INHERITANCE IN JAVA (In-Depth Complete Document)

---

1. Introduction to Inheritance

Inheritance is one of the core pillars of Object-Oriented Programming (OOP).

Definition

Inheritance is the process by which **one class acquires the properties (variables) and behaviors (methods) of another class**.

Why Inheritance?

- ✔ Code reusability
- ✔ Avoids redundancy
- ✔ Promotes hierarchical classification
- ✔ Enables runtime polymorphism

Memory Tip: Inheritance = Reuse + Relationship + Hierarchy

---

2. Terminology Used in Inheritance

Term	Meaning
Parent Class	Super class / Base class
Child Class	Sub class / Derived class

- A **parent class** provides properties and behaviors
- A **child class** acquires and can reuse or modify them

3. Basic Syntax of Inheritance

```
class Parent {
    int age;

    void display() {
        age = 10;
        System.out.println("Age is: " + age);
    }
}

class Child extends Parent {
    // inherits age and display()
}

public class LaunchInh1 {
    public static void main(String[] args) {
        Parent p = new Parent();
        p.display();

        Child c = new Child();
        c.display();
    }
}
```

---

4. `extends` Keyword

- Inheritance in Java is achieved using the **extends** keyword
- A class can extend **only one class**

```
class Child extends Parent
```

---

5. IS-A Relationship

Inheritance represents an **IS-A** relationship.

Examples:

- Dog IS-A Animal
- Student IS-A Person
- Child IS-A Parent

Java supports IS-A relationships via inheritance.

6. One Parent – Multiple Children (Allowed)

```
class Parent { }

class Child1 extends Parent { }
class Child2 extends Parent { }
```

✔ Allowed in Java

---

7. Multiple Inheritance (Not Allowed)

```
class A { }
class B { }
class C extends A, B { } // □ ERROR
```

Why Java does not allow it?

- Ambiguity problem
- Diamond-shaped problem

Memory Tip: One child can have only one father

---

8. Multilevel Inheritance (Allowed)

```
class Aeroplane { }
class CargoPlane extends Aeroplane { }
class FighterPlane extends CargoPlane { }
```

✔ Child can act as a parent

---

9. Cyclic Inheritance (Not Allowed)

```
class A extends B { }
class B extends A { } // □
```

Java strictly prohibits cyclic inheritance.

10. Object Class – Root of All Classes

Every class in Java implicitly extends the `Object` class.

```
class Test { }
```

Is treated as:

```
class Test extends Object { }
```

---

11. Private Members and Inheritance

Private members **do not get inherited**.

```
class Parent {
    private int x = 10;
}

class Child extends Parent {
    void show() {
        // x □ not accessible
    }
}
```

This ensures **encapsulation**.

---

12. Constructors and Inheritance

Important Rules

- Constructors are **not inherited**
- Constructors **do execute** during child object creation

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

class Child extends Parent {
    Child() {
        System.out.println("Child Constructor");
    }
}

public class Test {
    public static void main(String[] args) {
        new Child();
    }
}
```**Execution Order:** Parent → Child

13. Types of Methods in Inheritance

1. Inherited Methods

Methods used as-is from parent class.

```
class Parent {
    void height() {
        System.out.println("Height inherited");
    }
}
```

---

2. Overridden Methods

Methods modified in child class.

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

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

---

3. Specialized (Child-Specific) Methods

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

---

14. Upcasting and Downcasting

Upcasting (Automatic & Safe)

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

Downcasting (Explicit)

```
Child c = (Child) p;
```

 Incorrect downcasting leads to `ClassCastException`.

15. Access Specifier Visibility Table

Modifier	Same Class	Same Package	Subclass	Outside
private	✔	✘	✘	✘
default	✔	✔	✘	✘
protected	✔	✔	✔	✘
public	✔	✔	✔	✔

---

16. `this()` and `super()`

- `this()` → calls current class constructor
- `super()` → calls parent class constructor

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

class Child extends Parent {
    Child() {
        super(10);
        System.out.println("Child constructor");
    }
}
```

---

17. Uses of `this` and `super`

Keyword	Purpose
this	Refers current object
this()	Calls current class constructor
super	Refers parent members
super()	Calls parent constructor

---

18. Method Overriding Rules

Visibility Rule

Child method **cannot reduce visibility**.

```
protected void show() {}
public void show() {} // ✔
```

Return Type Rules

- Primitive return types → must be same
- Object return types → covariant allowed

```
class Parent {
    Object get() { return null; }
}

class Child extends Parent {
    String get() { return "Hello"; }
}
```

---

19. Method Overloading vs Overriding

Feature	Overloading	Overriding
Parameters	Different	Same
Return Type	Same / Different	Same / Covariant
Occurs In	Same class	Parent–Child
Polymorphism	Compile-time	Runtime

---

20. Can Constructors Be Overridden?

 No. Constructors are not inherited.

---

21. Static Methods and Inheritance

Static methods are inherited but **cannot be overridden**.

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

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

This is called **method hiding**, not overriding.

22. Final Summary

- Inheritance represents IS-A relationship
- Java supports single & multilevel inheritance
- Multiple and cyclic inheritance are not allowed
- Constructors execute but are not inherited
- Private members are not inherited
- Static methods are hidden, not overridden
- Parent reference + Child object enables runtime polymorphism