assumptions are,

1. We don’t actually track the time, we just use a random number generator to generate a random number and convert it to HH::MM::SS format.
2. We assume children of a Car class only show the car information and the parent class Car has start, stop and accelerate methods which contains only print functions nothing else. And this goes for every method of every parent or child class. This means you don’t need to make an actual car or track. Even the race can be a prototype with lots of print functions.
3. You are most welcome to make actual cars, tracks with complex functions but you need to complete all the objectives mentioned in the next section.

Objectives

You need to fulfill the objectives to get a satisfactory mark.

1. You need to incorporate a total of 5 different patterns inside the above scenario. The patterns are strategy, template, singleton, factory and decorator.
2. Car class has engine and turbocharger behavior and we can update this behavior in the run time. Strategy
3. Rii class (main class) has an abstract factory of cars and a factory of track and a player can choose car and track using these factories.
4. In the whole run time of this game, a player can create only one instance of Player class. Singleton
5. While racing we can use pit stops, where we add extra functionally of NOS to our car object at the run time. Decorator
6. In the race class we have a race method which has a generalized steps of action we need to do to complete a race. Template
7. You can implement more types than mentioned in the directions but not less.
8. Use New operator as low as possible in the main class Rii. (Hint : use static methods)

Directions

1. You can have these four cars ToyotaGR86, SubaruBRZ, PorscheBoxster, Ferrari812. Toyota and Subaru reside under the Coupe car category where Porsche and Ferrari fall under Roadster.

2. You can have these three tracks Blue Moon Bay Speedway (USA), BB Raceway (Japan), Circuit de Spa-Francorchamps (Germany).
3. You can have these three engines : v6, v8, v12 and turbochargers : Alpine, Cummins, Honeywell.
4. You can have these two NOS called Resonac and Sema.
5. You can have two types of racing : sprint and circuit. In circuit racing finishing line means the start of the track (since the track is circular) whereas in sprint racing finishing line means end of the track. And both have a start line at the start of the track.
6. You need to take input from a player such as what type of car he is choosing, what type of track and what sort of racing he prefers. The choices should not be hard coded.
7. In the Race class we have race method which is final and can have steps like,
   a) Car.carInfo();
   b) Track.trackInfo();
   c) startCar(Car);
   d) startLine(Track);
   e) accelerateCar(Car);
   f) NOS = pitStop(Car); //NOS is an extended class of Car
   g) applyNos(NOS);
   h) finishLine(Track);
   i) stopCar(NOS);
   j) lapTime(); // print random time

Helper Functions

1. You can import SimpleDateFormat and Random to implement lapTime().
2. You can import Scanner to take input from players.