Java 8 introduced **major enhancements** to interfaces, making them more powerful and flexible, especially for designing **APIs** and supporting **functional programming**.

**🔷 What’s New in Java 8 Interfaces?**

Java 8 added two key features to interfaces:

1. ✅ **Default Methods**
2. ✅ **Static Methods**
3. ✅ **Functional Interfaces** (with Lambda support)

Let’s explore each in detail.

**✅ 1. Default Methods**

Java 8 allows **method definitions inside interfaces** using the default keyword. This helps in **interface evolution** without breaking existing implementations.

**🔹 Syntax:**

interface MyInterface {

default void show() {

System.out.println("Default implementation");

}

}

**🔹 Example:**

interface Vehicle {

default void start() {

System.out.println("Starting vehicle...");

}

}

class Car implements Vehicle {

// Inherits default method

}

new Car().start(); // Output: Starting vehicle...

**✅ Why Default Methods?**

To add new methods to interfaces **without breaking old code**.

**✅ 2. Static Methods in Interfaces**

Java 8 allows **static methods** inside interfaces—used like utility/helper methods.

**🔹 Syntax:**

interface MathUtil {

static int square(int x) {

return x \* x;

}

}

**🔹 Usage:**

int result = MathUtil.square(5); // 25

**✅ Why Static Methods?**

To group **helper functions** logically with the interface instead of separate utility classes.

**✅ 3. Functional Interfaces**

An interface with **exactly one abstract method** is a **functional interface**. These are used to support **lambda expressions**.

**🔹 Examples:**

* Runnable → void run()
* Callable<V> → V call()
* Comparator<T> → int compare(T a, T b)
* Function<T, R>, Consumer<T>, Supplier<T>, Predicate<T>

**🔹 Custom Functional Interface:**

@FunctionalInterface

interface Calculator {

int operate(int a, int b);

}

Used with Lambdas:

Calculator add = (a, b) -> a + b;

System.out.println(add.operate(5, 3)); // 8

**✅ Why Functional Interfaces?**

To support **functional programming**, **cleaner syntax**, and **higher-order functions**.

**🔁 Interface Inheritance Rules (Java 8)**

**Multiple Interface Conflicts:**

If a class implements multiple interfaces that have the **same default method**, the class must **override it**.

interface A {

default void greet() { System.out.println("Hello from A"); }

}

interface B {

default void greet() { System.out.println("Hello from B"); }

}

class MyClass implements A, B {

public void greet() {

A.super.greet(); // or B.super.greet()

}

}

**📦 Real-world Use Case Example**

interface Logger {

default void log(String message) {

System.out.println("Log: " + message);

}

static void printInfo(String info) {

System.out.println("INFO: " + info);

}

}

class Service implements Logger {

public void execute() {

log("Executing Service");

}

}

new Service().execute(); // Log: Executing Service

Logger.printInfo("Ready!"); // INFO: Ready!

**🧠 Summary Table**

| **Feature** | **Description** |
| --- | --- |
| default method | Adds method with body inside interface |
| static method | Utility method, called via Interface name |
| Functional interface | Interface with single abstract method |
| Lambda support | Works with functional interfaces |
| Conflict resolution | Requires override if default method clashes |

**🔎 Common Interview Questions**

1. Can interfaces have constructors? → ❌ No.
2. Can interface have static method body? → ✅ Yes.
3. Can default method override Object methods like equals()? → ❌ No.
4. Can an abstract class have default methods? → ❌ Not needed (it already has implementations).
5. Can interfaces extend other interfaces with default methods? → ✅ Yes.