Default Methods

- Until 1.7 version onwards inside interface we can take only public abstract methods and public static final variables (every method present inside interface is always public and abstract whether we are declaring or not).
- Every variable declared inside interface is always public static final whether we are declaring or not.
- But from 1.8 version onwards in addition to these, we can declare default concrete methods also inside interface, which are also known as defender methods.
- We can declare default method with the keyword "default" as follows

```
1)  default void m1(){
2)  System.out.println ("Default Method");
3)  }
```

- Interface default methods are by-default available to all implementation classes. Based on requirement implementation class can use these default methods directly or can override.

Ex:

```
1)  interface Interf {
2)        default void m1() {
3)              System.out.println("Default Method");
4)        }
5)  }
6)  class Test implements Interf {
7)         public static void main(String[] args) {
8)               Test t = new Test();
9)               t.m1();
10)       }
11) }
```

- Default methods also known as defender methods or virtual extension methods.
- The main advantage of default methods is without effecting implementation classes we can add new functionality to the interface (backward compatibility).

Note: We can't override object class methods as default methods inside interface otherwise we get compile time error.

Ex:

```
1)  interface Interf {
2)        default inthashCode() {
3)              return 10;
4)        }
5)  }
```

CompileTimeError

Reason: Object class methods are by-default available to every Java class hence it's not required to bring through default methods.

Default method vs multiple inheritance

Two interfaces can contain default method with same signature then there may be a chance of ambiguity problem (diamond problem) to the implementation class. To overcome this problem compulsory we should override default method in the implementation class otherwise we get compile time error.

```
1)  Eg 1:
2)  interface Left {
3)        default void m1() {
4)              System.out.println("Left Default Method");
5)        }
6)  }
7)
8)  Eg 2:
9)  interface Right {
10)       default void m1() {
11)             System.out.println("Right Default Method");
12)       }
13) }
14)
15) Eg 3:
16) class Test implements Left, Right {}
```

How to override default method in the implementation class?

In the implementation class we can provide complete new implementation or we can call any interface method as follows.
interfacename.super.m1();

Ex:

```
1)  class Test implements Left, Right {
2)          public void m1() {
3)                System.out.println("Test Class Method"); OR Left.super.m1();
4)          }
5)          public static void main(String[] args) {
6)                  Test t = new Test();
7)                  t.m1();
8)          }
9)  }
```

Differences between interface with default methods and abstract class

Even though we can add concrete methods in the form of default methods to the interface, it won't be equal to abstract class.

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.

Interface with default method != abstract class

Static methods inside interface:

- From 1.8 version onwards in addition to default methods we can write static methods also inside interface to define utility functions.
- Interface static methods by-default not available to the implementation classes hence by using implementation class reference we can't call interface static
- methods. We should call interface static methods by using interface name.

Ex:

```
1)  interface Interf {
2)          public static void sum(int a, int b) {
3)                  System.out.println("The Sum:"+(a+b));
4)          }
5)  }
6)  class Test implements Interf {
7)           public static void main(String[] args) {
8)                   Test t = new Test();
9)                   t.sum(10, 20); //CE
10)                  Test.sum(10, 20); //CE
11)                  Interf.sum(10, 20);
12)       }
13) }
```

- As interface static methods by default not available to the implementation class, overriding concept is not applicable.
- Based on our requirement we can define exactly same method in the implementation class, it's valid but not overriding.

Ex:1

```
1)  interface Interf {
2)          public static void m1() {}
3)  }
4)  class Test implements Interf {
5)           public static void m1() {}
6)  }
```

It's valid but not overriding

Ex:2

```
1)  interface Interf {
2)        public static void m1() {}
3)  }
4)  class Test implements Interf {
5)        public void m1() {}
6)  }
```

This's valid but not overriding

Ex3:

```
1)  class P {
2)        private void m1() {}
3)  }
4)  class C extends P {
5)        public void m1() {}
6)  }
```

This's valid but not overriding

From 1.8 version onwards we can write main() method inside interface and hence we can run interface directly from the command prompt.

Ex:

```
1)  interface Interf {
2)        public static void main(String[] args) {
3)              System.out.println("Interface Main Method");
4)        }
5)  }
```

At the command prompt:
Javac Interf.Java
JavaInterf