Default Methods in Java 8

Before Java 8, interfaces could have only abstract methods. The implementation of these methods must be provided in a separate class. So, if a new method is to be added to an interface, then its implementation code must be provided in the class implementing the same interface. To overcome this issue, Java 8 introduced the concept of default methods, which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

The default methods were introduced to provide backward compatibility so that existing interfaces can use lambda expressions without implementing the methods in the implementation class. Default methods are also known as **defender methods or virtual extension methods**.

// A simple program to Test Interface default methods in Java

```
interface TestInterface
{
    public void square(int a);  // abstract method
    default void show() // default method
    {
      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);
        d.show();  // default method executed
    }
}
```

Output:

```
D:\Java Code 2k23>java TestClass
16
Default Method Executed
```

Static Methods:

The interfaces can have static methods as well, which is like the static method of classes.

// A simple Java program to TestClassnstrate static methods in Java

```
interface TestInterface
{
   // abstract method
   public void square (int a);

   // static method
   static void show()
   {
      System.out.println("Static 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);

      // Static method executed
      TestInterface.show();
   }
}
```

Output:

```
16
Static Method Executed
```

Default Methods and Multiple Inheritance

In case both the implemented interfaces contain default methods with the same method signature, the implementing class should explicitly specify which default method is to be used, or it should override the default method.

// A simple Java program to demonstrate multiple inheritance through default methods.

```
interface TestInterface1
{
   // default method
   default void show()
   {
      System.out.println("Default TestInterface1");
   }
}

interface TestInterface2
{
   // Default method
   default void show()
   {
      System.out.println("Default TestInterface2");
   }
}

// Implementation class code
class TestClass implements TestInterface1, TestInterface2
{
   // Overriding default show method
   public void show()
   {
      // use super keyword to call the show
      // method of TestInterface1 interface
      TestInterface1.super.show();

      // use super keyword to call the show
      // method of TestInterface2 interface
      TestInterface2.super.show();
   }
   public static void main(String args[])
   {
      TestClass d = new TestClass();
      d.show();
   }
}
```

Output:

```
Default TestInterface1
Default TestInterface2
```

Important Points: