In Java, **class** and **interface** are two core building blocks of object-oriented programming, but they have **different roles and rules**.

◇ Class in Java

A **class** is a blueprint for creating objects. It contains:

- Fields (variables)
- Methods (functions)
- Constructors
- Can have both **defined logic** (method body) and **data**

◇ Example:

```
class Car {
   void drive() {
      System.out.println("Car is driving");
   }
}
```

◇ Interface in Java

An **interface** is a reference type, like a class, but it only contains:

- **Abstract methods** (before Java 8)
- From Java 8 onwards: **default**, **static**, and **private methods**
- **No constructors**
- Used to define a **contract** that implementing classes must follow

◇ Example:

```
interface Vehicle {
   void drive();  // abstract method
}
```

◇ Implementing an Interface:

```
class Car implements Vehicle {
```

```
    public void drive() {
        System.out.println("Car is driving");
    }
}
```

🔁 **Key Differences Between Class and Interface:**

Feature	Class	Interface
Keyword	class	interface
Inheritance	Supports **single** inheritance only	A class can implement **multiple interfaces**
Method Implementation	Can have full method bodies	Only method signatures (until Java 7); from Java 8, can have default and static methods
Constructors	Can have constructors	Cannot have constructors
Variables	Regular variables	Only public static final (constants)
Usage	To define real-world entities	To define capability or contract (e.g., Flyable, Readable)
Instantiation	Can create objects using new	Cannot be instantiated directly

---

💡 **Quick Analogy:**

Term	**Think of it as...**
Class	A real-world **object** blueprint (like "Car")
Interface	A **behavior** promise (like "Drivable")

What is Inheritance in Java?

Inheritance is one of the fundamental concepts of **Object-Oriented Programming (OOP)** in Java. It allows a class (child/subclass) to **inherit** properties (fields) and behaviors (methods) from another class (parent/superclass).

🔸 **Why Use Inheritance?**

- **Code Reusability**: Write once, use many times.
- **Method Overriding**: Customize behavior of parent class.
- **Code Maintenance**: Easier updates and management.

Syntax of Inheritance in Java

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

class Child extends Parent {
    // inherits properties and methods of Parent
}
```

☑ Types of Inheritance in Java

Type	Supported in Java?	Description
1. Single Inheritance	✅ Yes	One child inherits from one parent
2. Multilevel Inheritance	✅ Yes	Class inherits from a class which itself inherits from another
3. Hierarchical Inheritance	✅ Yes	Multiple classes inherit from a single parent
4. Multiple Inheritance (via interfaces)	✅ Yes (only through interfaces)	A class implements multiple interfaces
5. Hybrid Inheritance	🚫 Not directly (Only via interfaces)	Combination of two or more types

1. Single Inheritance

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

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

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // inherited
        d.bark(); // own method
    }
}
```

Output:

This animal eats food.

Dog barks.

2. Multilevel Inheritance

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

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

class Puppy extends Dog {
    void weep() {
        System.out.println("Weeping...");
    }
}

public class Main {
    public static void main(String[] args) {
        Puppy p = new Puppy();
        p.eat();   // from Animal
        p.bark();  // from Dog
        p.weep();  // from Puppy
    }
}
```

Output:

Eating...

Barking...

Weeping...

3. Hierarchical Inheritance

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

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

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

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.bark();

        Cat c = new Cat();
        c.eat();
        c.meow();
    }
}
```

Output:

Eating...

Barking...

Eating...

Meowing...

4. Multiple Inheritance (via Interface)

Java **does not support** multiple inheritance using classes, but **does support** it through **interfaces**.

```
interface Printable {
    void print();
}

interface Showable {
    void show();
}

class A implements Printable, Showable {
    public void print() {
        System.out.println("Printing...");
    }

    public void show() {
        System.out.println("Showing...");
    }
}

public class Main {
    public static void main(String[] args) {
        A obj = new A();
        obj.print();
        obj.show();
    }
}
```

Output:

Printing...

Showing...

5. Hybrid Inheritance

Hybrid = Combination of types (like multiple + multilevel).

Java **doesn't allow hybrid using classes**, but you can simulate it using **interfaces**.

```
interface A {
    void msg();
}

interface B {
    void msg();
}

class C {
    void greet() {
        System.out.println("Hello from class C");
    }
}

class D extends C implements A, B {
    public void msg() {
        System.out.println("Message from D");
    }
}

public class Main {
    public static void main(String[] args) {
        D obj = new D();
        obj.greet();
        obj.msg();
    }
}
```

Output:

Hello from class C

Message from D

Summary Table

Type of Inheritance	Supported	Achieved By
Single	✅	extends
Multilevel	✅	extends
Hierarchical	✅	extends
Multiple	✅ (via interface)	implements
Hybrid	✅ (via interface)	extends + implements