**SDD**

Parking Garage Project

Software Design Document

# Revision History

| Date | Revision | Description | Author |
| --- | --- | --- | --- |
| 3/22/2025 | 1.0 | Started adding content to the SDD from SRS | Kurt, Vishal, Jefferson, and Raymond |
|  |  |  |  |

# Table of Contents

[Revision History 2](#_ffcjvve8rkh1)

[Table of Contents 3](#_cl1tmj2disvg)

[1. Introduction 5](#_5el1inidzkoa)

[2. System Overview 6](#_7bkvluik8pns)

[3. Reflection on Design 7](#_tdzkndv72fod)

[4. Use Cases 8](#_yf1v1k3wn70w)

[1. Use Case: Login and Logout 8](#_yoxt4wq97c99)

[2. Use Case: Ticket Handling 8](#_z891yjif5shp)

[3. Use Case: Handling Parking Data 9](#_xx4u9rbq1r0m)

[4. Use Case: Payment Processing 9](#_lpslftu354v7)

[5. Use Case: Automatic vs. Manual Payment Handling 10](#_xppx7ykbgzqe)

[UML Use Case Diagrams Document 11](#_di1b5bmitpy4)

[1. Use Case: Login and Logout 11](#_9ipg5hxlouoe)

[2. Use Case: Ticket Handling 11](#_3qzgzd8m47zm)

[3. Use Case: Handling Parking Data 12](#_1j1wt3fyo6e)

[4. Use Case: Payment Processing 12](#_gynrsha139tv)

[5. Use Case: Automatic vs. Manual Payment Handling 13](#_si2md9gimctk)

[Class Diagrams 15](#_tky7i8xjhal1)

[Brainstorming: 15](#_axnnuwea61xx)

[ParkingGarage 15](#_eduwvhbyll)

[ParkingSpace 15](#_nsgg5lk4seyd)

[Vehicle 15](#_mbdkcxqrls1q)

[Ticket 15](#_uxn6905eabk6)

[Payment 16](#_8lbbr9yz6emk)

[Employee 16](#_qcimf0cfnr05)

[Customer 16](#_sjxpdwpeqqk)

[SystemLog 17](#_pubutwf3y636)

[Class Diagram: 17](#_jh0667uqctlr)

[Sequence Diagrams 18](#_xbho7s906lrv)

[Sequence Diagram - Employee Side 19](#_f3p3yce0r79p)

[5. System Architecture and Detailed System Design 21](#_97o7ovy64brw)

[6. Classes and Objects 22](#_w0dchs3e9l74)

[ParkingGarage 22](#_xfm3bpp5joon)

[ParkingSpace 22](#_uzgvotsu8gkl)

[Ticket 23](#_xutuc7ed8tjn)

[Payment 23](#_k4zy2ovy924n)

[Employee 23](#_k08pizpl8oau)

[Customer 24](#_7dc2bs4t29e0)

[SystemLog 24](#_arqatyf5rcnx)

# Introduction

1.1. Overview

The parking garage software delivers administrative logging reports and a graphical user interface (GUI) for users. The software will be integrated to hardware components such as automated payment kiosks, tracking monitors and as well as parking garage’s work tablets.

1.2. Purpose

The parking garage software is intended to systematically manage and streamline an overhead for parking garage operations by tracking all vehicles entering and exiting, calculating cost, monitoring available parking spaces, and logging facility-related reports.

1.3 Scope

-

1.4 Target Audience

1.4.1. Investors seeking to bring the parking garage system

1.4.2. Potential business associates such as hardware solution manufacturers and parking resources.

1.4.3. Neighboring businesses

1.4.4. Entertainment venues, sports stadiums, and large event spaces

# System Overview

2.1. Software System’s functionality

2[.1.1. Vehicle Entry & Exit Management](#)

* Issuing tickets: Generates a ticket and prints a ticket containing the start time, parking location, and a unique id.

2[.1.2. Parking Space Management](#)

* Real-time Availability Tracking: monitoring guest entering and exiting the parking facility, we are able to provide the status of available parking.

2[.1.3. Payment Processing](#)

* Automated Fee Calculation: Parking fees are calculated based on the duration of a parking spot's occupancy.
* Enables Multiple Payment Methods:
* Issuing Penalties:

2[.1.4. Logging and Garage Report](#)s

2[.1.5. Administration and Maintenance](#)

# Reflection on Design

#### 

# Use Cases

## **1. Use Case: Login and Logout**

Use Case ID: UC-01  
Relevant Requirements: Employee Login, Graphical User Interface  
Primary Actor: Employee (Attendant, Manager)  
Pre-conditions: The system must be powered on; the user must have valid login credentials.  
Post-conditions: The user is successfully logged in or out of the system.

Basic Flow:

1. The user navigates to the login page.
2. The user enters their username and password.
3. The system verifies the credentials.
4. If valid, the system grants access based on the user's role.
5. The user is redirected to the dashboard.
6. When finished, the user selects the logout option, and the session ends.

Extensions/Alternate Flows:

* If credentials are incorrect, the system displays an error message.
* If three failed attempts occur, the account is locked for security.

Exceptions:

* System error preventing login.
* Unexpected session timeout during login.

Related Use Cases: Ticket Handling, Payment Processing

## **2. Use Case: Ticket Handling**

Use Case ID: UC-02  
Relevant Requirements: Ticketing System, Parking Space Tracking  
Primary Actor: System, Customer  
Pre-conditions: The parking garage has available space; the system is active.  
Post-conditions: The customer receives a ticket, and a parking spot is reserved.

Basic Flow:

1. The customer arrives at the parking garage entry.
2. The system checks for available parking spaces.
3. If a spot is available, the system generates a ticket.
4. The system assigns a parking space and updates the total capacity.
5. The ticket is printed and given to the customer.

Extensions/Alternate Flows:

* If the garage is full, the system displays a "Garage Full" message and does not issue a ticket.
* If the ticket printer fails, an employee must manually generate a ticket.

Exceptions:

* System error preventing ticket issuance.

Related Use Cases: Handling Parking Data, Payment Processing

## **3. Use Case: Handling Parking Data**

Use Case ID: UC-03  
Relevant Requirements: Parking Space Tracking, Multi-Garage Management  
Primary Actor: System, Employee  
Pre-conditions: The system is running and actively tracking vehicles.  
Post-conditions: The parking availability data is updated accurately.

Basic Flow:

1. A vehicle enters or exits the parking garage.
2. The system updates the number of occupied and available spaces.
3. The system assigns or frees up a parking space based on the transaction.
4. The system logs the event for tracking purposes.

Extensions/Alternate Flows:

* If a sensor fails to detect a vehicle, the system logs an error.
* Employees can manually adjust parking counts in case of a discrepancy.

Exceptions:

* System communication failure with parking sensors.

Related Use Cases: Ticket Handling, Payment Processing

## **4. Use Case: Payment Processing**

Use Case ID: UC-04  
Relevant Requirements: Payment Method, Parking Duration  
Primary Actor: Customer, Employee, System  
Pre-conditions: The customer has a valid ticket and is ready to leave.  
Post-conditions: The payment is processed, and the exit gate opens.

Basic Flow:

1. The customer drives to the exit gate.
2. The system scans the ticket and calculates the fee based on duration.
3. The system prompts for payment.
4. The customer completes the payment using an accepted method (cash, card).
5. Upon successful payment, the system opens the exit gate.

Extensions/Alternate Flows:

* If the customer lost their ticket, an employee manually retrieves parking data and processes the payment.
* If the payment terminal fails, the customer must use another terminal or pay manually.

Exceptions:

* System error preventing payment processing.

Related Use Cases: Automatic vs. Manual Payment

## **5. Use Case: Automatic vs. Manual Payment Handling**

Use Case ID: UC-05  
Relevant Requirements: Employee Charging, Payment Processing  
Primary Actor: Customer, Employee, System  
Pre-conditions: The system has ticket data stored for each customer.  
Post-conditions: The customer is charged correctly based on the method used.

Basic Flow (Automatic Payment):

1. A customer with an auto-payment account is detected at the exit.
2. The system verifies the linked payment method and calculates the fee.
3. The system processes the payment automatically.
4. If successful, the exit gate opens.

Basic Flow (Manual Payment):

1. A customer without auto-payment drives to an employee checkout booth.
2. The employee scans the ticket and retrieves the parking fee.
3. The customer makes the payment.
4. The employee confirms payment and allows exit.

Extensions/Alternate Flows:

* If automatic payment fails, the system prompts for manual payment.
* If the employee is unable to process a payment, the customer must use another payment method.

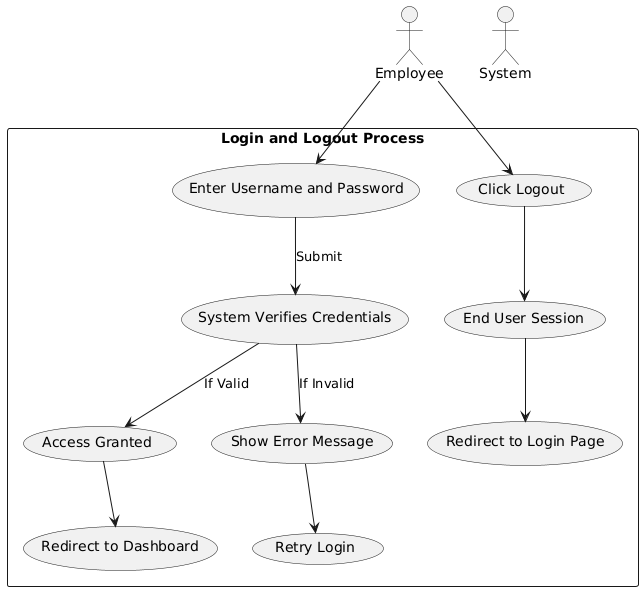
Exceptions:

* System failure preventing payment authorization.

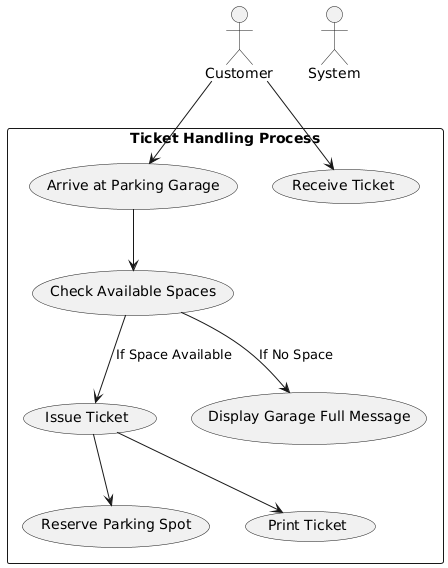
Related Use Cases: Payment Processing, Ticket Handling

# UML Use Case Diagrams Document

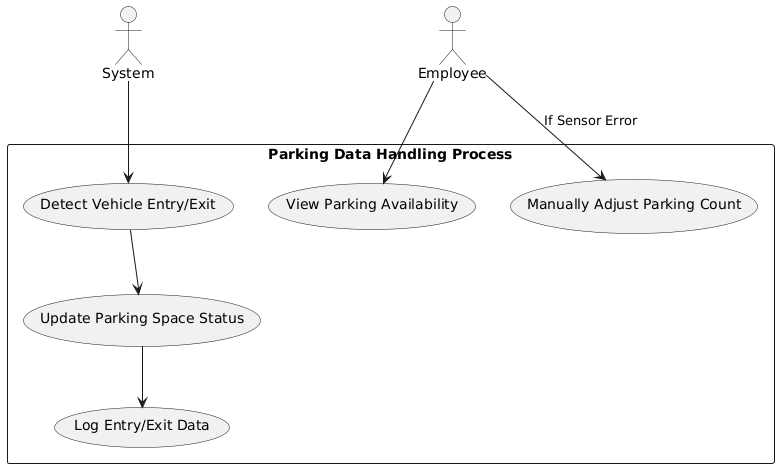
## **1. Use Case: Login and Logout**



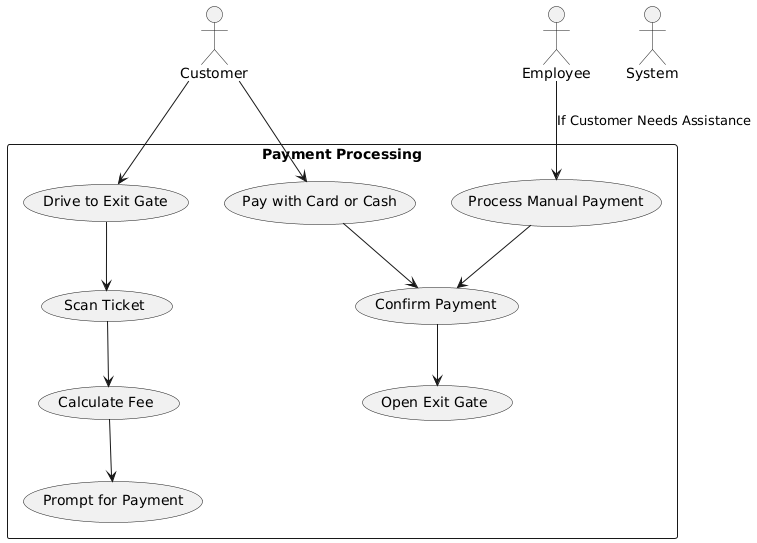
## **2. Use Case: Ticket Handling**



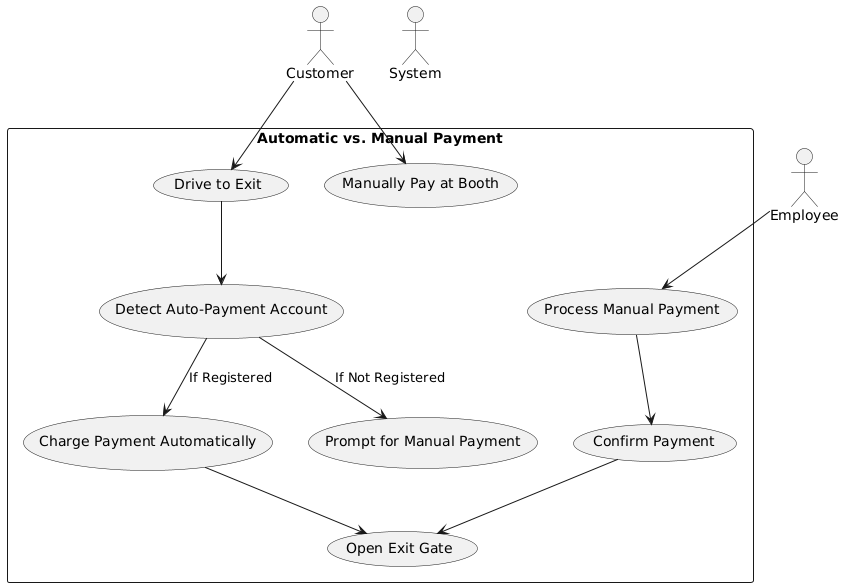
## **3. Use Case: Handling Parking Data**



## **4. Use Case: Payment Processing**



## **5. Use Case: Automatic vs. Manual Payment Handling**



# Class Diagrams

## **Brainstorming:**

#### ParkingGarage

What do I know?

* Data encapsulated: garageID, location, totalSpaces, availableSpaces, spaces (List<ParkingSpace>)

What do I do?

* Methods:
  + isFull(): Returns whether the garage is at full capacity.
  + updateAvailability(): Updates the count of available spaces.

#### ParkingSpace

What do I know?

* Data encapsulated: spaceID, isOccupied

What do I do?

* Methods:
  + assignVehicle(Vehicle v): Marks the space as occupied by a vehicle.
  + removeVehicle(): Frees up the space.

#### Vehicle

What do I know?

* Data encapsulated: licensePlate, vehicleType

What do I do?

* No methods (Vehicle only stores data).

#### Ticket

What do I know?

* Data encapsulated: ticketID, entryTime, exitTime, fee

What do I do?

* Methods:
  + calculateFee(): Calculates the total parking fee based on duration.

#### Payment

What do I know?

* Data encapsulated: paymentID, amount, paymentMethod, isPaid

What do I do?

* Methods:
  + processPayment(): Marks the payment as completed.

#### Employee

What do I know?

* Data encapsulated: employeeID, name, role

What do I do?

* Methods:
  + processTicket(): Handles ticket verification and processing.
  + handlePayment(): Manages customer payments.

#### Customer

What do I know?

* Data encapsulated: name, contactInfo

What do I do?

* Methods:
  + enterGarage(): Allows the customer to enter the parking garage.
  + exitGarage(): Allows the customer to exit after payment.

#### SystemLog

What do I know?

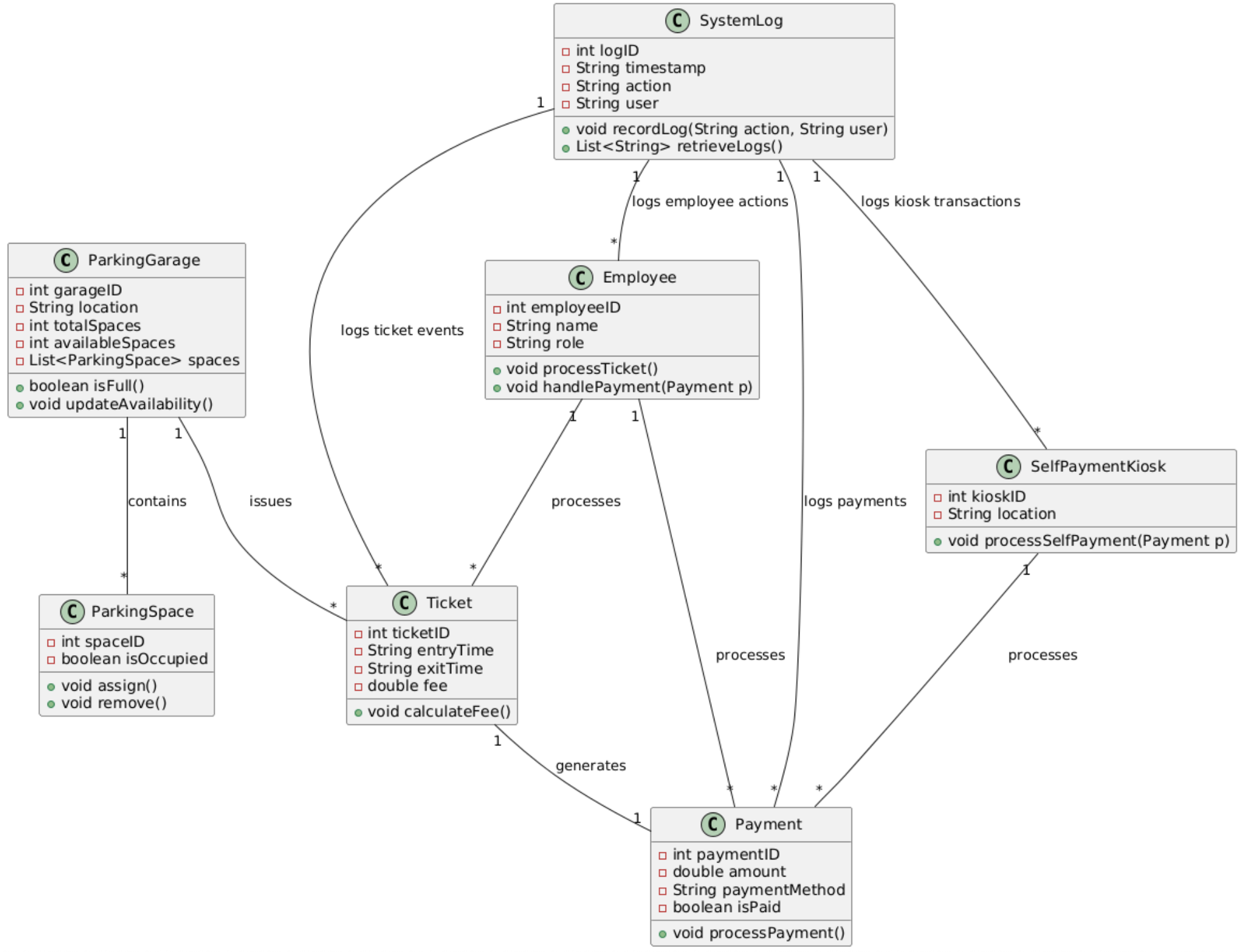
* Data encapsulated: logID, eventDetails, timestamp

What do I do?

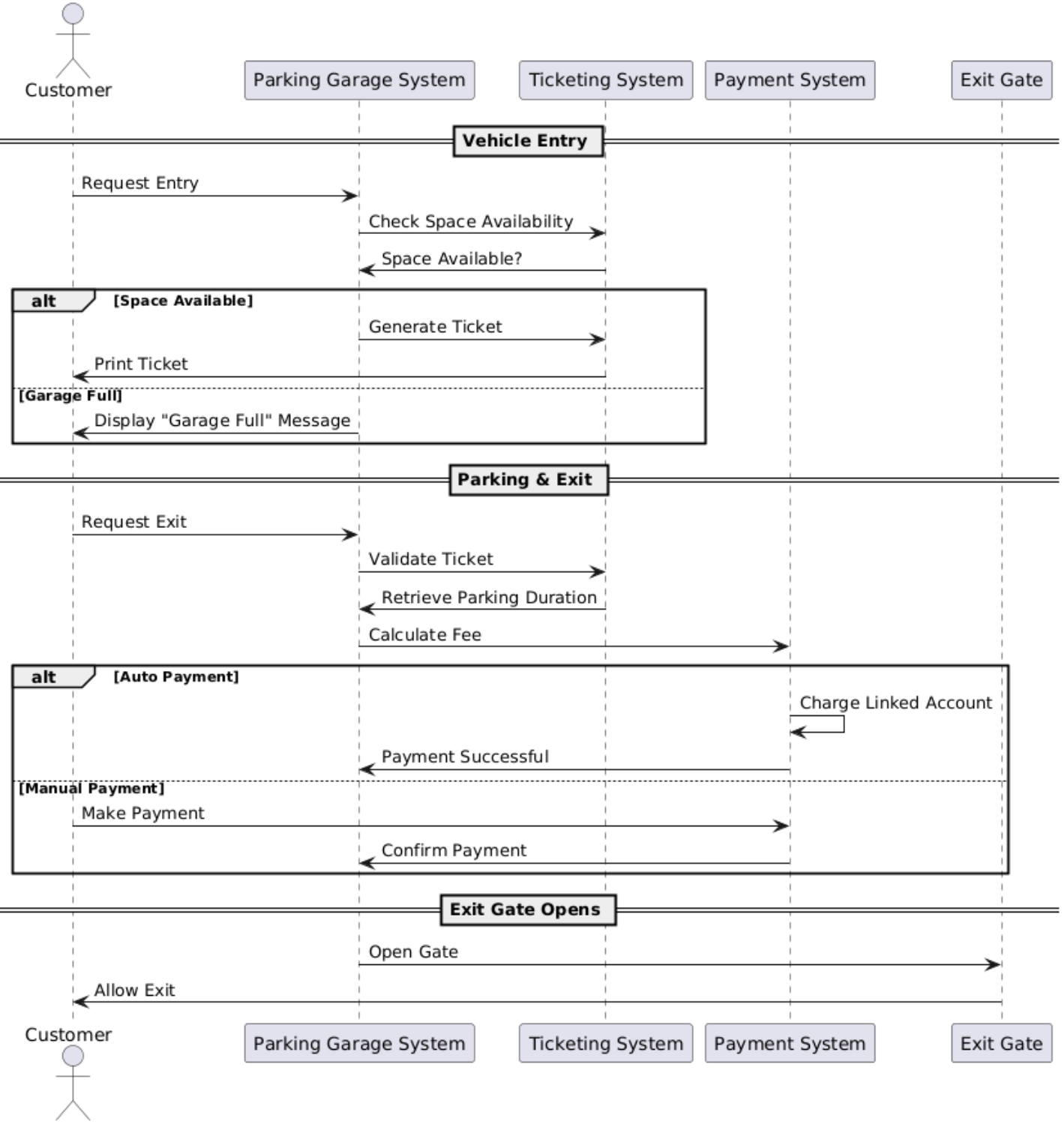
* Methods:
  + recordEvent(): Logs system activities such as vehicle entries, payments, and errors.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

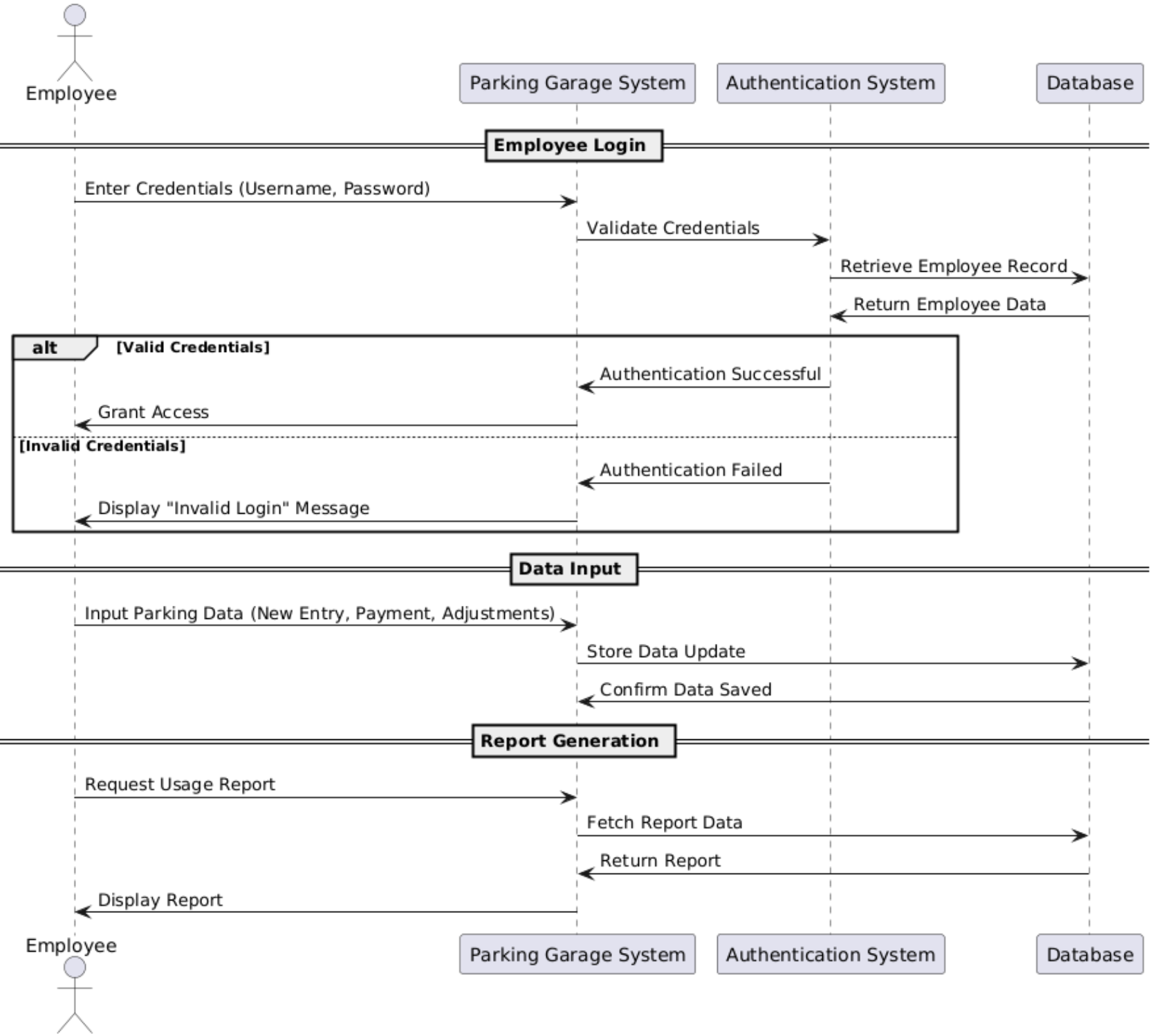
### **Class Diagram:**



# Sequence Diagrams



## **Sequence Diagram - Employee Side**



# System Architecture and Detailed System Design

# Classes and Objects

#### ParkingGarage

Class Variables

* garageID
* Location
* totalSpaces
* availableSpaces
* spaces (List<ParkingSpace>)

Methods

* isFull(): Returns whether the garage is at full capacity.
* updateAvailability(): Updates the count of available spaces.

#### ParkingSpace

Class Variables

* Data encapsulated: spaceID, isOccupied

Methods

* assignVehicle(Vehicle v): Marks the space as occupied by a vehicle.
* removeVehicle(): Frees up the space.

#### 

#### Ticket

Class Variables

* Data encapsulated: ticketID, entryTime, exitTime, fee

Methods

* calculateFee(): Calculates the total parking fee based on duration.

#### 

#### Payment

Class Variables

* Data encapsulated: paymentID, amount, paymentMethod, isPaid

Methods

* processPayment(): Marks the payment as completed.

#### 

#### Employee

Class Variables

* Data encapsulated: employeeID, name, role

Methods

* processTicket(): Handles ticket verification and processing.
* handlePayment(): Manages customer payments.

#### 

#### 

#### Customer

Class Variables

* Data encapsulated: name, contactInfo

Methods

* enterGarage(): Allows the customer to enter the parking garage.
* exitGarage(): Allows the customer to exit after payment.

#### 

#### SystemLog

Class Variables

* Data encapsulated: logID, eventDetails, timestamp

Methods

* recordEvent(): Logs system activities such as vehicle entries, payments, and errors.

Notes Tab

Tips for Creating Effective SDDs:

* Keep it Concise and Clear:  
  SDDs should be easy to read and understand, avoiding jargon and technical terms that are not widely known.
* Use Visual Aids:  
  Incorporate diagrams, flowcharts, and other visual aids to illustrate the software's architecture and functionality.
* Focus on Trade-offs:  
  Document the trade-offs made during the design process, explaining why certain choices were made over others.
* Get Feedback Early:  
  Share the SDD with stakeholders and developers early in the process to gather feedback and ensure alignment.
* Keep it Up-to-Date:  
  SDDs should be updated as the project progresses, reflecting any changes in the design or implementation.

Thread Implementation

* Client / Server
* GUI

Potentially Singleton for parking garage’s availability or capacity (if keeping count)

* Updates to resource count
* Or updates on resource isAvailable

State and statuses of certain classes - potential enum types

Creating Console/GUI classes

Message objects (maybe) via server client

Definitions, Acronyms, Abbreviations

* Important terms
  + Parking Garage Management System (PGMS): The software and hardware system used to manage parking spaces, transactions, and customer entry/exit in a parking facility.
  + Entry/Exit Log: A record of vehicles entering and leaving the parking garage, including timestamps and vehicle details.
  + Multithreading: A technique used to allow multiple tasks to run concurrently within the same program, improving performance and efficiency.
  + Payment Gateway: A system that processes financial transactions, allowing customers to pay for parking services.
  + Server: The central computing unit that manages and coordinates the operations of the parking garage system, including handling multiple garages and transactions.
  + Thread: A lightweight process that performs tasks concurrently with other threads in the system, used to handle multiple customer actions simultaneously.
  + Employee: A worker who oversees the system and its responsibilities in managing the network of parking garages.
  + Unified Modeling Language (UML): Modeling used to visualize system design.
* Useful abbreviations
  + GUI - Graphical User Interface
  + API - Application Programming Interface