11 Essential Rules of Java Interfaces

Contracts, Polymorphism, and the "Diamond Problem" Solved.

Swipe to Learn ➔

1. Contract & Polymorphism

Rule 1: The Contract

An interface acts as a **contract**. When a class implements an interface, it *promises* to provide the behavior defined in that contract.

Rule 2: Polymorphism

Interfaces enable loose coupling. You can use the Interface as the reference type for the object.

```
Calculator c = new MyCalculator();
// Parent reference, Child Object
```

2. Automatic Modifiers

You don't need to write everything explicitly. The compiler helps you.

Rule 3: Methods

Methods are implicitly **public abstract**.

Rule 4: Variables

Variables are implicitly **public static final** (Constants).

> **If you type:**
> `int count = 10;`
>
> **The compiler sees:**
> `public static final int count = 10;`

3. The Access Scope

Rule 5: Specialized Methods

You cannot access methods that exist *only* in the child class if you are using the Interface reference.

```
Calculator c = new MyCalculator();

c.add(); // Works (in interface)
c.mul(); // ERROR (not in interface)
```

⚠️ The reference 'c' only knows what is defined in the 'Calculator' interface.

4. Implementation Logic

Rule 6: Partial Implementation

If a class implements an interface but defines only *some* of the methods, that class must be declared abstract.

Rule 7: Multiple Inheritance

Classes cannot extend multiple classes, but they **can implement multiple interfaces**.

> This solves the **Diamond Problem** because interface methods are abstract (no body), so there is no ambiguity.

5. Interface vs. Interface

Rule 8: No Implementation

An interface **cannot** implement another interface.

`interface A implements B` ❌

Rule 9: Inheritance

An interface **can extend** another interface (or multiple interfaces!).

`interface A extends B, C` ✅

6. The Golden Order

Rule 10: Extends first!

A class can extend a parent class AND implement an interface simultaneously, but order matters.

```
class A extends B implements C {
  // correct syntax
}
```

Tip: Alphabetical order.
E (xtends) comes before **I** (mplements).

7. Marker Interfaces

Rule 11: Empty Interfaces

An interface with **no methods or fields** is called a Marker (or Tagged) Interface.

> **Common Examples:**
> 1. `Serializable`
> 2. `Cloneable`
> 3. `Remote`

They are used to signal special instructions to the JVM (Java Virtual Machine).

Summary

- ✅ Interfaces = Contracts (Abstraction)
- ✅ Variables are implicitly Constants
- ✅ Supports Multiple Inheritance
- ✅ **Extends** comes before **Implements**
- ✅ Partial implementation = Abstract Class
- ✅ Marker Interfaces have 0 methods