Final Project report

Name: Xu Gezheng, Tan Gansheng

Course: Object oriented Software design

Supervisor: Paolo Ballarini

Title: myUber: a car-ride sharing system

18/11/2018

Introduction

The project is an attempt to design and create an Java framework for myUber so that it can be used by numerous customers and drivers as well as the system manager himself(herself). The main difficulty with the creation of such a framework is the fact that the communication between different part of the system, and how to make our framework be extendable. The need for such a system is justified as the use of car-sharing application is a common part of everyday life for most people in urban city. It is likely that the usage of Uber will increase in the future.

For the developer of a more concrete and more realistic myUber application, this framework gives a great amount of hints for the methods that will be needed for user or driver as well as the simulation for the whole car sharing progress. This project also explores the possibility that new types of cars can be added to the system without great change in code, flexibility to set up the simulation zone and simulation time. In addition, we develop this system not entirely depend on the project requirement but taking into account the real-life facts.

Overview of the Project Progress

1. Define core class and design pattern
2. Handwrite the UML, build Github environment to sperate tasks
3. Define attributes
4. Use Junit test to implement methods
5. Simulation
6. CLML – not done yet
7. Evaluation and write the report.

Overview of this report

This report fully describes the project undertaken which is split into five main sections:

1. Introduction and background This introduces the project, its aims, an overview of the work undertaken in the project and an overview of this report.
2. Analysis of the design pattern and UML structure. Further, possible extensions to the basic design requirements are proposed.
3. Implementation - Discussion of the implementation choices taken and the software that was developed. Detailed advantages and drawbacks are given for the implementation.
4. Testing and result - Results of the success of the implementation for two main use cases tests.

5. Conclusions - Analysis of the successes and failures of the project, and discussion of the advances made.

In addition to these main sections there is a number of appendices. These appendices contain:
Appendix A – maybe some usage for certain functions

Background

As stated in the project description in this course, Uber is a ride-sharing system which allows inhabitants of a metropolitan area to get a ride on a car driven by a professional driver. The Uber system consists of several parts including: the cars (which circulate on the metropolitan area), the drivers, the customers (the persons registered to the system and that can book a car ride). Based on the systems requirements and what is like in practice we develop a Java framework, called myUber for representing and managing the Uber ride-sharing system. What's more, myUber system actually takes the shape of mobile application and since java is compatible with Android operation system, thus our framework can have great value for mobile application developer.

Precisely, what distinguishes our project is the fact that we personalize our using area whose center is Paris, with a radius of 20 km. We also personalize our testing time whose unit is second permitting that we test such a system in short time whilst keeping all his functions. what's more, we turn the actual process to java method after deep thought, which can serve as an example of how to aggregate actual process for other developers.

Analysis and design

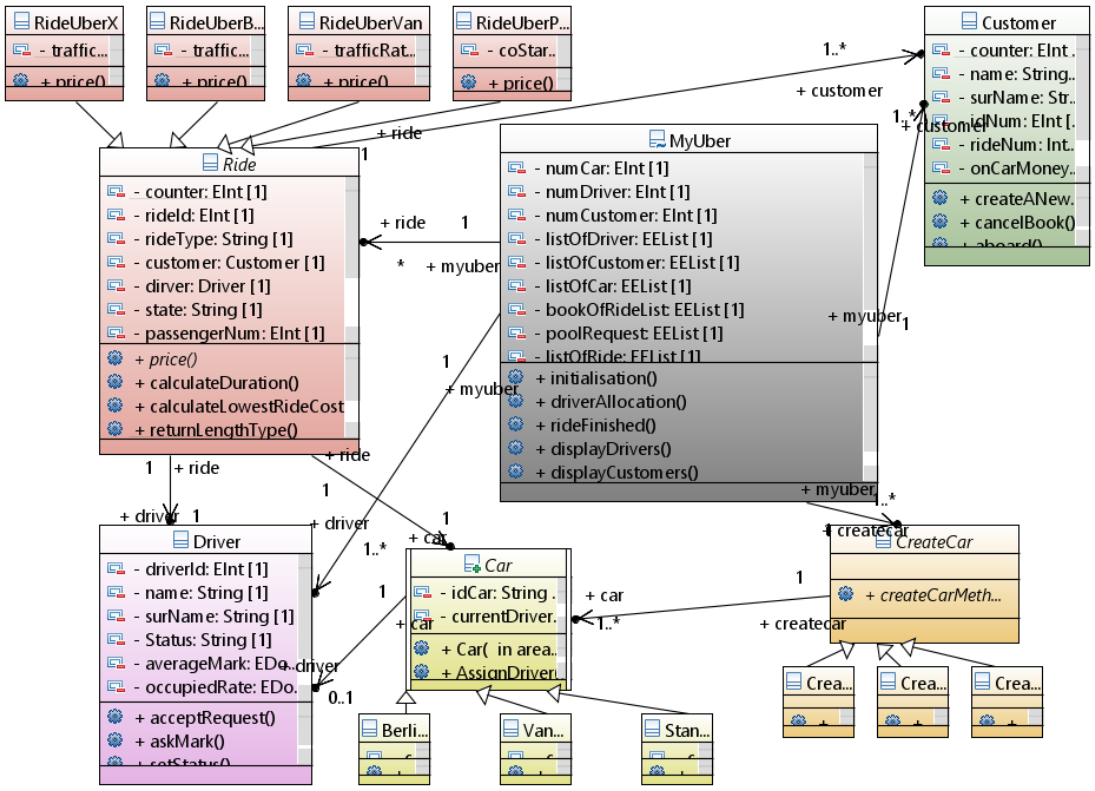

Fig 1 UML diagram

As shown in the UML diagram, the core class for our system is MyUber, Customer, Driver, Ride and Car. Besides, there are two class called MyTime and GPSLocation that we will their methods and attributes almost in every core class for the benefits of shorter time and more local test.

- Car: We decided to use factory since new types of car should be easily added in our system. The reason why we did not choose abstract factory pattern is that MyUber should only consider cars but now other types of transportation.
- Driver: Its field is built on drivers' personal information and the statistics that we want to display in our system. Each driver has two actions, accept request and ask for customer's appreciation mark. In reality, this action is initialized by customers, however in views that for a ride, one driver is associated with one customer, so the result of adding this action in there two classes is the same. Meanwhile, let driver process this method could avoid the customer's vicious attacks if they have access to modify driver's marks. This consideration reinforces the security of our system.
- Customer: The consideration is alike with Driver class except that we decided to replace the process of driver fetching customer to an aboard method which change nothing but the state of book. The reason behind this is to reduce the complexities of code implementation and fix the confusion of start time and start position in project description. By doing so, our framework takes driver's

accepting time and car original position as respectively the start time and start position of a ride.

- Ride: an interface which is implemented by four specific rides. The ride class build the connections between driver, customer and car class. It serves as calculate the price when customer send out a book request.
- MyUber: having all the initialization information, our managing system MyUber is a composition of other classes. The main processes are done here such as searching nearest car and calculate different statistics by invoking other instances' methods.

Implementation

The implementation of methods of our system should consider at least three functionalities: To represent a real process, that's to say, its subject and actions should be both true and logical in real life. To notify which is realizing information change or changing fields of instances of other classes. To give out a messenger that tells manager what's going on in MyUber system currently.

It's worth noting that the implementation by sharing code on GitHub. Once we agreed on the argument and output of a method, we split the tasks and work parallelly. This method of implementation has its strength and its shortcoming as well which will be detail in conclusion.

As for precise case, we have four main actions during a ride from the beginning to the end. Firstly, a Customer create a new ride, which is literally "createANewRide" method. It takes passenger number, start location, start time (if customer wants to reserve a ride for future). This method begins by creating two instances of MyTime and GPSLocation for future use as well as four types of ride. Then it uses the methods in specific ride to give customer a price. For second part, we will realize an interaction of customer and system, customer will choose a ride type, then this method will output an instance of this type of ride and proceed to next step.

The next main method is "driverallocation" in myUber class used to send this book of ride to nearest car and his current driver can choose to accept or not. It should have three functionalities, one is to sort the car by distance, the other two are to ask driver whether he or she accept or not and to refresh the system information: set driver for the ride or give out a messenger that no car available in case of being refused by all driver or simply no on-duty driver.

Next step is that the customer "aboard", as said before, we consider this action is done by instant for the sake of simplification in terms of code and clarifying the confusing start location and start time. Therefore, this method only needs to change the state of ride if the ride is not cancelled before aboard.

Finally, the ride finishes which is represented by “rideFinish” method in MyUber, this method does all the calculation to refresh both customer’s and driver’s data. In addition, we provide a possibility that each driver can switch its status after a ride, which is controlled by a stochastic mathematical function.

In addition, in each concrete ride type class, we override the abstract method “price” in “Ride” class, by using two hash maps who store its traffic and length type coefficients, to calculate the total price of one ride and return it to the customer.

Testing

Results

Conclusion

This section summarizes the overall conclusions drawn from this project and discusses what we learnt from it and what the resumed pipeline should be like. It also describes how the project could be extended with further work.

Thanks for this project, we are now more familiar with Java this language and the principles of OOP. It is our first time to design a java project from the very beginning, the UML, until the CLUI. We practice the factory pattern, the TDD, the JUnits test and we will try to create a GUI in the second part. However, in this process, we have also realized some disadvantages.

Firstly, although we know the importance of UML and TDD, which should be done at the beginning and could help a lot to our following works, we had failed to totally design them before we began coding, which is perhaps because we were not familiar enough with the knowledge we had learned in class.

Besides, we have found some problems about “Team Working”. A time-free teamwork is a great method to economize our working time and to give us more freedom of working. However, due to lack of a clear UML at the beginning and without enough communication, we did some repetitive work and make our class structure a little redundant.

We will do some optimization work in the second part of our project.

Appendices