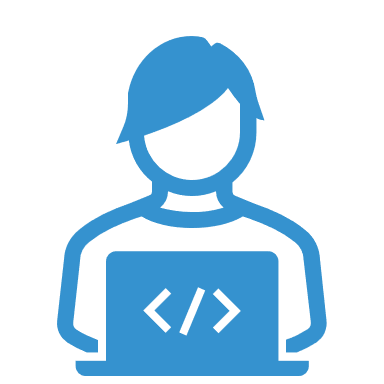


|  |
| --- |
|  |
| Group 7 Report  2023 |
| September 23  COMPANY NAME  Authored by Veerain Sood, Mokshit Reddy, ….. |



# Review of Our Project…

|  |
| --- |
| OOPs has been a very crucial way for maintaining the structure and readability of our programs. During our quest to build our Parking lot, we made many mistakes and learned how to make the program more dynamic.  Code Explanation for the Design Decisions:  Abstract Class Vehicle:   * This class represents a vehicle and uses the protected access modifier for fields like size, token, numberPlate, electric, paid, setPaidStatus(), and setToken(),to ensure that they are accessible within subclasses but not directly from outside the package. * Methods like paid\_status() are marked public for the vehicle owner to access them. * The setStartTime() and setEndTime () fields are used to track the parking duration, and methods like getTimeElapsedMinutes() are marked as protected to control access based on a government ID. * Methods like setParkingFloor(), setbuilding(), charging\_by\_cost(), printstatus(), payatportal(), and exit() are used for keeping track of where the vehicle is parked and to show floor parking availability. * Marked as public so that classes which import this package are able to access this class. * Usage of java.time.instant, java.time.duration classes to manage time of entry and exit of user.   Class userVehicle:   * This class represents a users vehicle and inherits from the vehicle abstract class. It is marked as public to allow external creation of userVehicle objects.   Class Payments:   * This class handles payment processing. It contains a private method checkpin() and a public static method Payment(). The checkpin() method is private to restrict direct access, while the Payment() method is marked as protected to allow payment processing from verified classes(not random ones) enhancing the security..   Class Building:   * This class represents a building with multiple floors. It contains references to individual floor instances (ground, first, second, third), to be able to keep track of floors of each building. * The freeFloor() method is marked as public and is used to find a free floor based on the vehicle type (electric or fossil) and size of the vehicle(#wheels). * The Capacityboard() method provides information about floor capacity and parking availability. * Methods like sendToFloor() and Capacityboard() have appropriate access modifiers to allocate a vehicle to each floor.   Class floors:   * This class represents individual floors in the parking lot. It inherits from Payments. It contains fields and methods to manage floor capacity and parking. * The paymentPortal() method is marked as protected and is used for payment processing at the payment portal which should not be accessed by other classes randomly(only those which inherit it).   Inner classes like entryPoint and exitPoint handle vehicle entry and exit and have appropriate access modifiers.  Use of OOP Concepts:  Inheritance: The userVehicle class inherits from the vehicle abstract class, allowing for code reuse and specialization of vehicle types.  Encapsulation: Access modifiers such as protected, private, and public are used to encapsulate data and control access to class members. For example, vehicle details are encapsulated within the vehicle class.  Polymorphism: Polymorphism is demonstrated by the class capacityboards which contains the function which capacity board which is implemented differently (overrode) by Buildings class(electric or fossil).  Abstraction: The vehicle class serves as an abstract representation of a vehicle with common attributes and behaviors.  Data Hiding: The use of access modifiers hides implementation details and allows controlled access to class members.  Overall, the code demonstrates a structured and object-oriented approach to building a parking lot management system in Java, with a focus on encapsulation, inheritance, and controlled access to class members. |

An example output Program

Assuming you provide the following inputs when running your Java program:

1. First Vehicle Entry:
   * Number of wheels: 4
   * Electric (1 for yes, 0 for no): 1
   * Number Plate: 12345
   * Payment at exit: 1 (yes)
2. Second Vehicle Entry:
   * Number of wheels: 2
   * Electric (1 for yes, 0 for no): 0
   * Number Plate: 54321
   * Payment at exit: 1 (yes)
3. Simultaneous Entry of Two Vehicles:
   * Vehicle 1 (Jeep):
     + Number of wheels: 4
     + Electric (1 for yes, 0 for no): 1
     + Number Plate: 11111
   * Vehicle 2 (Truck):
     + Number of wheels: 6
     + Electric (1 for yes, 0 for no): 0
     + Number Plate: 22222

Here is the structured demonstration of the expected output:

\*\*\*Welcome to MNLVS ParkingLot\*\*\*

Enter the type of vehicle (#wheels):

4

Enter the type of vehicle (Electrical or not) Enter 1 if electrical:

1

Enter the numberPlate of the vehicle: 12345

(Simulated messages about parking and charging)

Do you wish to pay at exit or at the payment portal?

Enter 1 if you wish to pay at the portal: 1

(Simulated status of the parking lot after the first vehicle exits) ..

Enter the type of vehicle (#wheels): 2

Enter the type of vehicle (Electrical or not) Enter 1 if electrical:

0

Enter the numberPlate of the vehicle: 54321

(Simulated messages about parking) ...

Do you wish to pay at exit or at the payment portal? Enter 1 if you wish to pay at the portal:

1

(Simulated messages about payment and exit) ..

. (Simulated status of the parking lot after the second vehicle exits)

(Simulated messages for simultaneous entry of Jeep and Truck) ...

(Simulated messages about payment and exit for Jeep and Truck) ...

(Simulated status of the parking lot after Jeep and Truck exit) ...