Inheritance in Java

Inheritance is the mechanism by which one class **acquires** the properties (i.e., variables and methods) of another class.

- **Properties** = Variables + Methods
- The class whose properties are inherited is called the **Parent / Super / Base** class.
- The class that inherits the properties is called the **Child / Sub / Derived** class.
- The keyword `extends` is used by the **child** class to specify its **parent** class.
- A **parent** class reference can hold an object of its **child** class, but the reverse is not allowed:

```
ParentClass pc = new ChildClass(); // Valid
ChildClass cc = new ParentClass(); // Invalid
```

The Object Class

- `Object` is the **root** superclass in Java.
- Every class in Java implicitly extends `Object` if no other superclass is specified.
- A class that does **not** explicitly extend another class is called a **Direct Child** of `Object`.
- A class that **extends** another class (except `Object`) is called an **Indirect Child** of `Object`.

Types of Inheritance

Java supports the following **five types** of inheritance:

1. **Single Inheritance**
2. **Multilevel Inheritance**
3. **Multiple Inheritance** (Not applicable to classes in Java)
4. **Hierarchical Inheritance**

5. **Hybrid Inheritance**

Single Inheritance

- One child class inherits from one parent class.

```java
class ParentClass {
    int var1 = 12;
    public void func1() {
        System.out.println("In Parent class");
    }
}

class ChildClass extends ParentClass {
    int var2 = 11;
    public void func2() {
        System.out.println("In Child class");
    }

    public static void main(String[] args) {
        ParentClass pc = new ParentClass();
        ParentClass pc1 = new ChildClass();
        ChildClass cc = new ChildClass();

        System.out.println(pc.var1);
        pc.func1();

        System.out.println(cc.var1);
        cc.func1();
        System.out.println(cc.var2);
        cc.func2();

        System.out.println(pc1.var1);
        pc1.func1();
    }
}
```

- A child class object can be referenced by a parent class variable (polymorphism).
- The reverse (assigning a parent object to a child reference) is **not allowed**.

Multilevel Inheritance

- A class inherits from a child class which in turn inherits from another class, forming a chain.

```java
class A {
    public static void func1() {
        System.out.println("In Class A.");
    }
}

class B extends A {
    public static void func2() {
        System.out.println("In Class B.");
    }
}

class C extends B {
    public static void func3() {
        System.out.println("In Class C.");
    }
}

class MultiLevelInheritance {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();

        a.func1();
        b.func1();
        b.func2();
        c.func1();
        c.func2();
        c.func3();
    }
}
```

Hierarchical Inheritance

- One parent class has multiple child classes.

```java
class A {
    public static void func1() {
        System.out.println("In Class A.");
    }
}

class B extends A {
    public static void func2() {
        System.out.println("In Class B.");
    }
}

class C extends A {
    public static void func3() {
        System.out.println("In Class C.");
    }
}

class HierarchicalInheritance {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();

        a.func1();
        b.func1();
        b.func2();
        c.func1();
        c.func3();
    }
}
```

Multiple Inheritance

- A class inherits from **multiple parent classes**.
- **Not supported in Java** through classes due to the **ambiguity problem**.
- Supported via **interfaces**.

```java
class A {
    public static void func() {
        System.out.println("In Class A.");
    }
}

class B extends A {
    public static void func() {
        System.out.println("In Class B.");
    }
}

// Invalid in Java:
// class MultipleInheritance extends A, B { ... }
```

> If multiple parent classes define the same method, the JVM cannot determine which one to use — this is the **diamond problem**.

Hybrid Inheritance

- A combination of more than one type of inheritance (excluding multiple via classes).
- Possible in Java through **interfaces**.

```java
interface A {
    default void funcA() {
        System.out.println("Inside Interface A");
    }
}

interface B {
    default void funcB() {
        System.out.println("Inside Interface B");
    }
}

// Hierarchical + Multiple (via interfaces) = Hybrid
class C implements A, B {
    public void funcC() {
        System.out.println("Inside Class C");
    }
}
```

```
19
20 class HybridInheritanceDemo {
21     public static void main(String[] args) {
22         C obj = new C();
23         obj.funcA();  // from Interface A
24         obj.funcB();  // from Interface B
25         obj.funcC();  // from Class C
26     }
27 }
```