**INHERITANCE**

**Assisted Problems**

1. **Animal Hierarchy**

**Description:** Create a hierarchy where Animal is the superclass, and Dog, Cat, and Bird are subclasses. Each subclass has a unique behavior.

**Tasks:**

1. Define a superclass Animal with attributes name and age, and a method makeSound().
2. Define subclasses Dog, Cat, and Bird, each with a unique implementation of makeSound().

**Goal:** Learn basic inheritance, method overriding, and polymorphism with simple classes.

class Animals {  
 String name;  
 int age;  
  
 public Animals(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void makeSound() {  
 System.*out*.println("Animals give sound.");  
 }  
}  
  
class Dog extends Animals {  
  
 public Dog(String name, int age) {  
 super(name, age);  
 }  
  
 @Override  
 public void makeSound() {  
 System.*out*.println("Dog Barks");  
 }  
}  
  
class Cat extends Animals {  
  
 public Cat(String name, int age) {  
 super(name, age);  
 }  
  
 @Override  
 public void makeSound() {  
 System.*out*.println("Cat meow");  
 }  
}  
  
class Bird extends Animals {  
  
 public Bird(String name, int age) {  
 super(name, age);  
 }  
  
 @Override  
 public void makeSound() {  
 System.*out*.println("Bird chirp");  
 }  
}  
  
public class Animal {  
  
 public static void main(String[] args) {  
  
 Animals dog = new Dog("Jimmy", 5);  
 Animals cat = new Cat("Tom", 4);  
 Animals bird = new Bird("Tuttu", 2);  
  
 dog.makeSound();  
 cat.makeSound();  
 bird.makeSound();  
 }  
}

**OUTPUT:**

Dog Barks

Cat meow

Bird chirp

1. **Employee Management System**

**Description**: Create an Employee hierarchy for different employee types such as Manager, Developer, and Intern.

**Tasks**:

* 1. Define a base class Employee with attributes like name, id, and salary, and a method displayDetails().
  2. Define subclasses Manager, Developer, and Intern with unique attributes for each, like teamSize for Manager and programmingLanguage for Developer.

**Goal**: Practice inheritance by creating subclasses with specific attributes and overriding superclass methods.

public class Employee {  
  
 public String name;  
 public int id;  
 public double salary;  
  
 public Employee(String name, int id, int salary) {  
 this.name = name;  
 this.id = id;  
 this.salary = salary;  
 }  
  
 public void displayDetails() {  
 System.*out*.println("Employee Name: " + name);  
 System.*out*.println("Employee Id: " + id);  
 System.*out*.println("Salary: " + salary);  
 }  
}  
  
class Manager extends Employee {  
  
 private int teamSize;  
  
 public Manager(String name, int id, int salary, int teamSize) {  
 super(name, id, salary);  
 this.teamSize = teamSize;  
 }  
  
 @Override  
  
 public void displayDetails() {  
 System.*out*.println("Employee Name: " + name);  
 System.*out*.println("Employee Id: " + id);  
 System.*out*.println("Salary: " + salary);  
 System.*out*.println("Team Size: " + teamSize);  
 }  
}  
  
class Developer extends Employee {  
  
 private String language;  
  
 public Developer(String name, int id, int salary, String language) {  
 super(name, id, salary);  
 this.language = language;  
 }  
  
 @Override  
  
 public void displayDetails() {  
 System.*out*.println("Employee Name: " + name);  
 System.*out*.println("Employee Id: " + id);  
 System.*out*.println("Salary: " + salary);  
 System.*out*.println("Programming Language: " + language);  
 }  
}  
  
class Intern extends Employee {  
  
 private int experience;  
  
 public Intern(String name, int id, int salary, int experience) {  
 super(name, id, salary);  
 this.experience = experience;  
 }  
  
 @Override  
  
 public void displayDetails() {  
 System.*out*.println("Employee Name: " + name);  
 System.*out*.println("Employee Id: " + id);  
 System.*out*.println("Salary: " + salary);  
 System.*out*.println("Intern Experience: " + experience);  
 }  
}  
  
class main {  
  
 public static void main(String[] args) {  
  
 Employee manager = new Manager("Alex", 101, 50000, 5);  
 Employee developer = new Developer("Dennis", 202, 45000, "Java");  
 Employee intern = new Intern("Richard", 303, 15000, 2);  
  
 manager.displayDetails();  
 System.*out*.println();  
 developer.displayDetails();  
 System.*out*.println();  
 intern.displayDetails();  
 }  
 }

}  
}

**OUTPUT:**

Employee Name: Alex

Employee Id: 101

Salary: 50000.0

Team Size: 5

Employee Name: Dennis

Employee Id: 202

Salary: 45000.0

Programming Language: Java

Employee Name: Richard

Employee Id: 303

Salary: 15000.0

Intern Experience: 2

1. **Vehicle and Transport System**
   1. **Description**: Design a vehicle hierarchy where Vehicle is the superclass, and Car, Truck, and Motorcycle are subclasses with unique attributes.
   2. **Tasks**:
      1. Define a superclass Vehicle with maxSpeed and fuelType attributes and a method displayInfo().
      2. Define subclasses Car, Truck, and Motorcycle, each with additional attributes, such as seatCapacity for Car.
      3. Demonstrate polymorphism by storing objects of different subclasses in an array of Vehicle type and calling displayInfo() on each.
   3. **Goal**: Understand how inheritance helps in organizing shared and unique features across subclasses and use polymorphism for dynamic method calls.

* class Vehicle {  
   int maxSpeed;  
   String fuelType;  
    
   public Vehicle(int maxSpeed, String fuelType) {  
   this.maxSpeed = maxSpeed;  
   this.fuelType = fuelType;  
   }  
    
   public void displayInfo() {  
   System.*out*.println("Max Speed: " + maxSpeed + " km/h, Fuel Type: " + fuelType);  
   }  
  }  
    
  class Car extends Vehicle {  
   int seatCapacity;  
    
   public Car(int maxSpeed, String fuelType, int seatCapacity) {  
   super(maxSpeed, fuelType);  
   this.seatCapacity = seatCapacity;  
   }  
    
   @Override  
   public void displayInfo() {  
   System.*out*.println("Car -> \nMax Speed: " + maxSpeed + " km/h \nFuel Type: " + fuelType + "\nSeat Capacity: " + seatCapacity);  
   }  
  }  
    
  class Truck extends Vehicle {  
   int loadCapacity;  
    
   public Truck(int maxSpeed, String fuelType, int loadCapacity) {  
   super(maxSpeed, fuelType);  
   this.loadCapacity = loadCapacity;  
   }  
    
   @Override  
   public void displayInfo() {  
   System.*out*.println("Truck -> \nMax Speed: " + maxSpeed + " km/h \nFuel Type: " + fuelType + "\nLoad Capacity: " + loadCapacity + " tons");  
   }  
  }  
    
  class Motorcycle extends Vehicle {  
   boolean hasSidecar;  
    
   public Motorcycle(int maxSpeed, String fuelType, boolean hasSidecar) {  
   super(maxSpeed, fuelType);  
   this.hasSidecar = hasSidecar;  
   }  
    
   @Override  
   public void displayInfo() {  
   System.*out*.println("Motorcycle -> \nMax Speed: " + maxSpeed + " km/h \nFuel Type: " + fuelType + " \nSidecar: " + (hasSidecar ? "Yes" : "No"));  
   }  
  }  
    
  class VehicleTransportSystem {  
   public static void main(String[] args) {  
    
   Vehicle car = new Car(220, "Petrol", 5);  
   Vehicle truck = new Truck(120, "Diesel", 15);  
   Vehicle motorcycle = new Motorcycle(180, "Petrol", false);  
    
   car.displayInfo();  
   System.*out*.println();  
   truck.displayInfo();  
   System.*out*.println();  
   motorcycle.displayInfo();  
   }  
  }

**OUTPUT:**

Car ->

Max Speed: 220 km/h

Fuel Type: Petrol

Seat Capacity: 5

Truck ->

Max Speed: 120 km/h

Fuel Type: Diesel

Load Capacity: 15 tons

Motorcycle ->

Max Speed: 180 km/h

Fuel Type: Petrol

Sidecar: No

**SINGLE INHERITANCE**

**Sample Problem 1: Library Management with Books and Authors**

* **Description**: Model a Book system where Book is the superclass, and Author is a subclass.
* **Tasks**:
  + Define a superclass Book with attributes like title and publicationYear.
  + Define a subclass Author with additional attributes like name and bio.
  + Create a method displayInfo() to show details of the book and its author.
* **Goal**: Practice single inheritance by extending the base class and adding more specific details in the subclass.

class Book {  
  
 public String title;  
 public int publicationYear;  
  
 public Book(String title, int publicationYear) {  
 this.title = title;  
 this.publicationYear = publicationYear;  
 }  
  
 public void displayInfo() {  
 System.*out*.println("Title Name: " + title);  
 System.*out*.println("Publication Year: " + publicationYear);  
 }  
}  
  
class Author extends Book {  
  
 private String author;  
 private String bio;  
  
 public Author (String title, int publicationYear, String author, String bio) {  
 super(title, publicationYear);  
 this.author = author;  
 this.bio = bio;  
 }  
  
 @Override  
 public void displayInfo() {  
 System.*out*.println("Title Name: " + title);  
 System.*out*.println("Author: " + author);  
 System.*out*.println("Publication Year: " + publicationYear);  
 System.*out*.println("Bio: " + bio);  
 }  
}  
  
public class LibraryManagement {  
  
 public static void main(String[] args) {  
  
 Author author = new Author("The Great Gatsby", 1925, "F. Scott Fitzgerald", "American novelist known for his depiction of the Jazz Age.");  
 author.displayInfo();  
 }  
}

**OUTPUT:**

Title Name: The Great Gatsby

Author: F. Scott Fitzgerald

Publication Year: 1925

Bio: American novelist known for his depiction of the Jazz Age.

**Sample Problem 2: Smart Home Devices**

**Description**: Create a hierarchy for a smart home system where Device is the superclass and Thermostat is a subclass.

* **Tasks**:
  + Define a superclass Device with attributes like deviceId and status.
  + Create a subclass Thermostat with additional attributes like temperatureSetting.
  + Implement a method displayStatus() to show each device's current settings.

**Goal**: Understand single inheritance by adding specific attributes to a subclass, keeping the superclass general

class Device {  
  
 public int deviceId;  
 public String status;  
  
 public Device(int deviceId, String status) {  
 this.deviceId = deviceId;  
 this.status = status;  
 }  
  
 public void displayInfo() {  
 System.*out*.println("Device Id: " + deviceId + "\nStatus: " + status);  
 }  
}  
  
class Thermostat extends Device {  
  
 private String temperatureSetting;  
  
 public Thermostat(int deviceId, String status, String temperatureSetting) {  
 super(deviceId, status);  
 this.temperatureSetting = temperatureSetting;  
 }  
  
 @Override  
  
 public void displayInfo() {  
 System.*out*.println("Device Id: " + deviceId + "\nStatus: " + status + "\nTemperature Setting: " + temperatureSetting);  
 }  
}  
  
public class SmartHomeDevices {  
  
 public static void main(String[] args) {  
  
 Device thermostat = new Thermostat(100, "Active", "Cool");  
 thermostat.displayInfo();  
 }  
}

**OUTPUT:**

Device Id: 100

Status: Active

Temperature Setting: Cool

**MULTILEVEL INHERITANCE**

**Sample Problem 1: Online Retail Order Management**

* **Description**: Create a multilevel hierarchy to manage orders, where Order is the base class, ShippedOrder is a subclass, and DeliveredOrder extends ShippedOrder.
* **Tasks**:
  + Define a base class Order with common attributes like orderId and orderDate.
  + Create a subclass ShippedOrder with additional attributes like trackingNumber.
  + Create another subclass DeliveredOrder extending ShippedOrder, adding a deliveryDate attribute.
  + Implement a method getOrderStatus() to return the current order status based on the class level.

**Goal**: Explore multilevel inheritance, showing how attributes and methods can be added across a chain of classes.

import java.util.Date;  
  
class Order {  
  
 public String orderId;  
 public String orderDate;  
  
 public Order (String orderId, String orderDate) {  
 this.orderId = orderId;  
 this.orderDate = orderDate;  
 }  
  
 public void getOrderStatus() {  
 System.*out*.println("Order Id: " + orderId + "\nOrder Date: " + orderDate);  
 }  
}  
  
class ShippedOrder extends Order {  
  
 public int trackingNumber;  
  
 public ShippedOrder(String orderId, String orderDate, int trackingNumber) {  
 super(orderId, orderDate);  
 this.trackingNumber = trackingNumber;  
 }  
  
 @Override  
  
 public void getOrderStatus() {  
 System.*out*.println("Order Id: " + orderId + "\nOrder Date: " + orderDate + "\nTracking Number: " + trackingNumber);  
 }  
}  
  
class Delivered extends ShippedOrder {  
  
 private String date;  
 public Delivered(String orderId, String orderDate, int trackingNumber, String date) {  
 super(orderId, orderDate, trackingNumber);  
 this.date = date;  
 }  
  
 @Override  
  
 public void getOrderStatus() {  
 System.*out*.println("Order Id: " + orderId + "\nOrder Date: " + orderDate + "\nTracking Number: " + trackingNumber + "\nDelivery Date: " + date);  
 }  
}  
  
public class OnlineRetailOrderManagement {  
  
 public static void main(String[] args) {  
  
 Order delivery = new Delivered("100", " 13 February, 2025", 301, "15 February, 2025");  
 delivery.getOrderStatus();  
 }  
}

**OUTPUT:**

Order Id: 100

Order Date: 13 February, 2025

Tracking Number: 301

Delivery Date: 15 February, 2025

**Sample Problem 2: Educational Course Hierarchy**

**Description**: Model a course system where Course is the base class, OnlineCourse is a subclass, and PaidOnlineCourse extends OnlineCourse.

* **Tasks**:
  + Define a superclass Course with attributes like courseName and duration.
  + Define OnlineCourse to add attributes such as platform and isRecorded.
  + Define PaidOnlineCourse to add fee and discount.

**Goal**: Demonstrate how each level of inheritance builds on the previous, adding complexity to the system

class Course {  
  
 public String courseName;  
 public int duration;  
  
 public Course(String courseName, int duration) {  
 this.courseName = courseName;  
 this.duration = duration;  
 }  
  
 public void displayDetails() {  
 System.*out*.println("Course Name: " + courseName + "\nDuration: " + duration);  
 }  
}  
  
class OnlineCourse extends Course{  
  
 public String platform;  
 public boolean isRecorded;  
  
  
 public OnlineCourse(String courseName, int duration, String platform, boolean isRecorded) {  
 super(courseName, duration);  
 this.platform = platform;  
 this.isRecorded = isRecorded;  
 }  
  
 @Override  
 public void displayDetails() {  
 System.*out*.println("Course Name: " + courseName + "\nDuration: " + duration + "\nPlatform: "  
 + platform + "\nIs Recorded? " + isRecorded);  
 }  
}  
  
class PaidOnlineCourse extends OnlineCourse {  
  
 private double fee;  
 private double discount;  
  
 public PaidOnlineCourse(String courseName, int duration, String platform, boolean isRecorded, double fee, double discount) {  
 super(courseName, duration, platform, isRecorded);  
 this.fee = fee;  
 this.discount = discount;  
 }  
  
 @Override  
 public void displayDetails() {  
 System.*out*.println("Course Name: " + courseName + "\nDuration: " + duration + "\nPlatform: "  
 + platform + "\nIs Recorded? " + isRecorded + "\nFee: " + fee + "\nDiscount: " + discount);  
 }  
}  
  
public class Education {  
  
 public static void main(String[] args) {  
  
 PaidOnlineCourse poc = new PaidOnlineCourse("Java Programming", 8, "Microsoft Teams", true, 25000, 10);  
 poc.displayDetails();  
 }  
}

**OUTPUT:**

Course Name: Java Programming

Duration: 8

Platform: Microsoft Teams

Is Recorded? true

Fee: 25000.0

Discount: 10.0

**HIERARCHICAL INHERITANCE**

**Sample Problem 1: Bank Account Types**

* **Description**: Model a banking system with different account types using hierarchical inheritance. BankAccount is the superclass, with SavingsAccount, CheckingAccount, and FixedDepositAccount as subclasses.
* **Tasks**:
  + Define a base class BankAccount with attributes like accountNumber and balance.
  + Define subclasses SavingsAccount, CheckingAccount, and FixedDepositAccount, each with unique attributes like interestRate for SavingsAccount and withdrawalLimit for CheckingAccount.
  + Implement a method displayAccountType() in each subclass to specify the account type.

**Goal**: Explore hierarchical inheritance, demonstrating how each subclass can have unique attributes while inheriting from a shared superclass.

class BankAccount {  
 public String accountNumber;  
 public double balance;  
  
 public BankAccount(String accountNumber, double balance) {  
 this.accountNumber = accountNumber;  
 this.balance = balance;  
 }  
  
 public void displayDetails() {  
 System.*out*.println("Account Number: " + accountNumber);  
 System.*out*.println("Balance: $" + balance);  
 }  
}  
  
class SavingsAccount extends BankAccount {  
  
 public double interestRate;  
  
 public SavingsAccount(String accountNumber, double balance, double interestRate) {  
 super(accountNumber, balance);  
 this.interestRate = interestRate;  
 }  
  
 public void displayAccountType() {  
 System.*out*.println("Account Type: Savings Account");  
 displayDetails();  
 System.*out*.println("Interest Rate: " + interestRate + "%");  
 }  
}  
  
class CheckingAccount extends BankAccount {  
 double withdrawalLimit;  
  
 public CheckingAccount(String accountNumber, double balance, double withdrawalLimit) {  
 super(accountNumber, balance);  
 this.withdrawalLimit = withdrawalLimit;  
 }  
  
 public void displayAccountType() {  
 System.*out*.println("Account Type: Checking Account");  
 displayDetails();  
 System.*out*.println("Withdrawal Limit: $" + withdrawalLimit);  
 }  
}  
  
class FixedDepositAccount extends BankAccount {  
 int durationMonths;  
  
 public FixedDepositAccount(String accountNumber, double balance, int durationMonths) {  
 super(accountNumber, balance);  
 this.durationMonths = durationMonths;  
 }  
  
 public void displayAccountType() {  
 System.*out*.println("Account Type: Fixed Deposit Account");  
 displayDetails();  
 System.*out*.println("Duration: " + durationMonths + " months");  
 }  
}  
  
public class BankSystem {  
 public static void main(String[] args) {  
 SavingsAccount savings = new SavingsAccount("SA001", 15000.0, 3.5);  
 CheckingAccount checking = new CheckingAccount("CA001", 8000.0, 1000.0);  
 FixedDepositAccount fd = new FixedDepositAccount("FD001", 50000.0, 12);  
  
 savings.displayAccountType();  
 System.*out*.println();  
  
 checking.displayAccountType();  
 System.*out*.println();  
  
 fd.displayAccountType();  
 }  
}

**OUTPUT:**

Account Type: Savings Account

Account Number: SA001

Balance: $15000.0

Interest Rate: 3.5%

Account Type: Checking Account

Account Number: CA001

Balance: $8000.0

Withdrawal Limit: $1000.0

Account Type: Fixed Deposit Account

Account Number: FD001

Balance: $50000.0

Duration: 12 months

**Sample Problem 2: School System with Different Roles**

**Description**: Create a hierarchy for a school system where Person is the superclass, and Teacher, Student, and Staff are subclasses.

* **Tasks**:
  + Define a superclass Person with common attributes like name and age.
  + Define subclasses Teacher, Student, and Staff with specific attributes (e.g., subject for Teacher and grade for Student).
  + Each subclass should have a method like displayRole() that describes the role.
* **Goal**: Demonstrate hierarchical inheritance by modeling different roles in a school, each with shared and unique characteristics.

class Person {  
 String name;  
 int age;  
  
 public Person(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void displayDetails() {  
 System.*out*.println("Name: " + name);  
 System.*out*.println("Age: " + age);  
 }  
}  
  
class Teacher extends Person {  
 String subject;  
  
 public Teacher(String name, int age, String subject) {  
 super(name, age);  
 this.subject = subject;  
 }  
  
 public void displayRole() {  
 System.*out*.println("Role: Teacher");  
 displayDetails();  
 System.*out*.println("Subject: " + subject);  
 }  
}  
  
class Student extends Person {  
 String grade;  
  
 public Student(String name, int age, String grade) {  
 super(name, age);  
 this.grade = grade;  
 }  
  
 public void displayRole() {  
 System.*out*.println("Role: Student");  
 displayDetails();  
 System.*out*.println("Grade: " + grade);  
 }  
}  
  
class Staff extends Person {  
 String department;  
  
 public Staff(String name, int age, String department) {  
 super(name, age);  
 this.department = department;  
 }  
  
 public void displayRole() {  
 System.*out*.println("Role: Staff");  
 displayDetails();  
 System.*out*.println("Department: " + department);  
 }  
}  
  
public class SchoolSystem {  
 public static void main(String[] args) {  
 Teacher teacher = new Teacher("Mr. Raj", 40, "Mathematics");  
 Student student = new Student("Anita", 16, "10th Grade");  
 Staff staff = new Staff("Joseph", 35, "Administration");  
  
 teacher.displayRole();  
 System.*out*.println();  
  
 student.displayRole();  
 System.*out*.println();  
  
 staff.displayRole();  
 }  
}

**OUTPUT:**

Role: Teacher

Name: Mr. Raj

Age: 40

Subject: Mathematics

Role: Student

Name: Anita

Age: 16

Grade: 10th Grade

Role: Staff

Name: Joseph

Age: 35

Department: Administration

**HYBRID INHERITANCE (SIMULATING MULTIPLE INHERITANCE)**

Since Java doesn’t support multiple inheritance directly, hybrid inheritance is typically achieved through **interfaces**.

**Sample Problem 1: Restaurant Management System with Hybrid Inheritance**

* **Description**: Model a restaurant system where Person is the superclass and Chef and Waiter are subclasses. Both Chef and Waiter should implement a Worker interface that requires a performDuties() method.
* **Tasks**:
  + Define a superclass Person with attributes like name and id.
  + Create an interface Worker with a method performDuties().
  + Define subclasses Chef and Waiter that inherit from Person and implement the Worker interface, each providing a unique implementation of performDuties().

**Goal**: Practice hybrid inheritance by combining inheritance and interfaces, giving multiple behaviors to the same objects

interface Worker {  
 void performDuties();  
}  
  
class Persons {  
 String name;  
 int id;  
  
 public Persons(String name, int id) {  
 this.name = name;  
 this.id = id;  
 }  
  
 public void showInfo() {  
 System.*out*.println("Name: " + name);  
 System.*out*.println("ID: " + id);  
 }  
}  
  
class Chef extends Persons implements Worker {  
 public Chef(String name, int id) {  
 super(name, id);  
 }  
  
 public void performDuties() {  
 showInfo();  
 System.*out*.println("Duties: Preparing meals and managing the kitchen.");  
 }  
}  
  
class Waiter extends Persons implements Worker {  
 public Waiter(String name, int id) {  
 super(name, id);  
 }  
  
 public void performDuties() {  
 showInfo();  
 System.*out*.println("Duties: Serving customers and taking orders.");  
 }  
}  
  
public class RestaurantSystem {  
 public static void main(String[] args) {  
 Chef chef = new Chef("Gordon", 101);  
 Waiter waiter = new Waiter("Alice", 202);  
  
 chef.performDuties();  
 System.*out*.println();  
  
 waiter.performDuties();  
 }  
}

**OUTPUT:**

Name: Gordon

ID: 101

Duties: Preparing meals and managing the kitchen.

Name: Alice

ID: 202

Duties: Serving customers and taking orders.

**Sample Problem 2: Vehicle Management System with Hybrid Inheritance**

**Description**: Model a vehicle system where Vehicle is the superclass and ElectricVehicle and PetrolVehicle are subclasses. Additionally, create a Refuelable interface implemented by PetrolVehicle.

* **Tasks**:
  + Define a superclass Vehicle with attributes like maxSpeed and model.
  + Create an interface Refuelable with a method refuel().
  + Define subclasses ElectricVehicle and PetrolVehicle. PetrolVehicle should implement Refuelable, while ElectricVehicle include a charge() method.
* **Goal**: Use hybrid inheritance by having PetrolVehicle implement both Vehicle and Refuelable, demonstrating how Java interfaces allow adding multiple behaviors.

interface Refuelable {  
 void Refuel ();  
}  
  
class Vehicles {  
  
 public int maxSpeed;  
 public String model;  
  
 public Vehicles (String model, int maxSpeed) {  
 this.model = model;  
 this.maxSpeed = maxSpeed;  
 }  
  
 public void displayVehicle() {  
 System.*out*.println("Model: " + model + "\nMax Speed: " + maxSpeed + " Km/h");  
 }  
}  
  
class ElectricVehicle extends Vehicles {  
  
  
 public ElectricVehicle(String model, int maxSpeed) {  
 super(model, maxSpeed);  
 }  
  
 public void Charge() {  
 displayVehicle();  
 System.*out*.println("Charging the electric vehicle.");  
 }  
}  
  
class PetrolVehicle extends Vehicles implements Refuelable{  
  
 public PetrolVehicle(String model, int maxSpeed) {  
 super(model, maxSpeed);  
 }  
  
 public void Refuel() {  
 displayVehicle();  
 System.*out*.println("Refueling the petrol vehicle.");  
 }  
}  
  
public class VehicleSystem {  
  
 public static void main(String[] args) {  
 ElectricVehicle ev = new ElectricVehicle("BMW X6", 250);  
 PetrolVehicle pv = new PetrolVehicle("Volkswagen Virtus", 220);  
 ev.displayVehicle();  
 System.*out*.println();  
 pv.displayVehicle();  
 }  
}

**OUTPUT:**

Model: BMW X6

Max Speed: 250 Km/h

Model: Volkswagen Virtus

Max Speed: 220 Km/h