Default methods

- Until 1.7 version onwards inside the interface we can take only public abstract methods and public static final variables.
- Default methods enable you to add new functionality to existing interfaces and ensure binary compatibility with code written for older versions of those interfaces.
- Default methods, in particular, allow you to extend existing interfaces with methods that accept lambda expressions as parameters.

Introduction

- Interfaces could only have abstract methods before Java 8.
- These methods implementation must be provided in a separate class.
- If a new method is added to an interface, the implementation code for that method must be included in the class that implements the same interface.
- To address this problem, Java 8 introduced default methods, which allow interfaces to include implementation-ready methods without changing the classes that implement the interface.
- Syntax:

```
public interface DemoClass {

    public default void m1() {
        System.out.println("I am default method..!!");
    }

}
```

- This feature has been provided for backward compatibility so that existing interfaces can take advantage of Java 8's lambda expression functionality
- Defender methods or virtual extension methods are other names for default methods.

Default Methods and Multiple Inheritance

- It's possible that a class implements two interfaces with identical default methods because of default functions in interfaces.
- The implementing class should either override the default method or explicitly specify which default method should be used.
- e.g.,

```java
public interface DemoClass {
    public default void m1() {
        System.out.println("Demo class 1");
    }
}
```

```java
public interface DemoClass2 {
    public default void m1() {
        System.out.println("Demo class 2");
    }
}
```

```java
public class Test implements DemoClass, DemoClass2{

    @Override
    public void m1() {
        System.out.println("Overrideen m1 method to avoid ambiguity");
    }
}
```

OR

```java
public class Test implements DemoClass, DemoClass2{

    @Override
    public void m1() {
        DemoClass.super.m1();
    }

}
```

Interface with Default Methods	Abstract Class
Inside interface every variable is Always public static final and there is No chance of instance variables	Inside abstract class there may be a Chance of instance variables which Are required to the child class.
Interface never talks about state of Object.	Abstract class can talk about state of Object.
Inside interface we can't declare Constructors.	Inside abstract class we can declare Constructors.
Inside interface we can't declare Instance and static blocks.	Inside abstract class we can declare Instance and static blocks.
Functional interface with default Methods Can refer lambda expression.	Abstract class can't refer lambda Expressions.
Inside interface we can't override Object class methods.	Inside abstract class we can override Object class methods.

Static Method

- Interfaces can also have static methods, which are similar to class static methods.
- Static methods in Java are those methods whose same copy is shared by all the objects of the class and can be called without creating a class object.
- They are referenced by the class name itself or reference to the object of that class.
- Static methods can also be defined within the interface.
- Utility methods are defined using static methods.
- e.g.,

```
public interface DemoClass {

    public static void m1() {
        System.out.println("I am static method...");
    }
}
```

```java
public class Demo {

    public static void main(String[] args) {
        DemoClass.m1();
    }
}
```