Unit 2 Inheritance

Inheritance:
Working with Inheritance: Inheritance Basics & Types, using super, Method Overriding, Dynamic method dispatch, final

Inheritance with Constructor Overloading (Employee & Manager)
Problem Statement
Create a Employee class with name and salary fields. Extend it with a Manager class that adds a bonus field. Ensure **constructor chaining**.

```
class Employee {
    String name;
    double salary;

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Method to display details
    public void displayDetails() {
        System.out.println("Employee Name: " + name);
        System.out.println("Salary: $" + salary);
    }
}

// Manager extends Employee
class Manager extends Employee {
    double bonus;

    // Constructor chaining
    public Manager(String name, double salary, double bonus) {
        super(name, salary); // Calls Employee constructor
        this.bonus = bonus;
    }

    // Overriding method
    @Override
    public void displayDetails() {
        super.displayDetails(); // Calls Employee's method
        System.out.println("Bonus: $" + bonus);
        System.out.println("Total Salary: $" + (salary + bonus));
    }
}

public class Main {
    public static void main(String[] args) {
```

```
        Manager mgr = new Manager("Alice", 50000, 10000);
        mgr.displayDetails();
    }
}
```

Output

```
Employee Name: Alice
Salary: $50000.0
Bonus: $10000.0
Total Salary: $60000.0
```

If data is provate in base class

```
class Employee {
    private String name;  // Changed to private
    private double salary; // Changed to private

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Getter methods to access private fields
    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    // Method to display details
    public void displayDetails() {
        System.out.println("Employee Name: " + name);
        System.out.println("Salary: $" + salary);
    }
}

// Manager extends Employee
class Manager extends Employee {
    private double bonus;

    // Constructor chaining
    public Manager(String name, double salary, double bonus) {
        super(name, salary);
        this.bonus = bonus;
    }
```

```
    // Overriding method
    @Override
    public void displayDetails() {
        super.displayDetails(); // Calls Employee's method
        System.out.println("Bonus: $" + bonus);
        // Use getter method to access salary
        System.out.println("Total Salary: $" + (getSalary() + bonus));
    }
}

public class Main {
    public static void main(String[] args) {
        Manager mgr = new Manager("Alice", 50000, 10000);
        mgr.displayDetails();
    }
}
```

Method Overriding & Super Keyword (Vehicle & Car)

Problem Statement

Create a Vehicle class with a speed property. Extend it with a Car class that has a specific gear system. Override move() in Car.

```
class Vehicle {
    int speed;

    public Vehicle(int speed) {
        this.speed = speed;
    }

    public void move() {
        System.out.println("Vehicle is moving at " + speed + " km/h.");
    }
}

// Car extends Vehicle
class Car extends Vehicle {
    int gears;

    public Car(int speed, int gears) {
        super(speed);
        this.gears = gears;
    }

    // Overriding move method
    @Override
    public void move() {
        super.move();
        System.out.println("Car is using " + gears + " gears.");
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car(120, 6);
        myCar.move();
    }
}
```

Output

```
Vehicle is moving at 120 km/h.
Car is using 6 gears.
```

```
class Vehicle {
    private int speed; // Changed to private

    public Vehicle(int speed) {
        this.speed = speed;
    }

    // Getter method to access speed
    public int getSpeed() {
        return speed;
    }

    public void move() {
        System.out.println("Vehicle is moving at " + speed + " km/h.");
    }
}

// Car extends Vehicle
class Car extends Vehicle {
    private int gears;

    public Car(int speed, int gears) {
        super(speed);
        this.gears = gears;
    }

    // Overriding move method
    @Override
    public void move() {
        // Use getSpeed() to access private speed
        System.out.println("Vehicle is moving at " + getSpeed() + " km/h.");
        System.out.println("Car is using " + gears + " gears.");
    }
}
```

```
public class Main {
   public static void main(String[] args) {
      Car myCar = new Car(120, 6);
      myCar.move();
   }
}
```

Multilevel Inheritance (Person → Employee → Developer)
Problem Statement
Create a Person class with a name. Extend it with an Employee class that has an employeeID. Further extend it with a Developer class that has a programming language.

```
class Person {
   String name;

   public Person(String name) {
      this.name = name;
   }

   public void showInfo() {
      System.out.println("Name: " + name);
   }
}

// Employee extends Person
class Employee extends Person {
   int employeeID;

   public Employee(String name, int employeeID) {
      super(name);
      this.employeeID = employeeID;
   }

   @Override
   public void showInfo() {
      super.showInfo();
      System.out.println("Employee ID: " + employeeID);
   }
}

// Developer extends Employee
class Developer extends Employee {
   String programmingLanguage;

   public Developer(String name, int employeeID, String programmingLanguage) {
      super(name, employeeID);
      this.programmingLanguage = programmingLanguage;
   }
```

```
    @Override
    public void showInfo() {
        super.showInfo();
        System.out.println("Programming Language: " + programmingLanguage);
    }
}

public class Main {
    public static void main(String[] args) {
        Developer dev = new Developer("Charlie", 101, "Java");
        dev.showInfo();
    }
}
```

Output

```
Name: Charlie
Employee ID: 101
Programming Language: Java
```

```
Data members as private
class Person {
    private String name; // Changed to private

    public Person(String name) {
        this.name = name;
    }

    // Getter for name
    public String getName() {
        return name;
    }

    public void showInfo() {
        System.out.println("Name: " + name);
    }
}

// Employee extends Person
class Employee extends Person {
    private int employeeID; // Changed to private

    public Employee(String name, int employeeID) {
        super(name);
        this.employeeID = employeeID;
    }

    // Getter for employeeID
```

```
    public int getEmployeeID() {
        return employeeID;
    }

    @Override
    public void showInfo() {
        super.showInfo();
        System.out.println("Employee ID: " + employeeID);
    }
}

// Developer extends Employee
class Developer extends Employee {
    private String programmingLanguage; // Changed to private

    public Developer(String name, int employeeID, String programmingLanguage) {
        super(name, employeeID);
        this.programmingLanguage = programmingLanguage;
    }

    // Getter for programmingLanguage
    public String getProgrammingLanguage() {
        return programmingLanguage;
    }

    @Override
    public void showInfo() {
        super.showInfo();
        System.out.println("Programming Language: " + programmingLanguage);
    }
}

public class Main {
    public static void main(String[] args) {
        Developer dev = new Developer("Charlie", 101, "Java");
        dev.showInfo();
    }
}
```

Mcq

method overloading

What will be the output of the following Java program?

```
class Test {
   void display(int a, double b) {
      System.out.println("int-double");
   }

   void display(double a, int b) {
      System.out.println("double-int");
   }

   public static void main(String[] args) {
      Test obj = new Test();
      obj.display(10, 10); // What will happen here?
   }
}
```

Ans Ambiguous method call

Q What will be the output of the following Java code?

```
class Test {
   void show(int a) {
      System.out.println("int");
   }

   void show(double a) {
      System.out.println("double");
   }


   public static void main(String[] args) {
      Test obj = new Test();
      obj.show(5.5f); // What will be printed?
   }
}
```

Ans: double

Q **What will happen if you try to overload a method by only changing the return type?**

```
class Example {
   int sum(int a, int b) {
      return a + b;
   }

   double sum(int a, int b) { // Is this valid?
      return a + b;
```

```
    }

    public static void main(String[] args) {
        Example obj = new Example();
        System.out.println(obj.sum(5, 10));
    }
}
```
Ans: compilation error

Q
```
class Test {
    public void greet(String name, String... messages) {
        if (messages.length == 0) {
            System.out.println(name + ": Hello!"); // Default message
        } else {
            System.out.println(name + ": " + messages[0]);
        }
    }

    public static void main(String[] args) {
        Test obj = new Test();
        obj.greet("Alice"); // Uses default message
        obj.greet("Bob", "Good Morning!"); // Uses provided message
    }
}
```

```
class Test {
    public static void printNumbers(int... nums) {
        System.out.println("Total numbers: " + nums.length);
    }

    public static void main(String[] args) {
        printNumbers(1, 2, 3, 4, 5);
        printNumbers();
    }
}
```

Q **What will be the output of the following program?**
```
class Parent {
    void print(int a) {
        System.out.println("Parent int");
    }
}
```

```
class Child extends Parent {
   void print(int a, int b) {
      System.out.println("Child int-int");
   }
}

public class Main {
   public static void main(String[] args) {
      Parent obj = new Child();
      obj.print(10); // Which method will be called?
   }
}
```
Ans: Parent int

Q **What will be the output of the following program?**

```
class A {
   public int value = 10;

   public int getValue() {
      return value;
   }
}

class B extends A {
   public int value = 20;

   @Override
   public int getValue() {
      return value;
   }
}

public class InheritanceTest {
   public static void main(String[] args) {
      A a = new B();
      System.out.println(a.value + " " + a.getValue());
   }
}
```

Q **What will be the output of the following program?**

```
class Base {
   public void display() {
      System.out.println("Base display");
   }

   public void show() {
```

```
        System.out.println("Base show");
        display();
    }
}

class Derived extends Base {
    @Override
    public void display() {
        System.out.println("Derived display");
    }
}

public class InheritanceTest {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show();
    }
}
```

Final keyword

Q **What will be the output of the following program?**

```
class Parent {
    final void show() { // Final method (Cannot be overridden)
        System.out.println("This is a final method in Parent.");
    }
}

class Child extends Parent {
    // Attempting to override 'show()' will cause a compilation error
    /*
    void show() {
        System.out.println("Trying to override final method.");
    }
    */
}

public class Main {
    public static void main(String[] args) {
        Parent obj = new Parent();
        obj.show(); // Calls final method from Parent
    }
}
```

Q **What will be the output of the following program?**

```
class Story {
   void recite(int chapter) {}
}
class Adventure extends Story {
   final void recite(final int chapter) { // g1
      switch(chapter) { // g2
         case 2: System.out.print(9);
         default: System.out.print(3);
      }
   }
public static void main(String ... u) {
   var bedtime = new Adventure();
   bedtime.recite(2);
   }
}
```

Q

```
class Laptop extends Computer {
   public void startup() {
      System.out.print("laptop-");
   }
}

class Computer {
   public void startup() {
      System.out.print("computer-");
   }
```

```
public static void main(String[] args) {
   Computer computer = new Laptop();
   Laptop laptop = new Laptop();
   computer.startup();
   laptop.startup();
   }
}
```

Q **What will be the output of the following program?**

```
enum DaysOff {
Thanksgiving, PresidentsDay, ValentinesDay
}
class Vacation {
   public static void main(String[] unused) {
      final DaysOff input = DaysOff.Thanksgiving;
      switch(input) {
      default:
      case DaysOff.ValentinesDay:
         System.out.print("1");
      case DaysOff.PresidentsDay:
         System.out.print("2");
   }
   }
}
```

Ans: 12