JAVA INTERFACE

An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.

- The interface in Java is a mechanism to achieve abstraction.
- By default, variables in an interface are public, static, and final.
- It is used to achieve abstraction and multiple inheritances in Java.
- In other words, interfaces primarily define methods that other classes must implement.

```
import java.io.*;
// Interface Declared
interface testInterface {
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}
// Class implementing interface
class TestClass implements testInterface {

    // Implementing the capabilities of
    // Interface
    public void display(){
      System.out.println("Geek");
    }
```

```
}
class Geeks
{
    public static void main(String[] args)
    {
        TestClass t = new TestClass();
        t.display();
        System.out.println(t.a);
    }
}
```

Java Class vs Interfaces

In Java, the difference between a class and an interface is syntactically similar, both contain methods and variables, but they are different in many aspects. The main difference is,

- A **class** defines the state of behavior of objects.
- An **interface** defines the methods that a class must implement.

Class vs Interface

The following table lists all the **major differences between an interface and a class in Java**.

Features	Class	Interface
Keyword	The keyword used to create a class is “class”.	The keyword used to create an interface is “interface”.
Instantiation	A class can be instantiated i.e., objects of a class can be created.	The keyword used to create an interface is “interface”
Inheritance	Classes do not support multiple inheritance.	Interface supports multiple inheritance.

Features	Class	Interface
Inheritance Mechanism	A class can inherit another class using the keyword extends.	A class can implement an interface using the keyword “implements”. An interface can inherit another interface using “extends”.
Constructors	It can contain constructors.	It cannot contain constructors.
Methods	Methods in a class can be abstract, concrete, or both.	An interface contains abstract methods by default (before Java 8) or default/static methods (from Java 8 onward).
Access Specifiers	Variables and methods in a class can be declared using any access specifier(public, private, default, protected).	All variables and methods in an interface are declared as public
Variables	Variables in a class can be static, final, or neither.	All variables are static and final.
Purpose	A class is a blueprint for creating objects and encapsulates data and behavior.	An interface specifies a contract for classes to implement by focusing on capabilities rather than implementation

Nested Interface in Java

We can declare **interfaces as members of a class or another interface**. Such an interface is called a **member interface or nested interface**. Interfaces declared outside any class can have only public and default (package-private) access specifiers. In Java, **nested interfaces** (interfaces declared inside a class or another interface) can be declared with the **public, protected, package-private (default), or private access specifiers.**

- A nested interface can be declared public, protected, package-private (default), or private.

- A top-level interface (not nested) can only be declared as public or package-private (default). It cannot be declared as protected or private.

```
// working of interface inside a class
import java.util.*;

// Parent Class
class Parent {

    // Nested Interface
   interface Test {
      void show();
   }
}

// Child Class
class Child implements Parent.Test {
   public void show()
   {
      System.out.println("show method of interface");
   }
}

class GFG
{
   public static void main(String[] args)
   {
        // instance of Parent class
```

```
        // with Nested Interface
        Parent.Test obj;

        // Instance of Child class
        Child t = new Child();

        obj = t;
        obj.show();
    }
}
```

Interface in Another Interface

An interface can be declared inside another interface also. We mention the interface as **Parent.Test** where **Parent** is the name of the interface in which it is nested and **Test** is the name of the interface to be implemented**.**

```
// working of interface inside another interface

import java.util.*;

// Nested Interface-Interface

interface Parent {

    interface Test {

        void show();

    }

}

class Child implements Parent.Test {

    public void show() {
```

```
        System.out.println("show method of interface");
    }
}
// Main Class
class GFG
{
    public static void main(String[] args)
    {
        Parent.Test obj;
        Child t = new Child();
        obj = t;
        obj.show();
    }
}
```

Uses of Nested Interfaces

In Java, nested interfaces can be used for a variety of purposes, including:

- **To group related interfaces together:** By nesting one interface within another, you can organize related interfaces in a more logical and readable way. This can make your code easier to understand and maintain.
- **To create more secure code:** By making an interface nested inside a class, you can limit its scope and prevent it from being accessed outside of that class. This can make your code more secure and less prone to errors.

- **To implement multiple interfaces:** By nesting interfaces, you can implement multiple interfaces in a single class, without cluttering up the global namespace with too many interface names.
- **To create callbacks:** Nested interfaces can be used to create callback functions, where an object can be passed to another object and that object can call back a method defined in the nested interface.
- **To define a contract between classes:** By using nested interfaces, you can define a contract between classes, where each class implements the same interface, but provides its own implementation. This can make your code more modular and easier to test.

Interfaces and Inheritance in Java

A class can extend another class and can implement one and more than one Java interface. Also, this topic has a major influence on the concept of Java and Multiple Inheritance.

Inheritance is inheriting the properties of the parent class into the child class.

1. Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
2. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.
3. You can also add new methods and fields in your current class.
4. Inheritance represents the ***IS-A relationship*** which is also known as the ***parent-child relationship***

```
// Java program to demonstrate inheritance in
// interfaces.
import java.io.*;


interface intfA
{
   void geekName();
}
```

```
interface intfB extends intfA
{
    void geekInstitute();
}

// class implements both interfaces and provides
// implementation to the method.
class sample implements intfB
{

    @Override
      public void geekName() {
        System.out.println("Rohit");
    }

    @Override
      public void geekInstitute() {
        System.out.println("JIIT");
    }

    public static void main(String[] args)
    {
        sample ob1 = new sample();

        // calling the method implemented
        // within the class.
```

```
        ob1.geekName();
        ob1.geekInstitute();
    }
}
```

An interface can also extend multiple interfaces.

```
// multiple inheritance in interfaces
import java.io.*;

interface intfA
{
    void geekName();
}

interface intfB
{
    void geekInstitute();
}

// always remember that interfaces always
// extends interface but a class always
// implements a interface
interface intfC extends intfA, intfB {
    void geekBranch();
}

// class implements both interfaces and provides
// implementation to the method.
```

```
class sample implements intfC
{
    public void geekName() {
        System.out.println("Rohit");
    }

    public void geekInstitute() {
        System.out.println("JIIT");
    }

    public void geekBranch() {
        System.out.println("CSE");
    }

    public static void main(String[] args)
    {
        sample ob1 = new sample();

        // calling the method implemented
        // within the class.
        ob1.geekName();
        ob1.geekInstitute();
        ob1.geekBranch();
    }
}
```

Why Multiple Inheritance is not supported through a class in Java, but it can be possible through the interface?

Multiple Inheritance is not supported by class because of ambiguity. In the case of interface, there is no ambiguity because the implementation of the method(s) is provided by the implementing class up to Java 7. From Java 8, interfaces also have implementations of methods. So if a class implements two or more interfaces having the same method signature with implementation, it is mandated to implement the method in class also.