10.5-What is Interface

What is an Interface in Java?

In Java, the abstract keyword is used at the class level to make a class abstract. An abstract class can contain methods without implementation, called **abstract methods**, as well as methods with implementation, called **concrete methods**. However, when a class has only abstract methods, an alternative to using an abstract class to achieve abstraction we use an **interface**.

By using interface we can achieve 100% abstraction.

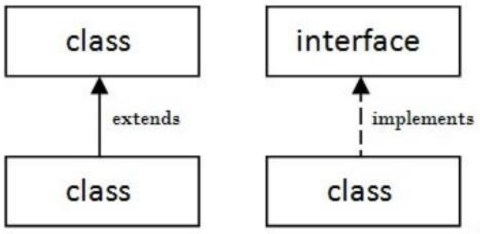
What is an Interface?

An **interface** in Java is a blueprint of a class that can have static constants and abstract methods. It is a mechanism to achieve **abstraction** and **multiple inheritance** in Java.

- Interfaces **cannot have method bodies**; they only declare methods.
- You cannot instantiate an interface directly, similar to abstract classes.
- Methods declared in an interface are, by default, marked as **public** and **abstract**.
- Variables in an interface are implicitly **public**, **static**, and **final**.

Key Features of an Interface

- **Total Abstraction:** All methods in an interface are abstract (i.e., they have no implementation). This means that the class implementing the interface must provide implementations for all declared methods.
- **IS-A Relationship:** Interfaces represent the "IS-A" relationship in Java, similar to inheritance.
- **Implements Keyword:** A class uses the **implements** keyword to indicate that it provides implementations for the interface methods.



- **Static and Final Variables:** Variables declared in an interface are static and final by default, meaning they are constants that cannot be changed once initialized.

Syntax of an Interface

To declare an interface, the interface keyword is used. Here's the syntax for declaring an interface:

```
interface InterfaceName {
    // Declare constant fields
    // Declare abstract methods
}
```

Example:

```
interface A {
    void show();  // implicitly public and abstract
    void config();
}
```

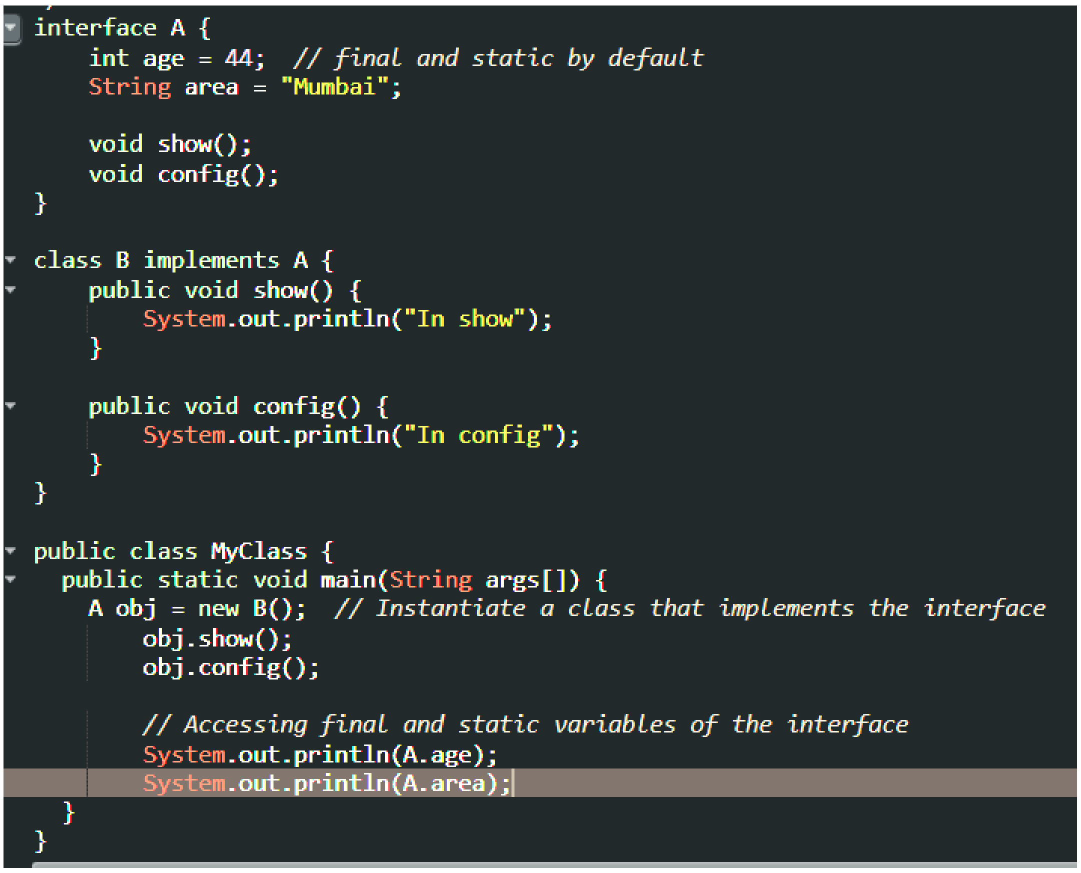
Instantiating an Interface

You cannot directly create an object of an interface, just like an abstract class. The following code will result in an error:

```
A obj = new A();  // Error: Cannot instantiate the type A
```

However, you can instantiate a class that implements the interface, as shown in the example below.

Example of Interface Implementation



```java
interface A {
    int age = 44;  // final and static by default
    String area = "Mumbai";

    void show();
    void config();
}

class B implements A {
    public void show() {
        System.out.println("In show");
    }

    public void config() {
        System.out.println("In config");
    }
}

public class MyClass {
  public static void main(String args[]) {
    A obj = new B();  // Instantiate a class that implements the interface
        obj.show();
        obj.config();

        // Accessing final and static variables of the interface
        System.out.println(A.age);
        System.out.println(A.area);
  }
}
```

Explanation of Interface Usage

An interface can be considered a **contract** between the class and the interface itself. It defines **unimplemented methods** that the class must provide implementations for. In essence, an interface lays down the **requirements** (methods and constants), which the implementing class must fulfill.

In software design, interfaces are sometimes referred to as **Service Requirement Specifications (SRS)** because they specify the services or methods that classes need to implement.

Important Points About Interfaces in Java

- Interfaces are not classes but provide a structure similar to classes.
- Methods in interfaces are always public and abstract.
- Variables declared in interfaces are constants by default (public, static, final).
- A class that implements an interface must provide implementations for all its methods.
- A class can implement multiple interfaces, thus achieving multiple inheritance.