QCells Project Description

Take-Home Assignment:
https://github.com/sreeram-dev/qcells-assignment/blob/master/car-simulation.pdf

GitHub Link: https://github.com/sreeram-dev/qcells-assignment
ToDo List: https://github.com/sreeram-dev/qcells-assignment/blob/master/todo.md (Added abstract classes to some models, though)
UML File: https://github.com/sreeram-dev/qcells-assignment/blob/master/qcells_uml.xml

---

What were the specific requirements you were asked to solve?

- Build a car factory for two models (possibly the same model with a different configuration) and run a racing simulation.
- Build a parking lot for parking cars with some customisable options (number_of_cars as an option).
- The parking lot can raise an error if the capacity is total or replace an existing car.

What architectural decisions did you need to make in the project?

- One of the basic principles I tried to adhere to is SOLID principles in architectural design.
- I defined the project's structure by functional responsibility, i.e. car module contains code for the model, factory and related code.
- I used interfaces for push and pop operations for a parking lot. I implemented an ordered parking lot where a car is parked at the lowest parking number.
- I used abstract classes and enums (CarRoof, CarType) to break down the functional components of the car into smaller components. AbstractClass allowed me to follow the DRY principle.
- I followed the same paradigm for simulating a car race and defined an interface and a concrete class.
- **I let my tests verify the successful completion of the assignment by executing various scenarios instead of defining a lengthy README.**

What challenges did you encounter during the implementation?

- I needed to familiarise myself with the ABC class in Python to define abstract classes and enforce specific standards. The interface I first used was just informal specifications.
- Choosing between the LRU parking lot or the Ordered parking lot was more complex as the former is expected in any coding assignment, while the latter seemed more real-world. **Why would you park farther away from the entrance?**

- There was a bit of code smell in my first implementation of the car factory and model interface. I could do better and was able to define different car versions of the same model.

Is there anything you would do differently if starting from scratch?

- I would re-think how I coded the simulation class; more work is needed to make it production-ready.
- The IConvertibleSpecialization should be an *ABC* interface.
- Logging could have been better. However, *print* statements do just fine for an assignment.
- The heuristics for replace option feel like a code smell; I need to re-think the Parking Lot interface to support different implementations with the DRY principle.
- I could add a *Builder Class* for building a car instead of initialising the class and setting the properties in *CarFactory*. However, there were already many layers. I did not know about mixing two creational patterns.
- I could have added more unit tests. However, my tests covered all the acceptance scenarios and many integration tests. I verified my car models by asserting each property.

What new features would you like to add, if any?

- I did not add a stop simulation function because the simulation stops when one of the cars reaches 200 mph.
- The simulation class did not consider the velocity when the car roof is open or the number of seats in the car. There was no description of it. The simulation could be closer to a real-world scenario.
- The simulation could support a race between multiple cars and evaluate the results in a separate function.
- I was unfamiliar with VisualStudioCode Dev Containers when I submitted the assignment, but they are very flexible in sharing sandboxed environments to reduce the need for README further. (No Installation Instructions).