# Project Documentation – ParkEase System

## 1. Project Overview

Project Name: ParkEase System  
Technology Used: Java  
Concepts Implemented:  
- Interface (ParkingOperations)  
- Abstraction (Abstract class Vehicle)  
- Inheritance (SimpleVehicle extends Vehicle)  
- Encapsulation (Private fields with getters/setters in updated version)  
- Nested Class (ParkingLot.Slot)  
- Static Members (Static variables and constants in ParkingLot)  
- OOP Concepts (Classes, Objects, Polymorphism, Encapsulation, Inheritance, Abstraction)  
- File Handling (CSV export for checkout history)  
Purpose:  
The ParkEase System is a parking lot management application that allows users to park and remove vehicles, view parking status, and automatically calculate charges. It ensures that only 2-wheelers and 4-wheelers are allowed, and maintains a record of checkout history in a CSV file.

## 2. Objectives

* Efficient Parking Management – Allocate slots dynamically and release them when vehicles are removed.
* Automated Fare Calculation – Calculate charges based on vehicle type and parking duration.
* Data Recording – Maintain checkout history in a CSV file for record-keeping.
* User-Friendly Interaction – Provide a menu-driven interface for easy operations.
* Maintain OOP Standards – Implement abstraction, encapsulation, inheritance, and interface usage.

## 3. System Features

* Vehicle Parking: Accepts only 2 or 4-wheelers and allocates the first available slot.
* Vehicle Removal: Calculates parking charges based on duration and vehicle type, with an option to display a receipt after removal, and stores details in checkout\_history.csv.
* Slot Display: Shows the status of all parking slots (occupied or empty).
* Receipt Generation: Displays vehicle details, duration, and charges in a formatted manner.

## 4. System Flow

1. User chooses an operation from the menu.
2. If Parking: User enters vehicle number & wheel count, system validates and parks in the first available slot.
3. If Removing: User enters slot number and hours parked, system calculates charges and displays amount, user decides whether to view detailed receipt, details are recorded in a CSV file.
4. If Displaying Slots: System lists all slots and their statuses.
5. Repeat until Exit option is selected.

## 5. Class Diagram (UML Representation)

+------------------+ +------------------+  
| <<interface>> | | Vehicle |  
| ParkingOperations |<------>|------------------|  
|-------------------| |- number: String |  
|+ parkVehicle() | |- wheels: int |  
|+ removeVehicle() | |------------------|  
|+ displaySlots() | |+ getType():String|  
+-------------------+ +------------------+  
 ^ ^  
 | |  
 | |  
+------------------+ +------------------+  
| ParkingLot | | SimpleVehicle |  
|------------------| |------------------|  
|- MAX\_SLOTS: int | |+ getType():String|  
|- currentSlots: int| +------------------+  
|- slots: Slot[] |  
|------------------|  
|+ parkVehicle() |  
|+ removeVehicle() |  
|+ displaySlots() |  
+--------+---------+  
 |  
 | (Nested Class)  
 v  
 +-----------+  
 | Slot |  
 |-----------|  
 |- slotNumber: int |  
 |- vehicle: Vehicle|  
 +-----------+

## 6. Applied OOP Concepts

* Encapsulation – Vehicle fields are private with controlled access via getters/setters.
* Abstraction – Vehicle is an abstract class hiding implementation details.
* Inheritance – SimpleVehicle inherits from Vehicle.
* Polymorphism – Method overriding in SimpleVehicle.getType().
* Interface Implementation – ParkingLot implements ParkingOperations.
* Nested Class – ParkingLot.Slot used to represent each slot.
* Static Members – Used for MAX\_SLOTS and currentSlots.

## 7. File Handling

Uses BufferedWriter with FileWriter to append checkout history.  
Automatically creates checkout\_history.csv if it doesn’t exist.  
Appends new records without overwriting old data.  
Sample CSV Output:  
Vehicle Number,Type,Hours Parked,Charge  
KA01AB1234,2-Wheeler,3,15  
KA02BC5678,4-Wheeler,2,20

## 8. Example Output

=== Parking Lot Menu ===  
1. Park Vehicle  
2. Remove Vehicle  
3. Display Slots  
4. Exit  
Choose an option: 1  
Enter vehicle number: KA01AB1234  
Enter number of wheels (2 or 4): 2  
[SUCCESS] Parked at slot 1  
  
Choose an option: 2  
Enter slot number to remove: 1  
Enter number of hours parked: 3  
Charge: Rs. 15  
Do you want to view receipt? (yes/no): yes  
--- Receipt ---  
Vehicle Number: KA01AB1234  
Vehicle Type: 2-Wheeler  
Hours Parked: 3  
Charge: Rs. 15  
----------------

## 9. Future Enhancements

* Add Date & Time Tracking for automatic hours calculation.
* Integrate a GUI using JavaFX or Swing.
* Support multiple floors in the parking lot.
* Implement online payment integration.

# 10. Applicable SOLID Principles

The ParkEase System incorporates certain principles from the SOLID design philosophy:  
  
1. Single Responsibility Principle (SRP)–   
 Each class in the system has a single, well-defined responsibility. For example:  
 - `Vehicle` handles vehicle-related properties and behavior.  
 - `ParkingLot` manages slot allocation, removal, and display.  
 - `Slot` (nested class) only stores and manages slot-specific details.  
  
2. Liskov Substitution Principle (LSP)–   
 Any subclass of `Vehicle` (e.g., `SimpleVehicle`) can be used in place of its parent class without breaking functionality.

## 11. Conclusion

The ParkEase System successfully implements a menu-driven parking lot management solution using core Java concepts. It demonstrates the use of OOP principles, interfaces, nested classes, encapsulation, and file handling in a single cohesive project. The code structure is modular, making it easy to understand, maintain, and extend for future requirements.