✅ What is an Abstract Class?

An abstract class provides a base definition for other classes to extend, while enforcing specific behaviors through abstract methods that must be overridden by subclasses.

✅ Abstract Method

- Syntax:

  abstract returnType methodName(parameter-list);

- Has no body.
- Must be overridden by the subclass.
- Acts as a “subclass’s responsibility”.

✅ Rules of Abstract Classes

Rule	**Description**
Declaration	Must be declared using abstract keyword.
Abstract Methods	Must be implemented subclass.
Instantiation	Cannot create objects of abstract class.
Constructor	Can have constructor, but cannot be used to directly instantiate.
Static Methods	Allowed (but cannot be abstract).
Abstract Static	❌ Not allowed (makes no sense to call unimplemented method via class name).
Contains	Abstract and concrete (implemented) methods both.
Polymorphism	Can be used as reference types for dynamic method dispatch.

✅ Abstract Class Example:

```
abstract class Animal {
    abstract void sound();  // abstract method

    void breathe() {        // concrete method
        System.out.println("Breathing...");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Barks");
    }
}
```

🔶 Key Concepts and Notes

✅ Why No Abstract Static Methods?

- Static methods are called via class name.
- Abstract methods have no body and are **meant to be overridden**.
- Hence, combining both **doesn't make sense** in Java.

🔹 What is an Interface?

- An interface in Java is a reference type, similar to a class, but can only contain abstract method declarations (until Java 7).
- Introduced to solve the problem of multiple inheritance in Java.

🔹 Interface Basics

Feature	Interface
Inheritance	Supports **multiple inheritance**
Methods (Java 7)	Only abstract methods (implicitly public abstract)
Methods (Java 8+)	Can have default, static, and private methods
Variables	All are public static final (constants)
Constructors	❌ Not allowed
State	❌ Cannot maintain state (no instance variables)

✅ Syntax

```
interface Drawable {
    void draw(); // implicitly public and abstract
}
class Circle implements Drawable {
    public void draw() {
        System.out.println("Drawing circle...");
    }
}
```

🔹 Interface vs Class

Feature	Interface	Class
Inheritance	Can be implemented by multiple classes	Can extend only one class
Method Bodies (Pre-Java 8)	Not allowed	Allowed

Feature	Interface	Class
Variables	public static final only	Any type allowed
State	❌ Cannot hold instance state	✅ Can hold state
Constructors	❌ No constructors	✅ Constructors allowed

🔹 **Key Properties of Interfaces**

- All methods are implicitly:
  - public abstract (unless they're default, static, or private in Java 8+)
- All variables are implicitly:
  - public static final (must be initialized)
- Cannot instantiate an interface.
- A class can implement **multiple interfaces**.

🔹 **Interface Example with Polymorphism**

```
interface Animal {
    void sound();
}

class Dog implements Animal {
    public void sound() {
        System.out.println("Barks");
    }
}

class Cat implements Animal {
    public void sound() {
        System.out.println("Meows");
    }
}

class Test {
    public static void main(String[] args) {
        Animal a = new Dog(); // reference via interface
        a.sound(); // Outputs: Barks
    }
}
```

🔹 Interface as a Polymorphic Reference

- Interface references can be used like superclass references.
- Helps in **dynamic method resolution**:
    - Method called at **runtime** based on the object being referenced.

🔹 Interface Extension

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

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

- A class implementing B must implement **both show() and display()**.

🔹 Default Methods (Java 8)

✅ Purpose:

Allow interfaces to evolve by adding new methods **without breaking existing implementations**.

✅ Syntax:

```
interface MyInterface {
    default void greet() {
        System.out.println("Hello!");
    }
}
```

✅ Key Rules:

- A class method always takes priority over interface default methods.
- If a class implements **two interfaces with the same default method**, it **must override** it to resolve ambiguity.
- Default methods **can be overridden** in the implementing class.

🔹 Static Methods in Interfaces (Java 8+)

- Must have a method body.
- Not inherited by implementing classes or subinterfaces.

```
interface Utility {
    static int square(int x) {
        return x * x;
    }
}
int result = Utility.square(5); // Accessed via interface name
```

🔹 Nested Interfaces

- Interface declared **inside a class or another interface**.
- Can have access modifiers: public, private, or protected.

```
class A {
    public interface NestedIF {
        boolean isNotNegative(int x);
    }
}

class B implements A.NestedIF {
    public boolean isNotNegative(int x) {
        return x >= 0;
    }
}
```

🔹 Access Modifiers in Interface Implementation

- All interface methods must be public when implemented.
- Access modifier must be same or more accessible than the original.
  - E.g., protected in interface → must be protected or public in implementation.

⚠️ Interface Notes and Cautions

- Cannot **store instance data** (state) → no instance variables.
- Overusing interfaces can lead to **performance overhead** due to dynamic method resolution.
- Ideal for **defining contracts** between unrelated classes.
- Do not use interfaces **casually** in performance-critical code.

- Use abstract classes if **partial implementation or state management** is needed.

🔄 Abstract Class vs Interface

Feature	**Abstract Class**	**Interface**
Methods	Abstract + Concrete	Only abstract (Java 7), + default/static (Java 8+)
Method Access	Can be private, protected, etc.	All methods are public
Variables	Any type (final, static, non-final)	public static final only
Multiple Inheritance	❌ Only one class can be extended	✅ Can implement multiple interfaces
Constructors	✅ Allowed	❌ Not allowed
State	✅ Can maintain state	❌ Cannot hold instance variables
Use When	Partial implementation needed	Only behavior declaration is needed
Extending/Implementing	extends	implements
Inheritance	Can extend one class and implement multiple interfaces	Can extend multiple interfaces