**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 and an interface that handles the commands 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. Plus, 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 considering 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 and main methods
4. Use Junit test to implement methods
5. Simulation
6. CLUI and GUI design
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 as well as the example scenario.
5. Conclusions - Analysis of the successes and failures of the project, and discussion of the advantages and limitation of our design, at last the possible refinement.

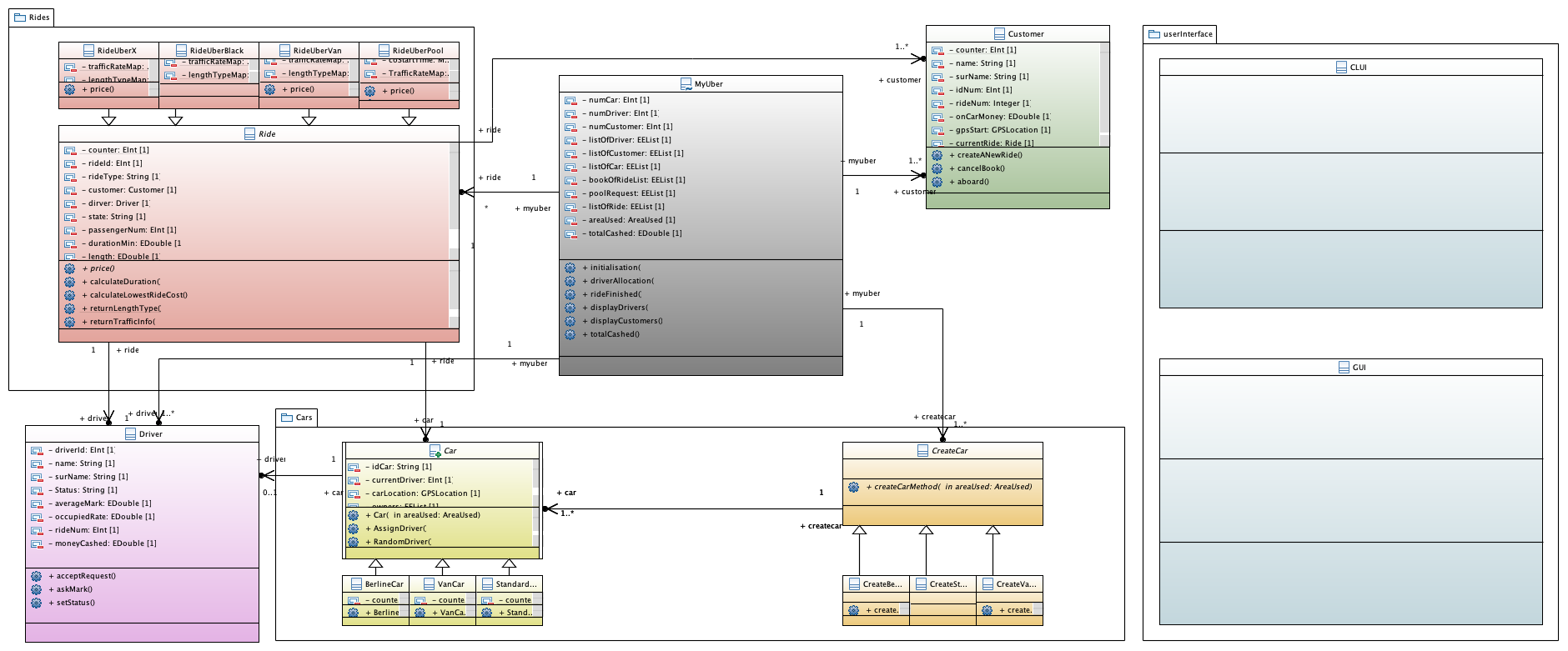
In addition to these main sections there is an appendices which shows how the workload has been split and its realization.

# 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.
* CLUI: We create a class to treat the commands. In this class, we composite MyUber class, because it’s reasonable that the command is sent to one myUber system. Then we defined a bunch of methods which mainly invoke the methods finished in first part and add the code for reading input stream and writing to a txt file. At last, we define a main method so that when the supervisor run the class, he or she can begin typing command to test.
* GUI: We discussed the appearance firstly, then we wrote the element needed in Swing in GUI class and put them orderly in each container. Nevertheless it left us not much time to develop the listener part which is crucial to interactivity.

# 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.

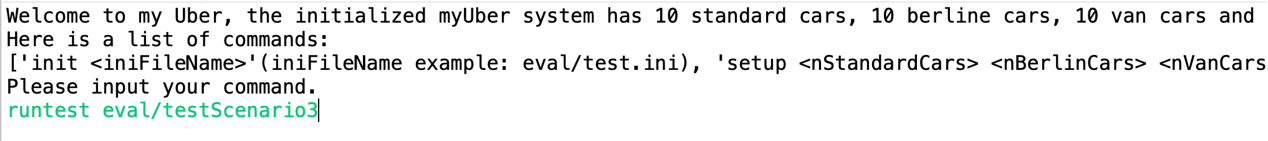
In addition, in each concrete ride type class, we override the abstract method “price” in “Ride” class, by using two hash maps who store its traffic and length type coefficients, to calculate the total price of one ride and return it to the customer.

# Testing

## Junit tests:

Junit tests have been done for the main classes. The Junit begins by setting up then it tests the methods respectively. Below is the description for the test scenario.

## Test scenarios:

We have created three main tests scenario, the TXT file names are: “testScenario1.txt”, “testScenario2.txt”, “testScenario3.txt”. In order to run these tests scenario, you should run ***CLUI*** class in ***userInterface***package and input “runtest eval/testScenarioX” where X is chosen from 1, 2 or 3.

**Fig 2 application of testScenario**

In *testScenario1.txt*, we have tested the main commands in the command list of project description, including *setup, addCustomer, addCarDriver, addDriver, moveCar, moveCustomer, setDriverStatus, simRide, etc*. We stored all output in *testScenario1Result.txt* automatically by using *runtest* method.

In *testScenario2.txt,* we mainly tested uberpool function. We have tried to input two uberpool requests both subsequently and alternately by another uber ride (in our test, it is an uberx). And finally, all requests have been processed normally.

In *testScenario3.txt*, we have tested some abnormal situations. Firstly, we had sent a uberx ride request with 5 passengers. MyUber system distinguished automatically this error, reminded user and returned a ubervan ride. Besides, we tried to reset myUber system with setup command, observed and compared two system status by displayState method to make sure that we have successfully initialized and reset our Uber system. What’s more, we have tested a scenario where the first two uberpool requests could not been combined, which means the total passengers number of these two requests is more than four, and we have verified that out system can handle this situation.

## Instructions of CLUI

In order to use CLUI of MyUber system, you have to run ***CLUI*** class in ***userInterface*** package. The system is initialized with *10 standard cars, 10 berline cars, 10 van cars and 10 customers*. There are 16 mains commands in this system. You can use *init, setup or runtest* commands to initialize this system. More details, please read doc documents.

# Results

So far, all the basic requirements are met in our system. That’s to say, major classes are complete and solid, there is no error when we run the test. The CLUI handle all the commands listed in the project description and can be extended easily for the new command. The frame of GUI class is defined but the interaction still needs to be developed

# 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. Then it talks about the advantage and limitations of our design. It also describes how the project could be extended with further work.

## What we learn from this project:

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, 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.

## The advantages and the limitations

The advantage of our project is mainly its extendibility, for example, we can add varied types of cars or simply augment its number without changing the codes in MyUber. We considered all the possible exception and handle it in a small and reasonable way. For example, we deliberately typed wrong to see if our system will handle these mistakes well or not. The answer is affirmative.

However, we cannot deny that there exist limitations in our development. The state of the driver and the ride can be defined as Enum type. It is more logical in this way.

We will do some optimization work in the second part of our project. After second thought, especially when we went through the lectures before the final exam, we realized that the observer pattern can be applied to this system. Especially for the Uberpool ride, when there is two customer book a Uberpool ride, and the driver accept, both of them should be noticed.

# Appendices

## Workload separation

X: Xu Gezheng

T: Tan Gansheng

|  |  |  |
| --- | --- | --- |
| Task Number and Name | Task Description | Finished by |
| 1. Set up project environment | Clarify GitHub working pattern and sharing process | T |
| 1. project interpretation | Define main classes, main attributes and principal function | X |
| 1. Design Pattern | Apply design pattern, decide class composition and heritance | T, X |
| 1. Cars classes | Develop Cars classed using factory pattern | T |
| 1. Ride classes | Define different rides classes using heritance and define the calculating method. | X |
| 1. MyUber class | Define the main methods required for the core code | T,X |
| 1. MyTime class | Define the own running time apart from the system synchronized time, and add time subtract and addition methods, and different constructor to suit every possible uses | T |
| 1. GPSLocation class | Define gps location and give out distance calculating method and generate random location | X |
| 1. Junits test on AreadUsed, Car, Customer, MyUber | Test all the main methods of the mentioned classed | X |
| 1. CLUI main command method construction | Enable user type commands and see the results | T,X |
| 1. UML | Predesign and refine the UML model at the beginning and after the part one | X |
| 1. Condes for Runset command | Reuse the code in CLUI to read a txt test file then show the result | X |
| 1. GUI framework | Create ordered framework and select suitable responder(listener) | T |