Java Inheritance - Deep Dive with Examples

What is Inheritance?

Inheritance is one of the four pillars of Object-Oriented Programming (OOP) in Java. It allows one class (child/subclass) to inherit fields and methods from another class (parent/superclass). This helps in **code reuse, maintainability, and hierarchical classification**.

Types of Inheritance in Java

1. **Single Inheritance** – One class inherits from another.
2. **Multilevel Inheritance** – A class inherits from another class, which itself is inherited from another.
3. **Hierarchical Inheritance** – Multiple classes inherit from a single parent.
4. **Multiple Inheritance (via Interfaces)** – A class implements multiple interfaces since Java does not support multiple inheritance via classes.

---

1. Single Inheritance

📌 *A child class inherits properties from a single parent class.*

```
// Parent class
class Animal {
    String name = "Generic Animal";

    void makeSound() {
        System.out.println("Some generic animal sound");
    }
}

// Child class inheriting from Animal
class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks: Woof Woof!");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        System.out.println(dog.name); // Inherited property
        dog.makeSound(); // Inherited method
        dog.bark(); // Own method
    }
}
```

Output:

```
Generic Animal
Some generic animal sound
Dog barks: Woof Woof!
```

---

2. Multilevel Inheritance

📌 *A class inherits from another class, which in turn inherits from another class.*

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Mammal extends Animal {
    void walk() {
        System.out.println("Mammal is walking");
    }
}

class Dog extends Mammal {
    void bark() {
        System.out.println("Dog barks!");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // Inherited from Animal
        dog.walk(); // Inherited from Mammal
        dog.bark(); // Defined in Dog
    }
}
```

Output:

```
Animal is eating
Mammal is walking
Dog barks!
```

---

3. Hierarchical Inheritance

📌 *Multiple child classes inherit from the same parent class.*

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    void meow() {
        System.out.println("Cat meows");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.bark();

        Cat cat = new Cat();
        cat.makeSound();
        cat.meow();
    }
}
```

Output:

```
Animal makes a sound
Dog barks
Animal makes a sound
Cat meows
```

---

4. Multiple Inheritance (via Interfaces)

📌 *Java does not support multiple inheritance using classes but allows it using interfaces.*

```
interface Animal {
    void eat();
}

interface Sound {
    void makeSound();
}

// Dog class implements multiple interfaces
class Dog implements Animal, Sound {
    public void eat() {
        System.out.println("Dog is eating");
    }

    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.makeSound();
    }
}
```

Output:

CopyEdit

```
Dog is eating
Dog barks
```

`super` Keyword in Inheritance

📌 The `super` keyword is used to access **parent class methods and constructors**.

```
class Animal {
    String name = "Animal";

    void display() {
        System.out.println("This is an animal");
    }
}

class Dog extends Animal {
    String name = "Dog";
    void display() {
        System.out.println("This is a dog");
        System.out.println("Parent name: " + super.name); // Access
parent class property
        super.display(); // Call parent method
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.display();
    }
}
```

Output:

```
This is a dog
Parent name: Animal
This is an animal
```

Method Overriding in Inheritance

📌 *A child class can provide a specific implementation of a method defined in the parent class.*

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Dog();
        myAnimal.makeSound(); // Calls overridden method in Dog
    }
}
```

Output:

```
Dog barks
```

Abstract Class & Inheritance

📌 *An abstract class can have both abstract and concrete methods. Child classes must implement abstract methods.*

```
abstract class Animal {
    abstract void makeSound(); // Abstract method

    void sleep() {
        System.out.println("Animal is sleeping");
    }
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.sleep();
    }
}
```

Output:

```
Dog barks
Animal is sleeping
```

Final Keyword in Inheritance

📌 *`final` prevents a class from being inherited and methods from being overridden.*

```
final class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

// This will cause an error
// class Dog extends Animal { }

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.makeSound();
    }
}
```

Error:

```
Cannot inherit from final class 'Animal'
```

For **final methods**, they can't be overridden:

```
class Animal {
    final void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    // Error: Cannot override the final method
    // void makeSound() { System.out.println("Dog barks"); }
}
```

Conclusion

✅ **Inheritance** helps in **code reusability and hierarchy creation**.
✅ Java supports **single, multilevel, hierarchical, and multiple inheritance (via interfaces)**.
✅ **Method Overriding** allows specific implementations.
✅ **`super` keyword** accesses parent class members.
✅ **Abstract classes** provide blueprints for subclasses.
✅ **Final classes/methods** prevent modification.