

EECS 3311 Final Project Winter-2021

Farhan Reza Latif

farhan95

215145592

farhan95@my.yorku.ca

Table of contents

Project Summary	3
Project Strategy.....	3
Design pattern.....	4
Assumptions.....	4
Program Structure.....	5
Difference between final implementation and Mid-term implementation.....	7
Final Product	7
Testing.....	8
Class-Diagram	9

Project Summary

Project Name : instaParking (Car Parking management system)

The purpose of this project is to minimize the number of parking meters that need to be visited to collect money. Help parking enforcement workers monitor payment . The product will allow customers to pay for single parking spaces online, while the Toronto Parking Authority personnel's do not have to collect money from each parking meter. Our project has three users, Admin, Customer, and Parking enforcement officer (PEO). In this project we have up-to 9 system features or system requirements that builds up and completes the project.

Project Strategy

In this project we have up-to 9 system features, for the ease of building the project I shuffled and mapped the requirements to their corresponding user. First, I breakdown the users of this product;

User = customers + parking enforcement officer

Customer = the one who pays for the parking space

Parking enforcement officer (PEO) = monitors for parking spaces

Admin = Adds and Removes PEO and confirms Payment status of the customer.

And then the system requirements 4.1 to 4.9 as per the users;

REQ 4.1, 4.9: GUI and backend for system admin to add/remove parking enforcement officer
GUI for system admin to update customers' payment status. => **Admin**

REQ 4.2: GUI and backend for customer to register and create account => **Customer**

REQ 4.3: GUI and backend for user to login and access (role based login) => **Admin + User**

REQ 4.4 and 4.5 and 4.6: GUI and backend for booking the parking space and duration of booking [4.4] and cancel the booking [4.5] and pay for parking 4.6 => **Customer**

REQ 4.7: GUI and backend to view bookings (Information such as expiry time and payment status can be viewed.) => **Customer**

REQ 4.8: GUI and backend for parking enforcement officer to manage parking spaces by adding or removing them => **PEO**

Design pattern

I'm employing design patterns which shapes up the product and helps us reach the final product, in the most efficient way. The design pattern used are:

- **Singleton Design Pattern:**

Singleton because, we are creating objects from classes which can be accessed directly without need to instantiate the object of the class. And the classes are required to control the execution of the program.

- **Builder design pattern :**

Here, we create a complex object that is built using simple objects in a step by step approach. Additionally, we are separating the implementation and representation of the object and hiding implementation details of the object from the user.

The primary reason to user builder pattern is that it, separates the implementation of a complex object from its representation so that the same construction process can create different representations. In here, we have different representations and different functionalities for the representations. Builder pattern provides the ease of separating them and provides greater efficiency and control over the implementation process.

- **Data Access Object (DAO) Pattern :**

This pattern is used to separate low level data accessing API or operations from high level services. The DAO interface, CSVOps, defines the standard operations to be performed on model. The DAO concrete class, Admin, Booking, and cancelBooking implements the interface, getting the data from the data source, and Model, payment, ViewBooking, stores the data retrieved by the concrete class.

Assumptions:

- The system does not replace current parking meters.
- Every Parking spot has up-to 50 Spots. (Consulted with Prof Song Wang.)
- The system administrator already exists in the system and has a master login.
- Customers are allowed to book more than one parking space (with a limit up to 3).
- Parking enforcement officers must login as a customer in order to book a parking space for themselves.
- Customers can book in custom time slots (Min -> 1hr, Max -> 24 hrs) .
- Customer can't book before current time and date.

Program Structure

Tools:

- **Language:** Java 8
- **Framework:** Java swing
- **Build tool:** Maven
- **Testing Framework :** JUnit

Step-1: Structuring

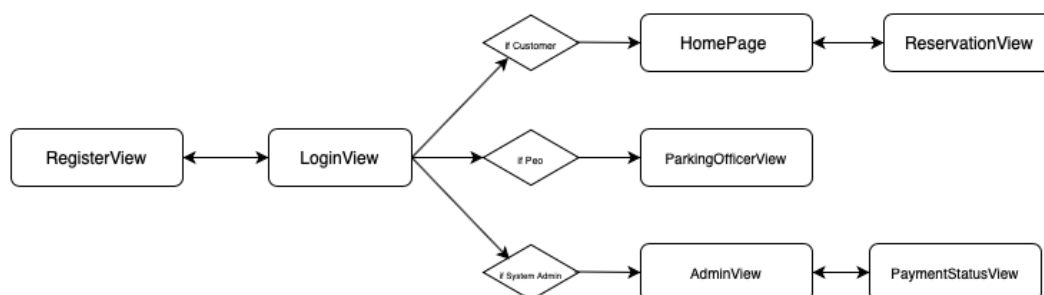
After careful studying and analyzing of the software requirements specification and system features, I have divided and grouped the functionalities as per the user actions and privileges. For example the customer can do book, cancel, pay and view booking. I have grouped by backend [model] classes, ie group by their functionalities.

- **AdminOperations** package contains all classes related to admin
- **PEOOperationsModel** package contain all classes related to parking officer
- **CustomerOperationsModel** package contain classes related to customer
- **LoginRegistrationModel** package contains classes related to login and registration
- **GUI** package contains all the GUI [front-end] classes.

Step-2: GUI development

- All GUI classes are done using the eclipse plugin window builder and significant part is hard coding.
- All classes are tested via debugging and simulations of the application.
- Below is the flow diagram of the GUI classes which displays their path/direction at runtime

GUI flow Diagram



Step-3: Login and Registration

LoginView and RegistrationView are the GUI classes where user puts in their data. The data is processed and by using Login and register. In the register classes we input the user data and the data is put into the database.csv file. In the LoginView we put in the username and password, using the login class, we authenticate. The authenticate() method in login goes through the database.csv and verifies if the user present in the CSV or not. If the CSV has the user then they get logged in otherwise not. Another feature in the login file is checkUserType() which finds out the user type of the user. If the user type is customer pressing the login button will take to the customer homepage, if the user type is parking enforcement officer (PEO) then pressing the login button will take to the PEO page.

Step-4 Booking and Cancelling (CustomerOperationsModel):

There are 4 classes dedicated this segment, Booking, CancelBooking, ViewBooking, Payment . The booking class contains checkParkingSpace() which checks whether the space is empty or not, the bookParkingSpace() method puts in all the data like username, parking spot name, license number etc, to the booking.csv database. The userCount() counts the number of time the user is present in the database, this is used to make sure the user can book spot up to the given limit. The bookingID() method generates a random booking ID for the parking transaction.

The CancelBooking class has two methods cancelBooking() and checkTiming(). The cancelBooking() method cancels the booking by checking the booking database for the entered spot name. The checkTiming() method checks whether the booking time is before the current time or not. The ViewBooking class has one method viewBooking() which displays the booking.csv data to the GUI.

The payment class has price() method which calculates the price based on the duration of the booking. The timeStamp() method gets the current time in timestamp format and is used to get the payment time. The counter method get the expire time by calculating the duration plus current time.

Step-5: Parking officer operations (PEOOperationsModel)

The parking enforcement officer (PEO) is allowed to add and remove parking spaces. The parkingOfficerView is the GUI where PEO does this operations. In the PEO class we write the

addSpaces() and removeSpaces() method which adds and removes spaces respectively. The countingParkingSpaces() counts the empty parking spaces in the location and displays the data in the GUI.

Step-6 Admin operations (AdminOperationsModel)

The admin is allowed to add and remove Parking enforcement officer and confirm payment status. The Admin class deals with adding and removing the parking officer with writeUser() and removeUser() respectively. The PaymentStatus class deals with confirming payment with confirmPayment() method. The class also has viewBooking() which views the entire booking database to the GUI from which the admin can confirm by checking the timestamp.

Design trade-off: (consulted with Prof. Song Wang)

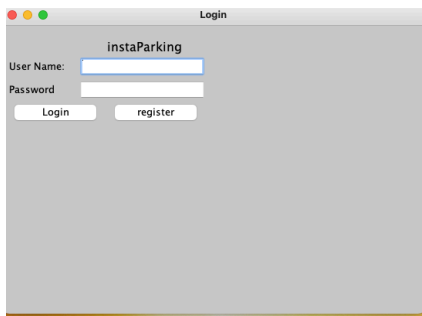
- The payment and booking are done in the same GUI to make user experience simple, usability easy, and better optimization of code.

Difference between final implementation and Mid-term implementation

In the Mid-term, I used only one design pattern, Builder pattern. And the implementation was simplistic not taking into account the relations between GUI classes and the model classes in the back end. The Mid-term implementation was rather is simplistic and high-level implementation with not attention given to the low-level detail of read, write and delete data from CSV. Much attention was not also given to the runtime issues and complexity of the code which can be understood and encountered once the application is built. For the final implementation, for the purpose of design efficiency, I added two more design patterns Singleton and DAO, and the GUI class's to ensure greater user experience. Additionally, for the final implementation user integration and user experience factors were also taken into account and some security features were also implemented which were not done in the Midterm. The final project builds on the mid-term implementation and builds on that.

Final Product

The final product is a desktop application built on Java, Java Swing, Maven as the build tool. It can run on MacOS, windows, Linux. It successfully fulfils all the requirements asked in efficient and secured way, and ensures good user experience.



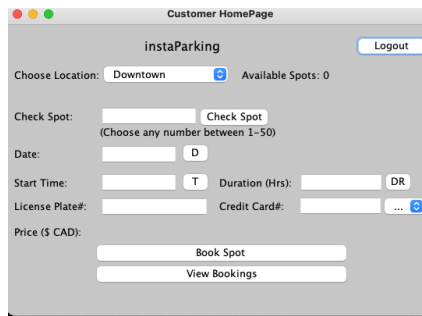
Login

instaParking

User Name:

Password:

Login register



Customer Home Page

instaParking Logout

Choose Location: Downtown Available Spots: 0

Check Spot: Check Spot
(Choose any number between 1-50)

Date: D

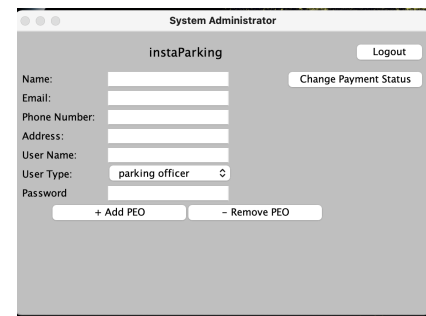
Start Time: T Duration (Hrs): DR

License Plate#: Credit Card#: ...

Price (\$ CAD):

Book Spot

View Bookings



System Administrator

instaParking Logout

Name:

Email:

Phone Number:

Address:

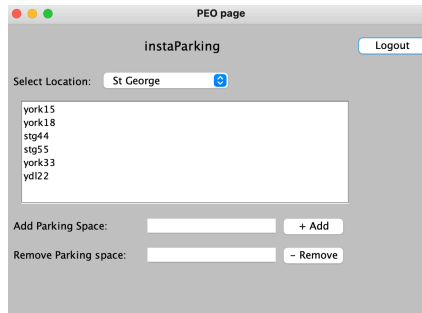
User Name:

User Type: parking officer

Password:

+ Add PEO - Remove PEO

Change Payment Status



PEO page

instaParking Logout

Select Location: St George

york15
york18
stg44
stg55
york33
ydl22

Add Parking Space: + Add

Remove Parking space: - Remove

Testing

This Application was built in a test-driven environment(TDD), where the test cases were written simultaneously with the code, and testing was conducted at a very high amount to ensure quality. The testing was done in JUnit5 and additionally, the code was debugged extensively using the PSVM method in Java. Finally, to ensure quality final product extensive simulation was done of the application.

Code Coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
parking	0.1 %	7	7,550	7,557
src/main/java	0.1 %	7	6,766	6,773
GUI	0.0 %	0	4,542	4,542
HomePage.java	0.0 %	0	1,209	1,209
AdminView.java	0.0 %	0	714	714
ParkingOfficerView.java	0.0 %	0	663	663
RegistrationView.java	0.0 %	0	490	490
PaymentStateView.java	0.0 %	0	421	421
ReservationView.java	0.0 %	0	397	397
LoginView.java	0.0 %	0	334	334
PayView.java	0.0 %	0	314	314
CustomerOperationsModel	0.0 %	0	706	706
Booking.java	0.0 %	0	315	315
CancelBooking.java	0.0 %	0	204	204
ViewBooking.java	0.0 %	0	110	110
Payment.java	0.0 %	0	77	77
AdminOperationsModel	1.0 %	7	679	686
Admin.java	1.9 %	7	365	372
PaymentStatus.java	0.0 %	0	314	314
RegistrationLoginModel	0.0 %	0	488	488
Login.java	0.0 %	0	222	222
User.java	0.0 %	0	166	166
Register.java	0.0 %	0	100	100
PEOOperationsModel	0.0 %	0	351	351
PEO.java	0.0 %	0	351	351
src/test/java	0.0 %	0	784	784
CustomerOperationsModel	0.0 %	0	320	320
BookingTest.java	0.0 %	0	118	118
PaymentTest.java	0.0 %	0	78	78
ViewBookingTest.java	0.0 %	0	67	67
CancelBookingTest.java	0.0 %	0	57	57
RegistrationLoginModel	0.0 %	0	235	235
UserTest.java	0.0 %	0	109	109
LoginTest.java	0.0 %	0	63	63
RegisterTest.java	0.0 %	0	63	63
AdminOperationsModel	0.0 %	0	154	154
PaymentStatusTest.java	0.0 %	0	84	84
AdminTest.java	0.0 %	0	70	70
PEOOperationsModel	0.0 %	0	75	75

Final
implementation
class diagram
(class-
diagram.png)

