Inheritance

📚 Complete Study Material on Inheritance in Java

1️⃣ Introduction to Inheritance

🤔 What is Inheritance?

Inheritance is a mechanism in Java that allows a 🏛️ **class** to acquire the **properties** 🏗️ and **behaviors (methods)** ⚙️ of another class. It promotes 🔄 **code reusability** and establishes a **parent-child** 👨‍👦 relationship between classes.

🔷 Why Use Inheritance?

1️⃣ ♻️ **Code Reusability** – Write once, use multiple times.
2️⃣
📏 **Extensibility** – Allows adding new features without modifying existing code.
3️⃣
🎭 **Method Overriding** – Enables runtime polymorphism.
4️⃣
📂 **Better Organization** – Helps in structuring code with hierarchical relationships.

2️⃣ Basic Syntax of Inheritance

In Java, we use the `extends` keyword to implement inheritance.

🖋️ Syntax:

```
class ParentClass {
    // Parent properties and methods 🏗️
}

class ChildClass extends ParentClass {
    // Additional properties and methods 🏗️
}
```

✅ Example:

```
class Animal {
    void makeSound() {
        System.out.println("Animals make sounds");
    }
}

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

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // 🏛️ Inherited method
        myDog.bark();      // 🆕 Child class method
    }
}
```

🖥️ Output:

```
Animals make sounds
Dog barks
```

3️⃣ Types of Inheritance in Java

1️⃣ Single Inheritance 🔗

One class **inherits** from another class.

```
class Parent {
    void show() {
        System.out.println("Parent class");
    }
}

class Child extends Parent {
    void display() {
        System.out.println("Child class");
    }
}
```

2️⃣ Multilevel Inheritance 🏛️➡️🏛️➡️🏛️

A child class **inherits** from another child class.

```
class GrandParent {
    void grandparentMethod() {
        System.out.println("Grandparent method");
    }
}

class Parent extends GrandParent {
    void parentMethod() {
        System.out.println("Parent method");
    }
}

class Child extends Parent {
    void childMethod() {
        System.out.println("Child method");
    }
}
```

3️⃣ Hierarchical Inheritance 🌳

One **parent** 🏛️ is inherited by multiple **child classes** 👶.

```
class Parent {
    void parentMethod() {
        System.out.println("Parent method");
    }
}

class Child1 extends Parent {
    void child1Method() {
        System.out.println("Child 1 method");
    }
}

class Child2 extends Parent {
    void child2Method() {
        System.out.println("Child 2 method");
    }
}
```

4️⃣ Hybrid Inheritance (❌ Not Supported in Java)

Java does **not** support hybrid inheritance **directly** due to ambiguity issues. Instead, Java provides **interfaces** to handle such cases. 🎭

4️⃣ Method Overriding 🔄

When a **child class** provides a **specific implementation** of a method that is **already defined** in its parent class. 🏛️➡️🎭

✅ Rules of Overriding:

1️⃣ The **method name** and **parameters** must be **the same**.
2️⃣ The
access modifier cannot be **more restrictive**.

3️⃣ The
return type must be **same or a subclass** (covariant return type).
4️⃣
`static` **methods cannot be overridden**.
5️⃣ The overriding method
cannot throw a broader exception than the parent method.

✅ Example:

```
class Parent {
    void show() {
        System.out.println("Parent show method");
    }
}

class Child extends Parent {
    @Override
    void show() {
        System.out.println("Child show method");
    }
}
```

🖥️ Output:

```
Child show method
```

5️⃣ The 🏛️ `super` Keyword

The `super` keyword is used to:

1️⃣ Call the **parent class constructor**.
2️⃣ Access
parent class methods.
3️⃣ Access
parent class variables.

✅ Example:

```
class Parent {
    Parent() {
        System.out.println("Parent constructor");
    }
}

class Child extends Parent {
    Child() {
        super(); // Calls Parent constructor
        System.out.println("Child constructor");
    }
}
```

🖥️ Output:

```
Parent constructor
Child constructor
```

6️⃣ Upcasting & Downcasting 🔄

🆙 Upcasting (Implicit Type Conversion)

```
Parent p = new Child();
```

🔽 Downcasting (Explicit Type Conversion)

```
Child c = (Child) p; // Explicit casting required
```

✅ Example:

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

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

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog(); // Upcasting
        a.sound();

        Dog d = (Dog) a; // Downcasting
        d.bark();
    }
}
```

7️⃣ Composition vs Inheritance 🤝

🛠️ When to Use Composition Instead of Inheritance?

- If the relationship is **"has-a"** instead of **"is-a"**, use **composition**.
- Composition provides **more flexibility** and avoids **tight coupling**.

✅ Example of Composition:

```
class Engine {
    void start() {
        System.out.println("Engine starting...");
    }
}

class Car {
    private Engine engine = new Engine(); // Composition
    void startCar() {
        engine.start();
    }
}
```

🎯 Key Takeaways

✅ **Inheritance promotes ♻️ code reusability.**
✅
Java supports single, multilevel, and hierarchical inheritance.
✅
Method overriding enables 🎭 runtime polymorphism.
✅
The `super` keyword helps access parent class members 🏛️.
✅
Use composition over inheritance when appropriate.