Java Interface – Full Pledged Notes (From Scratch to Advanced)

---

1. What is an Interface in Java?

An **interface** in Java is a blueprint of a class. It is used to achieve **100% abstraction (initially)** and to specify **what a class must do**, not **how it does**.

- An interface contains **method declarations** (abstract methods), **constants**, and (from Java 8 onwards) **default and static methods**.
- Interfaces represent **requirements / contracts**.

□ **Real-life example:**

- Driving rules are an interface.
- Different people (classes) implement those rules in their own way.

---

2. Why Interface is Needed?

Interfaces are needed to:

- Achieve **abstraction**
- Support **multiple inheritance** (which classes cannot do)
- Achieve **loose coupling**
- Achieve **polymorphism**
- Enforce **standardization**
- Define **common behavior** across unrelated classes

---

3. Syntax of Interface

```
interface Calc {
    void add(int a, int b);
    void sub(int a, int b);
}
```

Important Rules:

- Methods are **public and abstract by default**
- Variables are **public static final by default**

4. Implementing an Interface

A class uses the `implements` keyword to implement an interface.

```
class MyCalc1 implements Calc {
    public void add(int a, int b) {
        System.out.println(a + b);
    }

    public void sub(int a, int b) {
        System.out.println(a - b);
    }
}
```

□ **Rule:**

- The implementing class **must override all abstract methods** of the interface.
- Otherwise, the class must be declared **abstract**.

---

5. Interface and Polymorphism

```
class MyCalc2 implements Calc {
    public void add(int a, int b) {
        System.out.println("Sum is: " + (a + b));
    }

    public void sub(int a, int b) {
        System.out.println("Difference is: " + (a - b));
    }
}

public class LaunchInterface {
    public static void main(String[] args) {
        Calc c1 = new MyCalc1();
        c1.add(10, 5);

        Calc c2 = new MyCalc2();
        c2.add(20, 10);
    }
}
```

□ Same method signature, **different implementation → Runtime Polymorphism**

---

6. Key Points About Interface

- We **cannot create objects** of an interface
- We **can create reference variables** of interface type
- Method signatures must be **same** in all implementing classes
- Interfaces help achieve **plug-and-play architecture**

7. Variables in Interface

```
interface Demo {
    int x = 10;
}
```

Equivalent to:

```
public static final int x = 10;
```

 Interface variables are:

- public
- static
- final

---

8. Multiple Inheritance Using Interface

```
interface A {
    void m1();
}

interface B {
    void m2();
}

class C implements A, B {
    public void m1() {}
    public void m2() {}
}
```

 Java supports **multiple inheritance using interfaces**.

---

9. Class vs Interface

Feature	Class	Interface
Object creation	Yes	No
Multiple inheritance	No	Yes
Variables	Any type	public static final
Methods	Concrete + abstract	abstract, default, static
Constructor	Yes	No

10. Can Interface Extend Another Interface?

□ YES

```
interface A {
    void m1();
}

interface B extends A {
    void m2();
}
```

□ Interface **cannot extend a class**

---

11. Default Methods (Java 8)

Default methods have **method body inside interface**.

```
interface Demo {
    default void show() {
        System.out.println("Default method");
    }
}
```

□ Why default methods?

- To add new methods to interfaces **without breaking existing code**

---

12. Overriding Default Methods

- Overriding default method is **optional**
- Can be overridden if required

---

13. Static Methods in Interface (Java 8)

```
interface Demo {
    static void display() {
        System.out.println("Static method");
    }
}
```

- Static methods are **not inherited**
- Must be called using **interface name**

```
Demo.display();
```

14. Private Methods in Interface (Java 9)

- Used internally by default methods
- Cannot be accessed by implementing classes

---

15. Marker (Tag) Interface

An interface with **no methods** is called a marker interface.

Examples:

- Serializable
- Cloneable
- RandomAccess

□ Used to **inform JVM** to perform special behavior.

---

16. Interface and Abstract Class Relationship

- If a class implements an interface but does not implement all methods → must be **abstract**
- Abstract class can implement interface partially

---

17. Interface in Frameworks

Interfaces are heavily used in:

- JDBC → Driver, Connection
- Spring → BeanFactory
- Hibernate → Session

□ Interface promotes **loose coupling** and **dependency injection**.

---

18. Interface Reference and Downcasting

- Interface reference can access only interface methods
- Class-specific methods require **downcasting**

19. Important Interview Points

- Interface methods are public by default
- No constructors in interface
- Interface supports multiple inheritance
- Default methods introduced in Java 8
- Static methods are not inherited
- Interface cannot extend a class