Parking Garage Software Design Document
Group 1

Revision History

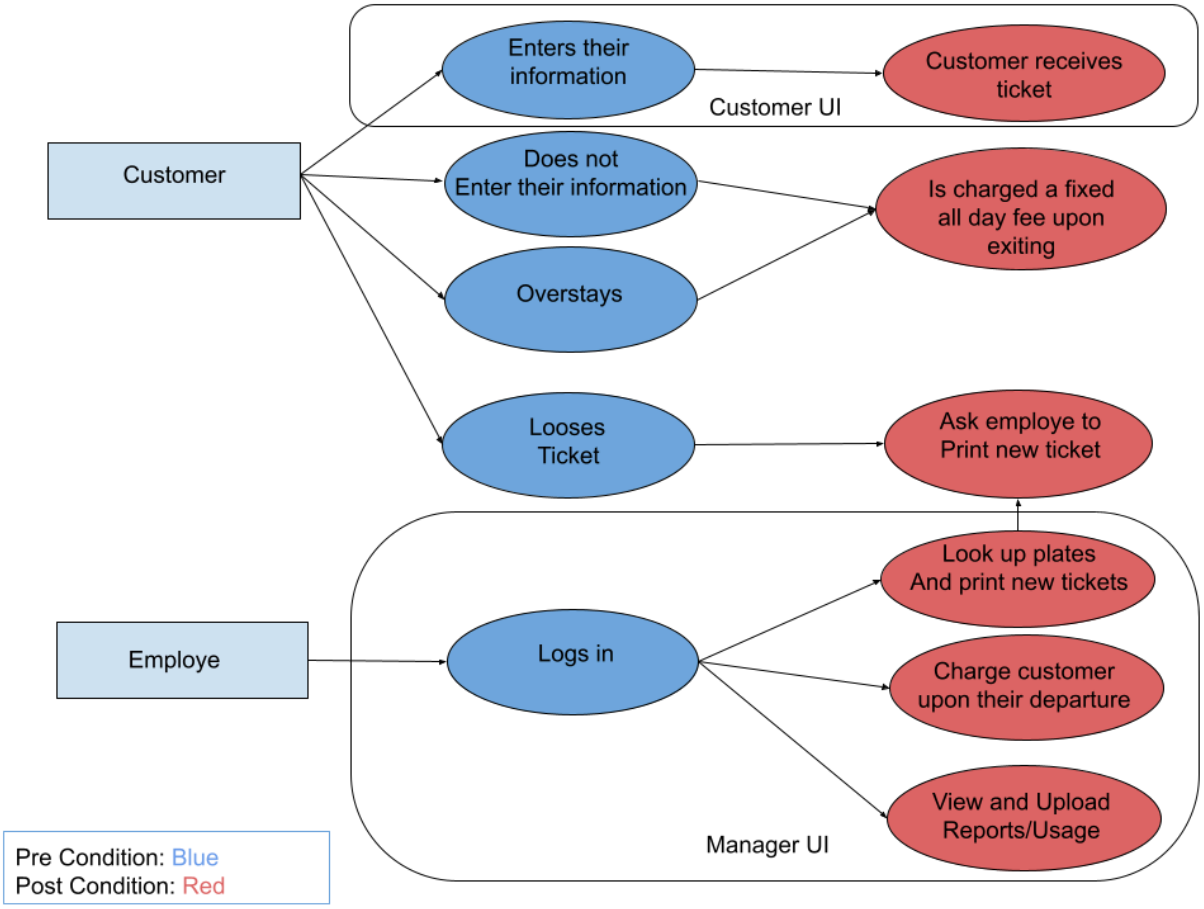
Date	Revision	Description	Author
10/22/24	0.1	Template	Jeremy Renati
10/25/24	1.0	Table Of Contents and organizing doc	Jeremy Renati
10/29/24	1.1	Added Class Diagram	Adrian McGee
10/29/24	1.2	Added use case for the GUI class	Adrian McGee
10/30/24	1.3	Updated use case Diagram	Omar Rosas
10/30/24	1.4	Updated class diagram with new added classes +attributes/methods	Marc Bassal
10/30/24	1.5	added a new updated refresh class diagram	Adrian McGee
10/30/24	1.6	added definitions to the class design section	Omar Rosas/Adrian McGee
11/12/24	2.0	Refreshed our Classes	Adrian McGee
11/20/24	2.1	Group Meeting with Professor - changed designs to reflect feedback received	Adrian McGee
11/30/24	3.0	Added finishing touches to accurately reflect our changes to our class designs	Adrian McGee
12/2/24	4.0	Changed to reflect completed code	Timofay Udalkin

-

Table of Contents

Use Cases and Sequence

1.1. Use Case Diagram



1.2. Use Cases

Use Case ID: UC001
Use Case Name: Parking Server Program is launched by the manager
Primary Actor: Manager or Authorized Individual
Pre-conditions:

- The program is installed on a machine with an IP that can be connected to.

Post-conditions:

- The server is running successfully.

Basic Flow or Main Scenario:

1. An authorized individual starts the server.

2. The program is launched as a Java project.
3. No arguments are passed.

Use Case ID: UC002
Use Case Name: Parking Client Program(s) launched by the manager (R0)
Primary Actor: Manager or Authorized Individual
Pre-conditions:

- The server is running.

Post-conditions:

- The client is running, connected to the server, and the manager login page is open.

Basic Flow or Main Scenario:

1. The hostname is entered as an argument.
2. The manager launches the program as a Java project.

Use Case ID: UC003
Use Case Name: Manager Prints Report (R0, R1)
Primary Actor: Manager
Pre-conditions:

- The manager is logged in.

Post-conditions:

- A report is printed onto a text file.

Basic Flow or Main Scenario:

1. The manager presses "Print Report."
2. The client sends a request for the report to the server.
3. The server generates and sends an up-to-date report.
4. The client prints the report to a text file and displays the file's location.
5. A window opens, displaying that the report has been printed.
6. The manager presses "OK" to return to the home screen.

Use Case ID: UC004
Use Case Name: Manager Starts Customer GUI

Primary Actor: Manager
Pre-conditions:

- The manager is logged in.

Post-conditions:

- The Customer GUI is started successfully.

Basic Flow or Main Scenario:

1. The manager selects the option to start the Customer GUI from the Manager Selection Screen.
2. The Customer GUI is launched, allowing customers to interact with the system.

Use Case ID: UC005
Use Case Name: Customer Prints Ticket
Primary Actor: Customer
Pre-conditions:

- The Customer GUI is running.
- The server is running.

Post-conditions:

- A ticket with a unique ID, timestamp, and unpaid status is generated.
- The ticket is sent to the client and displayed or printed for the customer.

Basic Flow or Main Scenario:

1. The customer initiates the "Print Ticket" action from the Customer GUI.
2. The Customer GUI sends a request to the server to check garage availability.
3. If the garage is not full, the server generates a ticket with:
    - Unique ID
    - Timestamp
    - Unpaid status
4. The ticket is sent to the client and printed or displayed to the customer.

5. If the garage is full, the Customer GUI informs the customer of the unavailability.

Use Case ID: UC006
Use Case Name: Customer Pays for Ticket
Primary Actor: Customer
Pre-conditions:

- The Customer GUI is running.
- The server is running.
- The customer has an unpaid ticket.

Post-conditions:

- The ticket status is updated to "Paid" on the server.
- The payment amount is logged for reporting.

Basic Flow or Main Scenario:

1. The customer selects the "Pay Ticket" option in the Customer GUI.
2. The customer inserts ticket and payment details (e.g., card information).
3. The Customer GUI sends the payment details and ticket ID to the server.
4. The server validates the ticket ID and processes the payment.
5. If the payment is successful:
    - The ticket status is updated to "Paid."
    - The amount is logged in the server for reporting.
    - A confirmation is displayed to the customer.
6. If the payment fails, an error message is displayed, and the customer is prompted to retry.

Use Case ID: UC007
Use Case Name: Manager Closes Customer GUI and Returns to Manager Selection Screen
Primary Actor: Manager
Pre-conditions:

- The Customer GUI is currently running.

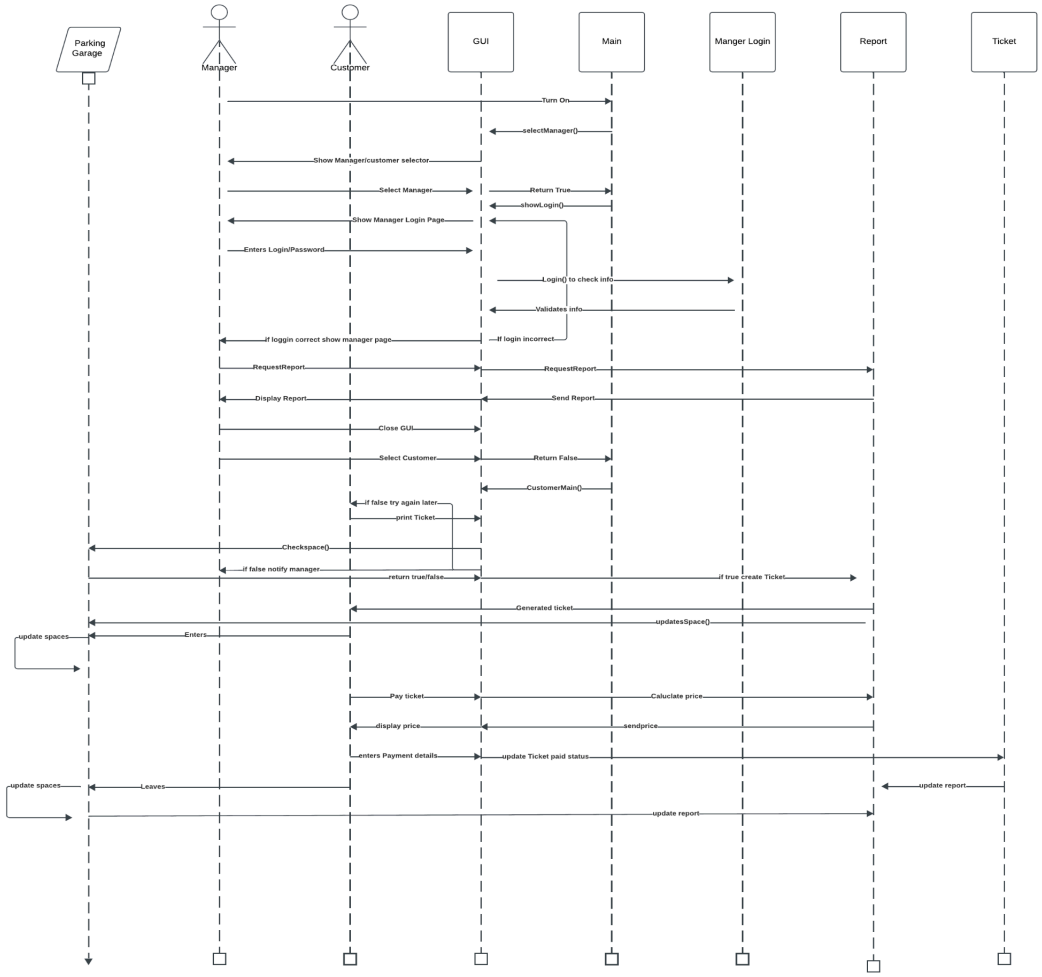
Post-conditions:

- The Customer GUI is closed.
- The Manager Selection Screen is displayed.

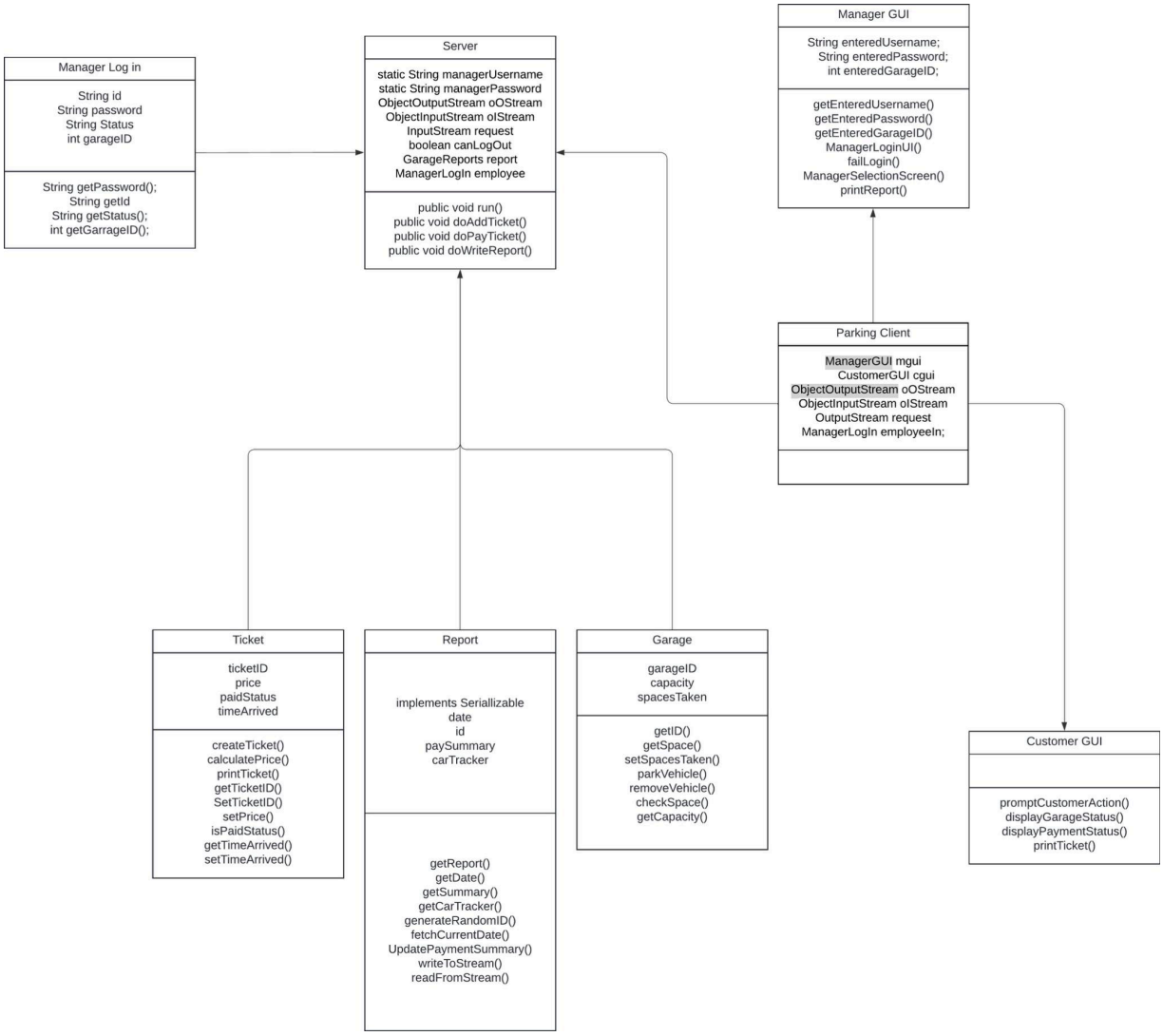
Basic Flow or Main Scenario:

1. The manager initiates the "Close Customer GUI" action from the Customer GUI interface.
2. The Customer GUI performs the following:
    - Saves any unsaved state or data.
    - Confirms that there are no ongoing actions (e.g., active transactions).
3. The Customer GUI is terminated.
4. The system automatically redirects the manager back to the Manager Selection Screen.

1.3. Sequence Diagram



1.4. Class Diagram



Class Design

2.1. Customer GUI

2.1.1. The GUI will provide an interface that is simple and easy to use for the customer to complete the desired actions.

2.1.2. Methods:

2.1.2.0. promptCustomerAction( ): Displays a message to the user to choose an action: “Print Ticket” or “Pay Ticket”

2.1.2.1. displayGarageStatus( ): Displays a message to the user on the status of the garage’s capacity: Garage is full or Welcome to the Garage

2.1.2.2. displayPaymentStatus( ): Displays a message to the customer on what their payment status is

2.1.2.3. printTicket( ): Displays a message to print ticket or if there was an error

2.2. Manager GUI

2.2.1. The GUI will provide an interface that is simple and easy to use for the manager to complete the desired actions.

2.2.1.0. getEnteredUsername(): Returns the entered username.

2.2.1.1. FetEnteredPassword(): Returns the password

2.2.1.2. getEnteredGarageID(): Returns the garageID

2.2.1.3. ManagerLoginUI(): Creates a window within the GUI for the manager to enter their credentials

2.2.1.4. failLogin(): Displays a message that the entered input was invalid

2.2.1.5. ManagerSelectionScreen(): Displays a message to either “Print Report” and “Turn on customer GUI”

2.2.1.6. printReport(): Prints a message that the report is being printed

2.3. ParkingClient

2.3.1. The Client class is crucial in a parking garage system because it facilitates seamless communication between the kiosk GUI (customer-facing) and the manager GUI, centralizing operations and ensuring data consistency.

2.3.2. Methods

2.3.2.0. ManagerGUI: Creates the object from the ManagerGUI and displays a connection

2.3.2.1. CustomerGUI: Creates the object from the CustomerGUI and displays a connection

2.4. Parking Server

2.4.1. The Client class is crucial in a parking garage system because it facilitates seamless communication between the kiosk GUI (customer-facing) and the manager GUI, centralizing operations and ensuring data consistency.

2.4.2. **Methods:**

- **2.4.2.0.** **run():** Allows the server to run and make a connection to the client
- **2.4.2.1.** **doAddTicket():** Adds the ticket method to the server
- **2.4.2.2.** **doPayTicket():** Adds the pay ticket method to the server
- **2.4.2.3.** **doWriteReport():** Adds the do write report method to the server

2.5. Manager Login

2.5.1. The manager-login class allows an employee to verify themselves to be able to use the employee UI.

2.5.2. **Methods:**

- **2.5.2.0.** **getID():** Returns the unique ID
- **2.5.2.1.** **getPassword():** Returns the password
- **2.5.2.2.** **getStatus():** Returns the current status
- **2.5.2.3.** **getGarageID():** Returns the garages ID
- **2.5.2.4.** **setStatus():** Sets the status of the Garage
- **2.5.2.5.** **validateLogin():** Displays a message if the login was successful or not
- **2.5.2.6.** **selectGarage():** Displays a message of the current selected garage and displays an error message if the user has not logged in.

2.6. Garage Reports

2.6.1. The Report class is essential in a parking garage ticketing system because it consolidates key data on daily operations and financial summaries, helping managers track and analyze parking usage, vehicle flow, and revenue generated within specific periods. This class allows managers to create, view, modify, and maintain records of garage performance.

2.6.2. **Methods**

- **2.6.2.0.** **getDate():** Returns the current date
- **2.6.2.1.** **getID():** Returns the unique ID of the report
- **2.6.2.2.** **getSummary():** Returns a summary of the crucial information from the garage
- **2.6.2.3.** **getCarTracker():** Returns how many cars were tracked how during the day
- **2.6.2.4.** **generate RandomID():** Generates a random ID that is assigned to the report
- **2.6.2.5.** **fetchCurrentDate():** Creates the a variable and stores the date

2.6.2.6. **updatePaymentSummary():** Updates the Garages total earnings and the car tracker

2.6.2.7. **writeToStream():** Write the summary to a text file and saves

2.6.2.8. **readFromStream():** Reads from a text file

2.7. Ticket

2.7.1. The Ticket class represents individual parking tickets in the system, managing ticket details and payment status. With attributes like ticketID for unique ticket identification, price is to indicate the final cost and paidStatus to indicate if the ticket has been settled, this class centralizes essential ticket information. This class supports smooth ticketing operations and accurate payment processing within the garage system.

2.7.2. Methods

2.7.2.0. **getTicketID():** Get the unique ticket ID

2.7.2.1. **setTicketID():** Set the ticket ID

2.7.2.2. **getPrice():** Retrieve the price of the ticket

2.7.2.3. **isPaidStatus():** Returns a boolean, true or false, if the payment has been paid or not

2.7.2.4. **setPaidStatus():** Set the paid status

2.7.2.5. **getTimeArrived():** Returns the time that was arrived

2.7.2.6. **setTimeArrived():** Sets the time that was arrived and stores it

2.7.2.7. **calculatePrice():** Calculates the price of the ticket based on how long the customer stayed at the garage

2.7.2.8. **printTicket():** Prints a summary on a ticket for the customer entailing the payment, ID, and paid status

2.8. Garage

2.8.1. The Garage class represents the physical parking garage in the system, tracking and managing parking capacity and availability. With attributes like garageID for unique identification, capacity for total parking spaces, and spacesTaken for currently occupied spots, this class maintains up-to-date information on parking availability.

2.8.2. Methods

2.8.2.0. parkVehicle(): Takes the vehicle and counts it temporarily within the parking space

2.8.2.1. removeVehicle(): When a customer leaves, the method removes a vehicle clearing the space and reducing the overall car count

2.8.2.2. checkSpace(): Tracks how many available spaces the

garage has left and if there is not any, it returns a message that garage is full

2.8.2.3. getID(): Returns the garages unique ID

2.8.2.4. getCapacity(): Returns how many cars are in the garage

2.8.2.5. getSpacesTaken(): gives the exact amount of how many cars are available