***CASE STUDY NO. 1***

* **Title**
* Model a real world scenario (vehicle class, fruit class, student management in university etc.) using Object Oriented Paradigm.
* **Theory Concepts Of Java/OOP Used in This Case Study**

1. **Class**: A class is a blueprint for creating objects, defining the properties (attributes) and behaviors (methods) that objects of the class will have. It specifies how an object should look and act. In this example, Vehicle, Car, and Bike are classes that model different types of vehicles, with shared and specific properties and behaviors.
2. **Object**: An object is an instance of a class that holds actual data. It represents a specific entity created based on the class definition. For instance, car and bike are objects of the Car and Bike classes, with unique data such as brand, speed, and additional properties specific to the type of vehicle.
3. **Inheritance**: Inheritance allows a class to inherit properties and methods from another class, promoting code reuse and logical hierarchy. The subclass inherits the attributes and methods of the superclass. In this case, both Car and Bike inherit from the Vehicle class, gaining its attributes like brand and speed, as well as the displayDetails() method.
4. **Encapsulation**: Encapsulation involves restricting access to an object's internal data and providing controlled access through public methods. This ensures that an object’s data is protected from unwanted changes. In the program, the Vehicle class uses encapsulation by making fields like brand and speed private and providing access through constructors and methods.
5. **Polymorphism**: Polymorphism allows methods to have different implementations based on the object calling them, enabling objects of different classes to respond differently to the same method. In the program, the displayDetails() method is overridden in both Car and Bike classes to provide specific details related to each vehicle, even though they share the same method name.
6. **Constructor**: A constructor is a special method in a class used to initialize objects. It is called when an object is created and assigns initial values to the object's fields. In this program, the constructors in Vehicle, Car, and Bike are used to initialize the brand, speed, and additional properties like doors or hasCarrier.

* **Algorithm:**

1. **Start**
2. Define a Vehicle class with the following:
   * Attributes: brand and speed.
   * A constructor to initialize the attributes.
   * A method displayDetails() to print the brand and speed.
3. Define a Car class that extends Vehicle:
   * Add an additional attribute: doors.
   * Create a constructor to initialize brand, speed, and doors using super().
   * Override the displayDetails() method to print the doors along with brand and speed.
4. Define a Bike class that extends Vehicle:
   * Add an additional attribute: hasCarrier.
   * Create a constructor to initialize brand, speed, and hasCarrier using super().
   * Override the displayDetails() method to print if the bike has a carrier along with brand and speed.
5. Define the VehicleManagement class with the main() method:
   * Create a Car object with values for brand, speed, and doors.
   * Create a Bike object with values for brand, speed, and hasCarrier.
   * Print the details of the Car using its displayDetails() method.
   * Print the details of the Bike using its displayDetails() method.
6. **End**

* **Flowchart:**

A diagram of a car

Description automatically generated

* **Program/Code:**
* class Vehicle

{

    private String brand;

    private int speed;

    public Vehicle(String brand, int speed)

    {

        this.brand = brand;

        this.speed = speed;

    }

    public void displayDetails()

    {

        System.out.println("Brand: " + brand + ", Speed: " + speed + " km/h");

    }

}

class Car extends Vehicle

{

    private int doors;

    public Car(String brand, int speed, int doors)

    {

        super(brand, speed);

        this.doors = doors;

    }

    @Override

    public void displayDetails()

    {

        super.displayDetails();

        System.out.println("Doors: " + doors);

    }

}

class Bike extends Vehicle

{

    private boolean hasCarrier;

    public Bike(String brand, int speed, boolean hasCarrier)

    {

        super(brand, speed);

        this.hasCarrier = hasCarrier;

    }

    @Override

    public void displayDetails()

    {

        super.displayDetails();

        System.out.println("Has Carrier: " + (hasCarrier ? "Yes" : "No"));

    }

}

public class VehicleManagement

{

    public static void main(String[] args)

    {

        Car car = new Car("Toyota", 180, 4);

        Bike bike = new Bike("Yamaha", 120, true);

        System.out.println("=== Car Details ===");

        car.displayDetails();

        System.out.println("\n=== Bike Details ===");

        bike.displayDetails();

    }

}

* **Output:**
* === Car Details ===

Brand: Toyota, Speed: 180 km/h

Doors: 4

=== Bike Details ===

Brand: Yamaha, Speed: 120 km/h

Has Carrier: Yes

* **Conclusion**
* The Vehicle Management System demonstrates how real-world objects like cars and bikes can be modeled using Object-Oriented Paradigm principles. By leveraging concepts like inheritance, encapsulation, and polymorphism, the program is structured, reusable, and easy to maintain. This approach ensures scalability and efficiencgy in managing additional vehicle types in the future.