10.15-Types of Interface

Interface in Java

In Java, an **interface** is a mechanism to achieve **abstraction**. Interfaces define a contract of methods that must be implemented by any class that implements the interface. Interfaces can only contain **abstract methods** (methods without a body) in earlier versions of Java, but from Java 8 onward, they can also contain **default** and **static methods**. Interfaces help to achieve multiple inheritance in Java, as a class can implement multiple interfaces.

Types of Interfaces in Java

There are three main types of interfaces in Java:

1. **Normal Interface**
2. **Functional Interface** (SAM – Single Abstract Method Interface)
3. **Marker Interface**

---

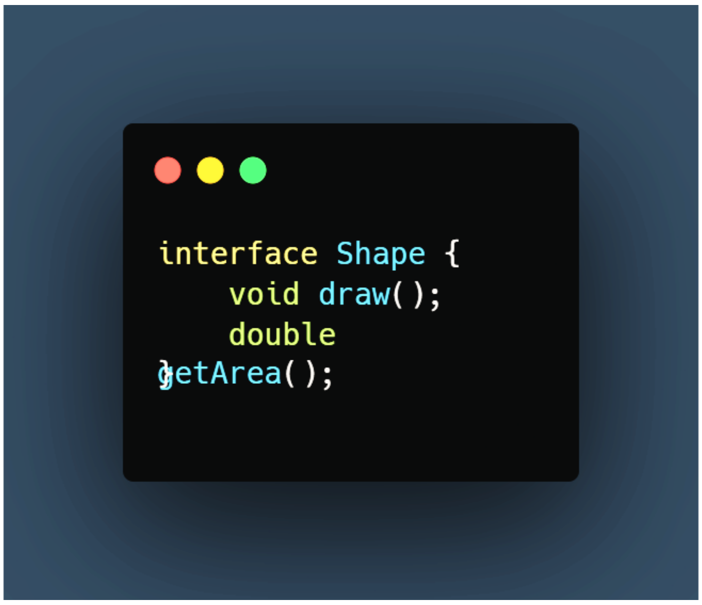
1. Normal Interface

A **normal interface** is an interface that contains **two or more abstract methods**. These interfaces are the most commonly used in Java. Any class that implements this interface must provide an implementation for all its abstract methods.

Starting from Java 8, normal interfaces can also include:

- **Default methods** (methods with a body that have a default implementation).
- **Static methods** (methods that belong to the interface class, not to an instance of the class).

Example:



```
interface Shape {
    void draw();
    double
}getArea();
```

In this example, Shape is a normal interface with two abstract methods: draw() and getArea(). Any class implementing this interface must define both methods.

---

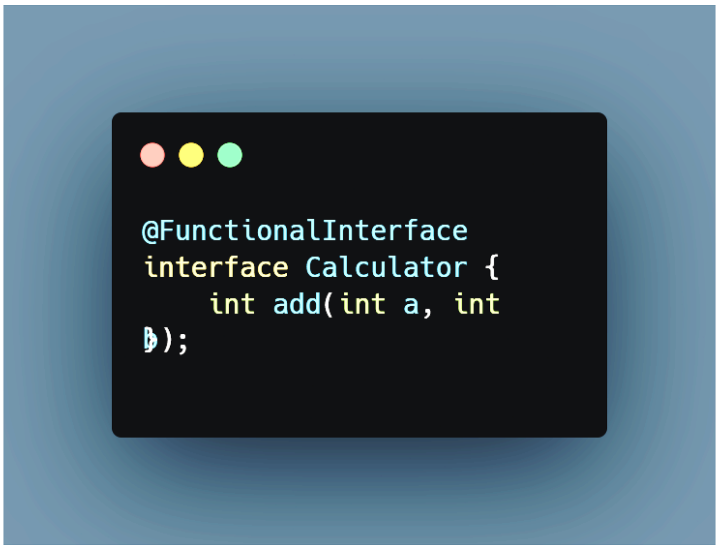
2. Functional Interface

A **functional interface** is an interface that contains **only one abstract method**. This type of interface is also called **SAM** (Single Abstract Method) interface. Functional interfaces can have any number of:

- **Static methods**
- **Default methods**
- **Methods from java.lang.Object**

Lambda expressions are typically used with functional interfaces in Java.

Example of Functional Interface:



```
@FunctionalInterface
interface Calculator {
    int add(int a, int
b);
```

Here, Calculator is a functional interface with a single abstract method add(int a, int b).

Important Note: Even though it has one abstract method, a functional interface can have multiple default and static methods.

---

3. Marker Interface

A **marker interface** is an interface that does **not contain any methods, fields, or constants**. Marker interfaces serve the purpose of **tagging** or **marking** a class as having a particular property. These interfaces rely on runtime behaviour.

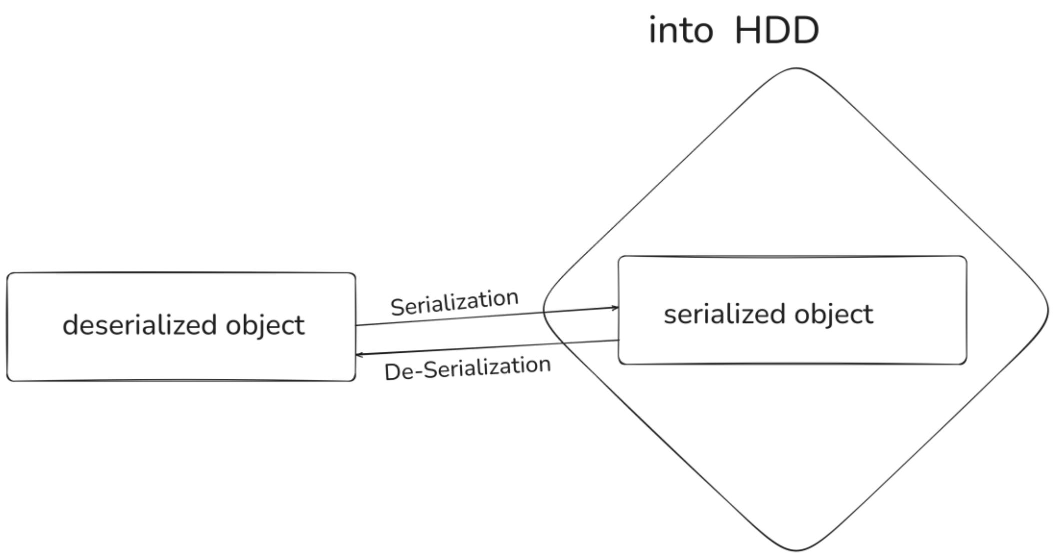
Examples of Marker Interfaces:

- **Serializable**
- **Cloneable**

Serializable Interface:

The **Serializable** interface is a marker interface defined in the java.io package. It is used to make a class **serializable**, meaning that the state of an object can be converted into a byte stream and stored in a file or database.

- **Serialization**: The process of converting an object's state to a byte stream, making it suitable for storage or transmission.
- **Deserialization**: The reverse process of restoring the object's state from a byte stream back into memory.



Key Points to Remember:

- **Normal Interface**: Contains two or more abstract methods. From Java 8 onward, they can also have default and static methods.
- **Functional Interface**: Contains only one abstract method. Can be used with lambda expressions.
- **Marker Interface**: Does not contain any methods or fields, used for tagging or marking classes (e.g., Serializable).