EECS3311 Final Report

Table of Contents :-

Introduction :-

I have made my project through JavaFX and SceneBuilder using Java version Java SE-1.8.

In my project, I have a total of 25 java classes that regulate my GUI in the Application package. Along with this, I also have 25 fxml files which I have used with SceneBuilder to build the appearance for my GUI. I also have certain png's in this package that I have used to make my GUI presentable.

For the nonGUI package, I contain 6 classes that each regulate and confirm things such as User login information, Customer Bookings, Paid Bookings, etc. I also have 6 txt files that I use as DataBase for my project. Each txt file contains vital information that are required for the performance of my project.

DataBase :-

I have 6 DataBase txt files for my project :-

1. AdminLoginInfo – Contains the login info for Admins. The file contains the following info :-
   firstname,lastname,id,email,password
   Robert,Downey,1,robert@yorku.ca,t1t1
   Chris,Evans,2,chris@yorku.ca,t2t2

2. AuthorityLoginInfo – Contains the login info for Toronto Parking Authority Officers. The file contains the following info :-
   firstname,lastname,id,email,password
   Peter,Parker,1,peter@yorku.ca,t1t1
   Thor,Odinson,2,thor@yorku.ca,t2t2

3. CustomerBookings – Contains the Bookings that Customers have made but not yet paid for. The file contains the following info :-
   firstname,lastname,ParkingSpace,LicencePlate,hours,mins,bookingid,email,status,approval,payment
   Venkatessh,Kumar,4,B78487,1,30,1,venki@yorku.ca,unpaid,granted,unpaid
   Patrick,Williams,5,A53849,0,30,2,patrick@yorku.ca,unpaid,ungranted,unpaid

4. CustomerLoginInfo – Contains the login info for Customers. The file contains the following info :-
   firstname,lastname,id,email,password
   Venkatessh,Kumar,1,venki@yorku.ca,t1t1
   Patrick,Williams,2,patrick@yorku.ca,t2t2

5. PaidBookings – Contains the Bookings of customers that have already been paid for, with the expiration time. The file contains the following info :-
   firstname,lastname,ParkingSpace,LicencePlate,hours,mins,bookingid,email,expiry
   Venkatessh,Kumar,7,B78487,1,0,1,venki@yorku.ca,15:00
   Patrick,Williams,6,A53849,0,30,2,patrick@yorku.ca,14:30

6. ParkingSpaceNumbers – Contains the list of all the Parking Space Numbers with the occupancy of the space next to it. The file contains the following info :-
   parkingspace,occupancy
   1,unoccupied
   2,unoccupied
   3,unoccupied
   4,occupied
   5,occupied
   6,occupied
   7,occupied
   8,unoccupied
   9,unoccupied
   10,unoccupied

Class Diagram :-

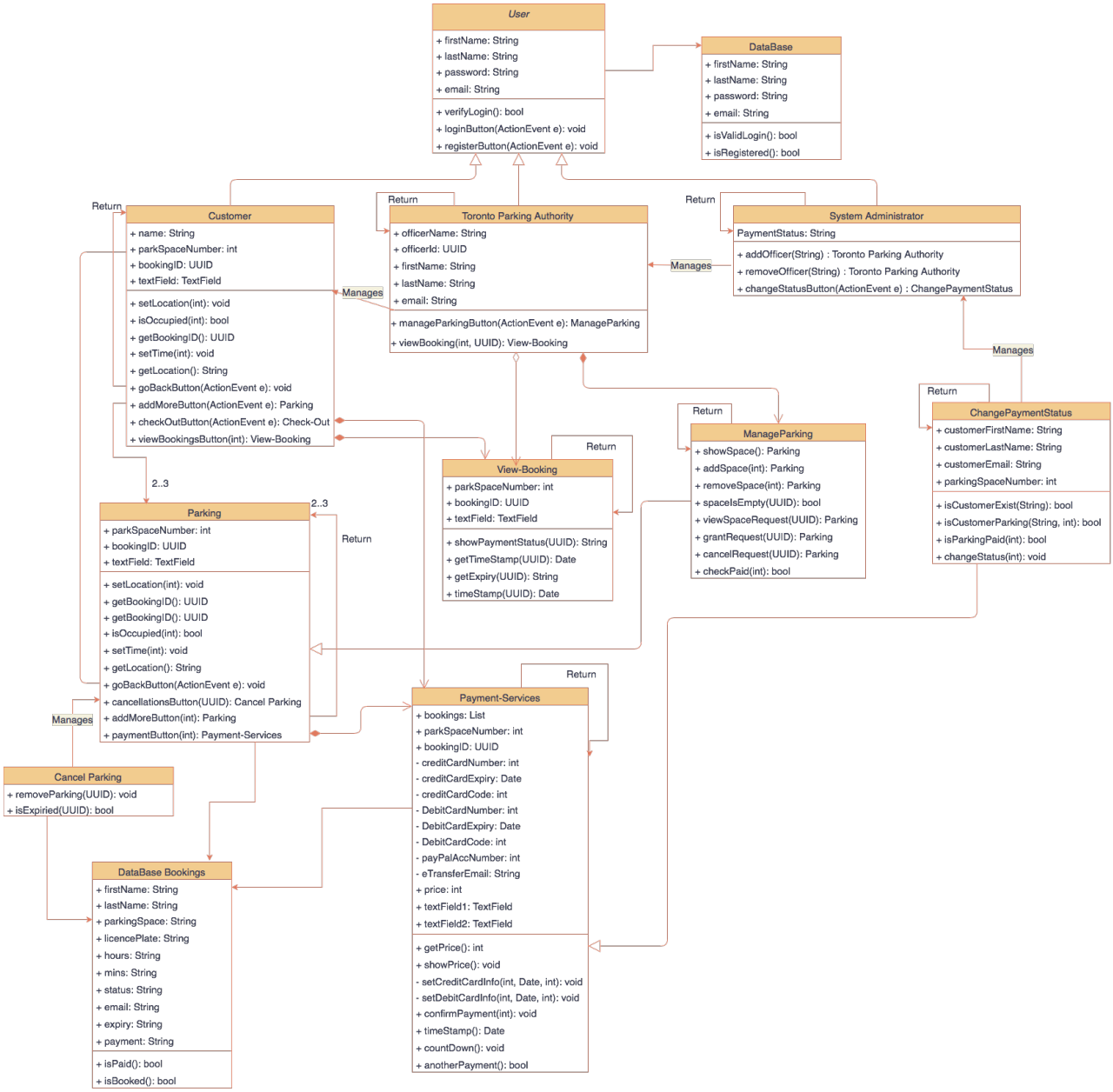
I have followed the same design that I have designed for my midterm apart from the addition of DataBase. The design patterns I have used for the project are the same as what I had specified for the midterm. Namely :-

1. Abstract Factory Pattern – The Factory pattern returns an instance of one of several possible classes depending on the data provided to it. The Factory is a class that decides which of the subclasses to return depending on the arguments you give it. The base class is abstract and the pattern must return a complete working class. Parameters are passed to the factory telling it which of several class types to return. The Abstract Factory pattern is one level of abstraction higher than the factory pattern. This pattern returns one of several related classes, each of which can return several different objects on request. In other words, the Abstract Factory is a factory object that returns one of several factories. I chose this patter because - One classic application of the abstract factory is the case where your system needs to support multiple “look-and-feel” user interfaces, such as Windows, Linux or Macintosh. You tell the factory that you want your program to look like Windows and it returns a GUI factory which returns Windows – like objects. When

you request specific objects such as buttons, check boxes and windows, the GUI factory returns Windows instances of these visual interface components. Since the system we are meant to create has to implement a GUI, using an Abstract Factory Pattern struck me as a sensible option due to the very reason that the GUI implementation is simplified using this pattern. How I implemented this design pattern into my system :-
For the Abstract Factory design pattern, by using a number of subclasses that instantiate objects in my system, they specify additional functionalities. Classes such as Customer, Toronto Parking Authority and System Administrator are subclasses that instantiate variables hence, this design pattern is useful. This approach is important because the subclasses can create instances and allows for loose coupling, which allows the object of Customer, Toronto Parking Authority and System Administrator to be independent.

2. Bridge Pattern – The Bridge pattern is intended to keep the interface to the client program constant while allowing us to change the actual kind of class we display or use. This can prevent us from recompiling a complicated set of user interface modules, and only require that we recompile the bridge itself and the actual end display class. In the system that we are implementing, since after login we need to perform multiple types of functions and it would be too tedious to restart the interface everytime we need to perform a function, the Bridge pattern helps the system to keep to the client program while traversing through the multiple methods that need to be invoked. How I implemented this design pattern into my system :-
In my system, by implementing the GUI system, I have chosen which screen from which class to display to the user through specified coding. The system I have designed also includes the feature allowing the user to move back and forth between the pages. This feature allows the user to go to any desired page without restarting the whole system. The bridge pattern has prevented my system to get overcomplicated and difficult to use for the user.

The following is my final UML Class Diagram that I have based my project off of. I have added the DataBase for bookings as visible in the bottom left hand corner of the image.



Description of GUI :-

The System I have designed for this project includes implementation of multiple classes and a hierarchy system starting off from an abstract class name “User”. This abstract class contains login details such as first name, last name, password and email id so that users can enter these details to either login to their account or they can register as a new user and then login. The details they enter here are saved into the DataBase class to grant the user’s access if they have an account created.

Moving on, when logged in, you can enter 3 interfaces depending on the login details that were entered. You could either have logged in as a Customer, a Toronto Parking Authority Officer or a System administrator. Depending on who you are logged in as, you have a set of tasks that you can do move deeper into the system. Taking the Customer interface for example, your name is collected from the login info that you have given. You are also allowed to book a parking spot by entering the parking space number. Doing this reserves a parking spot for you and returns an ID for the spot. You can book upto 3 parking spots after which you will be given the option to pay for the spots that you have chosen.

When trying to book parking spots, you also will be given a display for a button that says cancel booking which takes you to a new page where you can cancel a previous booking that was made by the customer. You also have the opportunity to return to the previous page using a back in case any of the details that you had specified was wrong. There is also a button that helps the customer to view their bookings by typing their parking space number.
The final action a customer is allowed to do is Payment-Services. This is the interface that lets the customer pay for every parking space they have booked for. Customers are free to pay using either Credit card or Debit card.

In the Payment-Services interface, the customer can check the total cost of their bookings and also receive a time stamp of their payment. In order to have the payment details of the customer in a secure environment, the card details/ payment details of the customer is put into a private variable which can only be accessed when in the Payment-Services interface and only by the customer who inputted the details. The Payment-Services interface also confirms the payment one last time with the user before the transaction is occurred. A countdown timer is set for when the transaction is in process.
Moving on to the next User Interface, the Toronto Parking Authority users are officers responsible to handle the info regarding the parking spots. When logged in, the interface will receive information such as officer name, officer ID, etc. from the officer. From this interface, the officers are allowed to login as a user to view the users booking regarding a parking spot; Or they can click the ManageParking button to handle the parking spaces. They can add a parking space, remove a parking space, and review the space request from a customer and decide whether to grant or cancel the request from the customer. The officers are also able to check if a parking space is empty and if a parking space is paid for. The Toronto Parking Authority interface also manages the Customer interface as the Officers are responsible for handling the request put forth by

the Customers.

Next, we have the System Administrator Interface which includes just one variable which a String names PaymentStatus. This variable holds the String “Paid” is the parking space in question is paid and “Unpaid” otherwise. The System Administrator also handles and manages the Toronto Parking Authority interface the administrators can add and remove officers in the Toronto Parking Authority class. The Administrators are also prompted with a button labelled “Change Payment Status” that lets the administrators check if for the existing customer, have they paid for their booked parking spots. After the checks have been done, the PaymentStatus String is changed accordingly.

Explaining the relations ship between each class :-
User is an abstract class that has a direct assosciation with DataBase for storing the login info of users.

The 3 interfaces, Customer, Toronto Parking Authority and System Administrator, each are class that inherit the User class. These interfaces are unlocked after the said user logs into the interface.

Parking is a class that is associated with the customer interface to handle the parking space bookings that will be prompted by the customer.

Cancel parking is another class that manages parking whose main functionality is to delete a previous booking due to the said request by the user.

View-Booking class is a composition class of the customer class since you cant view a booking without the customer placing the booking in the first place. It is also an aggregation of the Toronto Parking authority class since the officers can login as customer to check the customer’ bookings.

The payment services class is a composition to both the Customers’ class and the Parking class. Since the payment cannot take place without a booking being place, nor can it happen without a customer.

ManageParking inherits Parking class to perform the methods described in it such as checking for parking space, adding parking space, removing space, etc. which will be dealt with by the Toronto Parking Authority. ManageParking is also a composition to the Toronto Parking Authority class since the ManageParking cannot be accessed by anyone apart from the officers.

The ChangePaymentStatus class manages System Administrator to perform the said task of changing the status for a parking space based on if the payment has be done or not. This class also inherits the Payment-Service class to make sure that the parking space is paid for. All this takes place only if the customer that is passed through exists and if the customer is the user that booked the parking space in question.

Instructions on how to set up project :-

To run this project, Java SE-1.8 must be the least Java version required. To get the project working after importing, the E(fx)CLIPSE plugin must be installed in the user's eclipse (install from eclipse marketplace). This is all that is needed. But in the odd scenario where the project does not run even after installing the JavaFX plugin, please follow this video to get a more detailed instruction on running JavaFX projects :-
https://www.youtube.com/watch?v=bC4XB6JAaoU&t=323s

After install JavaFX plugin, right click on main.java in application package and click run as -> java application. The project should now run and the GUI should popup.