**Default Methods in Java**

Java 8 introduced several new concepts, including default methods and lambda expressions. Lambda expressions were introduced to better utilize multi-core systems and enable parallel processing. However, in order to use lambda expressions effectively, it was necessary to add methods to some of the interfaces in the Java Collections Framework. Unfortunately, adding methods to a published interface is difficult. All implementing classes will break if a method is added to an interface, since all implementing classes are required to implement the abstract methods provided in the interface.

To get around this problem, Java 8 introduced *default methods* for interfaces.

* A default method is declared in the usual way for interfaces, except that it begins with the word **default**. For example:

**default int foo(int x)**

* A default method includes the implementation of the method.
* You may have more than one default method in an interface.
* Classes implementing an interface can inherit the default method or override it.
* If a class implements two interfaces, each of which has a default method with the same signature, there will be a compile time error. There are two ways to fix this, both of which require overriding the default method.
  + Inside the class, override the default method by providing a new implementation.
  + Inside the class, override the default method by specifying in your implementation which of the default implementations should be used. You can do this by preceding the name of the method to be invoked with the name of the interface and the word **super**. For example, if the method is **foo** and you want to use the implementation from an interface named **A**, you can invoke the method using

**A.super.foo(...)**

**TRY IT!**  Follow the instructions below to get some experience with default methods and interfaces.

1. Create a project named **DefaultMethodInClassExercise**, with a package named **defaultmethods**. Place all java files in this package.
2. Create an interface named **Interface01**.
3. Create an abstract method named **foo** with one **int** parameter named **myInt** and return type **int**.
4. Create a default method named **goo** with one **String** parameter named **myString** and return type **int**. This method prints **Inside Interface01 – goo** and returns the length of the string parameter.
5. Create a class named **MyClass** that implements **Interface01**. You will get an error message. What does the error message say?

1. Override the abstract method **foo(int)** in **MyClass**. The method should print **Inside MyClass-foo** and should return the value of the parameter, multiplied by 2.
2. Create a main class named **DefaultDriver**. Add code to create an instance of **MyClass** named **class01** and execute method **foo**; then execute **goo**. Use 3 as the argument for **foo** and **Goodbye** as the argument for **goo**. Print the result returned by **foo** and **goo**. Write the output below.
3. Create a second interface named **Interface02**.
4. Create a default method named **goo** with one **String** parameter named **myString** and return type **int**. This method prints **Inside Interface02 – goo** and returns the index of the first occurrence of the letter **o** in the string passed as the parameter.
5. Add code to **MyClass** so that **MyClass** implements both **Interface01** and **Interface02**. This results in an error message. What does the error message say?

1. Fix this error by overriding **goo** in **MyClass**. The implementation in **MyClass** should print **Inside MyClass – goo**. Then it should invoke **goo** from **Interface02** and return the result. (To invoke **goo** from **Interface02**, use **Interface02.super.goo(…)**). Run the program and write the output below.
2. Change your solution in the previous step: Override **goo** in **MyClass** so that the index of the last occurrence of **o** will be returned. Before returning the result, the method should print **Inside MyClass – goo**. Write the output below.

**References:**

[**http://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html**](http://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html)

[**http://zeroturnaround.com/rebellabs/java-8-explained-default-methods/**](http://zeroturnaround.com/rebellabs/java-8-explained-default-methods/)

[**http://zeroturnaround.com/rebellabs/how-your-addiction-to-java-8-default-methods-may-make-pandas-sad-and-your-teammates-angry/**](http://zeroturnaround.com/rebellabs/how-your-addiction-to-java-8-default-methods-may-make-pandas-sad-and-your-teammates-angry/)

[**http://javax0.wordpress.com/2014/03/26/the-true-nature-of-java-8-default-methods/**](http://javax0.wordpress.com/2014/03/26/the-true-nature-of-java-8-default-methods/)

[**http://java.dzone.com/articles/interface-default-methods-java**](http://java.dzone.com/articles/interface-default-methods-java)

[**http://www.javaworld.com/article/2139921/learn-java/abstract-class-versus-interface-in-the-jdk-8-era.html**](http://www.javaworld.com/article/2139921/learn-java/abstract-class-versus-interface-in-the-jdk-8-era.html)