# Vehicle Information System

This Java application simulates a Car Rental Agency’s Vehicle Information System. It uses interfaces to enforce a common structure for various types of vehicles cars, motorcycles, and trucks, while allowing each to store and retrieve specific attributes. The application demonstrates principles of interface implementation, modular design, user input handling, and object-oriented programming best practices.

## System Components

1. Interface: GeneralVehicle

This base interface defines common vehicle behavior. It includes the following methods:

getVehicleMake()

getVehicleModel()

getVehicleYear()

Every vehicle class must implement this interface, ensuring a standard contract for retrieving make, model, and year.

### 2. Interface: CarDetails

This interface is implemented by the RentalCar class and includes:

setDoorCount(int doors)

getDoorCount()

setFuelVariant(String fuel)

getFuelVariant()

It ensures that car objects can store and retrieve their number of doors and fuel type (e.g., petrol, diesel, electric).

### 3. Interface: MotorcycleSpecs

This interface is used by the RentalMotorcycle class and defines:

setWheelCount(int wheels)

getWheelCount()

setMotorType(String type)

getMotorType()

It allows motorcycle objects to maintain information about their wheel count and motorcycle category (e.g., sport, cruiser, off-road).

### 4. Interface: TruckSpecs

Implemented by the RentalTruck class, this interface provides:

setCargoCapacity(double tons)

getCargoCapacity()

setTransmissionKind(String transmission)

getTransmissionKind()

Trucks can store their cargo capacity (in tons) and transmission type (manual or automatic).

## Class Implementations

Each vehicle type is represented by a class that implements both the base interface and its respective specialized interface.

RentalCar implements GeneralVehicle and CarDetails.

RentalMotorcycle implements GeneralVehicle and MotorcycleSpecs.

RentalTruck implements GeneralVehicle and TruckSpecs.

These classes include constructors for setting values and provide implementations for all interface methods. This enforces data consistency and code reusability.

## Main Program: VehicleAppLauncher

The main method performs the following steps:

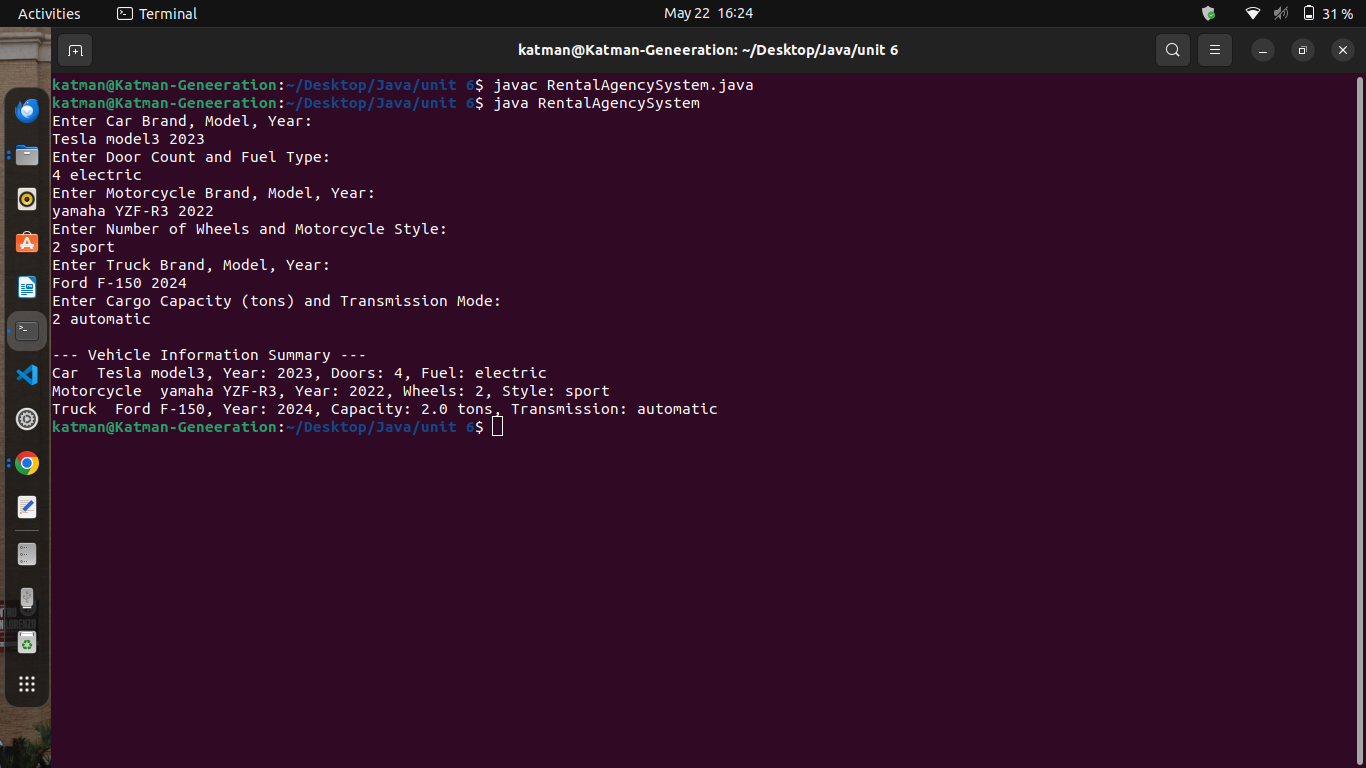
User Interaction: Uses Scanner to collect input for each type of vehicle, including make, model, year, and type-specific attributes (e.g., doors, wheels, fuel type, etc.).

Object Creation: Instantiates RentalCar, RentalMotorcycle, and RentalTruck using the provided data.

Output Display: Prints formatted details of each vehicle to the console using accessor methods.

Error Handling: Ensures input is valid (e.g., numeric year or capacity), and gracefully handles any incorrect inputs.

### Sample Output



### Conclusion

This program demonstrates a clean, interface-based architecture to represent diverse vehicle types in a rental agency system. It ensures data consistency, code organization, and easy maintenance, while offering a user-friendly interface to gather and display information interactively. This kind of structured application is valuable in enterprise environments where consistency and scalability are key.