Unit 3
Interface with default and static method

```
import java.io.*;
interface intfA
{
	void m1();
}
interface intfB
{
	void m2();
}
// class implements both interfaces
// and provides implementation to the method.
class sample implements intfA, intfB
{
	@Override
	public void m1()
	{
		System.out.println("Welcome: inside the method m1");
	}
	@Override
	public void m2()
	{
		System.out.println("Welcome: inside the method m2");
	}
}
```

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

                // calling the method implemented
                // within the class.
                ob1.m1();
                ob1.m2();
        }
}
```

Output;

```
Welcome: inside the method m1
Welcome: inside the method m2
```

Interfaces Can Be Extended

- One interface can inherit another by use of the keyword **extends.**
- **The syntax is the same as** for inheriting classes.
- When a class implements an interface that inherits another interface, it must provide implementations for all methods required by the interface inheritance chain.

```
// One interface can extend another.
interface A {
  void meth1();  void meth2();
}
// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
  void meth3();
}
// This class must implement all of A and B
class MyClass implements B {
  public void meth1() {
    System.out.println("Implement meth1().");
  }
  public void meth2() {
    System.out.println("Implement meth2().");
  }
  public void meth3() {
    System.out.println("Implement meth3().");
  }
}
class IFExtend {
  public static void main(String arg[]) {
    MyClass ob = new MyClass();
ob.meth1();
    ob.meth2();
    ob.meth3();
```

Program 2

```
// 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();
        }
}
```

```
Output


Rohit JIIT
```

Default Interface Methods

- A default method lets you define a default implementation for an interface method.
- In other words, by use of a default method, it is now possible for an interface method to provide a body, rather than being abstract.
- During its development, the default method was also referred to as an extension method

- A primary motivation for the default method was to provide a means by which interfaces could be expanded without breaking existing code.
- The default method solves this problem by supplying an implementation that will be used if no other implementation is explicitly provided. Thus, the addition of a default method will not cause preexisting code to break.

Default Method Fundamentals

- An interface default method is defined similar to the way a method is defined by a class.
- The primary difference is that the declaration is preceded by the keyword default. For example, consider this simple interface:

```
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getNumber();

  // This is a default method. Notice that it provides
  // a default implementation.
  default String getString() {
    return "Default String";
  }
}
```

```
// Implement MyIF.
class MyIFImp implements MyIF {
  // Only getNumber() defined by MyIF needs
to be implemented.
  // getString() can be allowed to default.
  public int getNumber() {
    return 100;
  }
}
```

```
// Use the default method.
class DefaultMethodDemo {public static void main(String
args[]) {
    MyIFImp obj = new MyIFImp();
    // Can call getNumber(), because it is explicitly
    // implemented by MyIFImp:
    System.out.println(obj.getNumber());
    // Can also call getString(), because of default
    // implementation:
    System.out.println(obj.getString());
  }
}
```

The output is shown here:

```
100
Default String
```

Program

```
interface TestInterface
{
        // abstract method
        public void square(int a);
        // default method
        default void show()
        {
        System.out.println("Default Method Executed");
        }
}
class TestClass implements TestInterface
{
        // implementation of square abstract method
        public void square(int a)
        {
                System.out.println(a*a);
        }
        public static void main(String args[])
        {
                TestClass d = new TestClass();
                d.square(4);
                // default method executed
                d.show();
        }
}
```

```
Output
16 Default Method Executed
```

The default method gives you
• a way to gracefully evolve interfaces over time, and
• a way to provide optional functionality without requiring that a class provide a
placeholder implementation when that functionality is not needed.

Use static Methods in an Interface

- JDK 8 added another new capability to **interface: the ability to define one or more static** methods.
- Like **static methods in a class, a static method defined by an interface can be** called independently of any object.
- Thus, no implementation of the interface is necessary, and no instance of the interface is required, in order to call a **static method.**
- **Instead, a static method is called by specifying the interface name, followed by a period, followed by** the method name.

Here is the general form:
InterfaceName.staticMethodName

The **static method is getDefaultNumber( ). It** returns zero.

```
public interface MyIF {
  // This is a "normal" interface method declaration.
  // It does NOT define a default implementation.
  int getNumber();

  // This is a default method. Notice that it provides
  // a default implementation.
  default String getString() {
    return "Default String";
  }
// This is a static interface method.
  static int getDefaultNumber() {
    return 0;
  }
}
```

Note: As mentioned, no implementation or instance of **MyIF is required to call getDefaultNumber( ) because it is static.**

One last point: **static interface methods are not inherited by either an implementing** class or a subinterface.

```
// Java program to demonstrate // static method in Interface.
interface NewInterface {
        // static method
        static void hello()
        {
                System.out.println("Hello, New Static Method Here");
        }
        // Public and abstract method of Interface
        void overrideMethod(String str);
}
// Implementation Class
public class InterfaceDemo implements NewInterface {
        public static void main(String[] args)
        {
                InterfaceDemo interfaceDemo = new InterfaceDemo();
                // Calling the static method of interface
                NewInterface.hello();
                // Calling the abstract method of interface
                interfaceDemo.overrideMethod("Hello, Override Method here");
        }
        // Implementing interface method
        @Override
        public void overrideMethod(String str)
        {
                System.out.println(str);
        }
}
```

Output:

```
Hello, New Static Method Here
Hello, Override Method here
```