Java SE 8: Implementing Default Methods in Interfaces

**Problem Statement:** There is an interface which has one abstract method. There are two classes which implement this interface and provide the implementation for this method. Now, I want to add one more method in this interface.

**Possible Solution**: To add one more method in the interface I will have to provide the implementation for this method in both the class.

What if there are hundred classes?

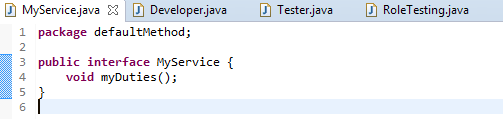
Definitely, I am not going to provide implementation for this method in all the hundred classes.

How this problem can be solved?

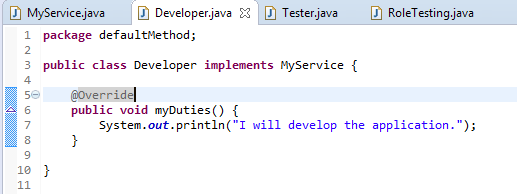
**Example:**

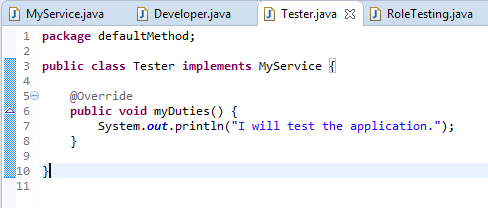
You develop an interface, MyService, and add an abstract method, myDuties(). You develop two implementation classes, Developer and Tester, which implement the MyService interface. You develop a Java class, RoleTesting, to test the application. Finally, you extend the MyService interface by adding another method, extraDuties().

### Developing the Interface

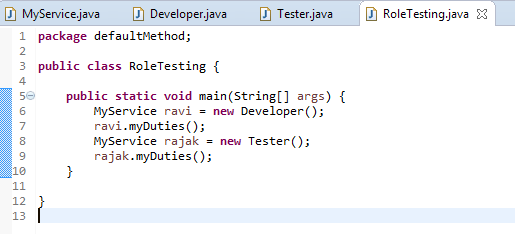


### Developing the Implementation Classes

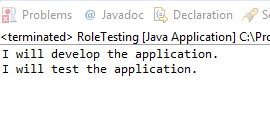




### Testing the Application

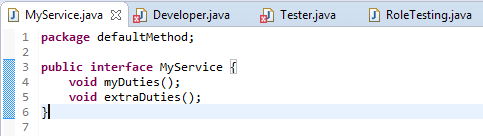


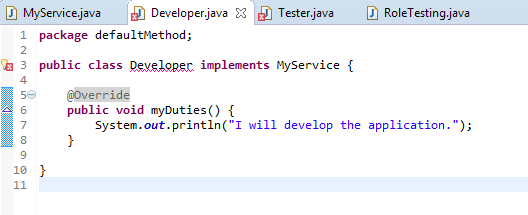
Output of the application

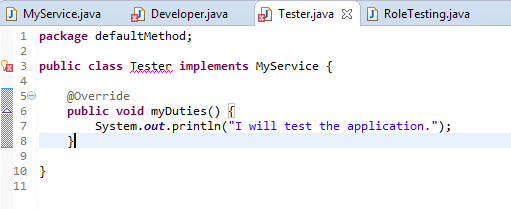


### Extending interface

I added one more method in MyService interface. Now, both the classes are showing error.

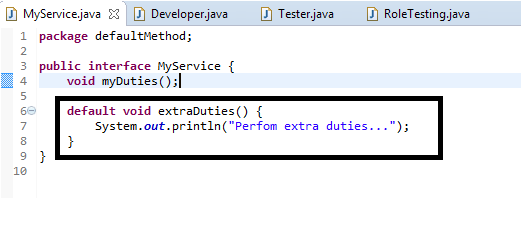






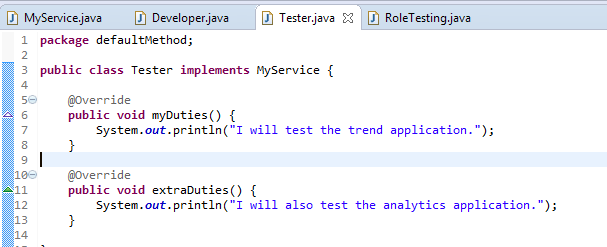
#### Default Method

Prior to Java SE 8, interfaces in Java could contain only method declarations and no implementations, and any non-abstract class implementing the interface had to provide the implementation. This limitation made it almost impossible to extend the existing interfaces and APIs. To overcome this limitation, a new concept, called default methods, is introduced in Java SE 8. The default methods are fully implemented methods in an interface, and they are declared by using the keyword default. Because the default methods have some default implementation, they help extend the interfaces without breaking the existing code.

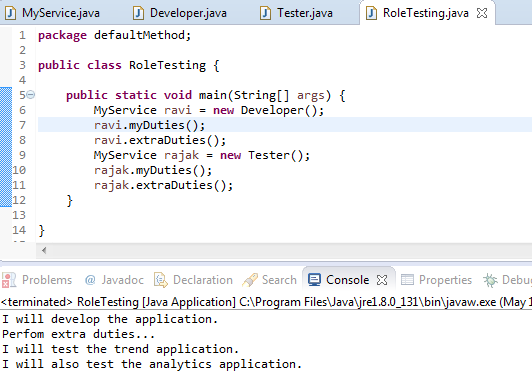


Now this default method is available in both the class and there is no need to provide its implementation in both the classes.

We can also override the existing implementation of this method if required.

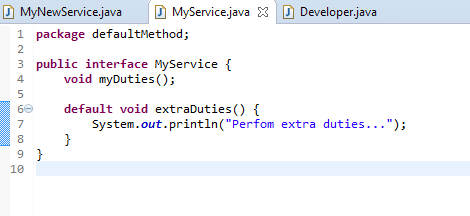


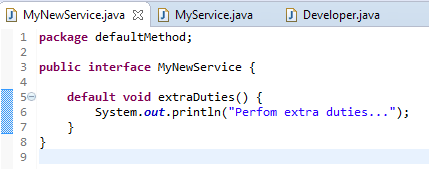
**Output**

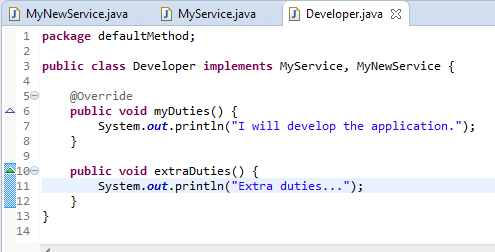


**Implementing Inheritance Rules of Default Methods**

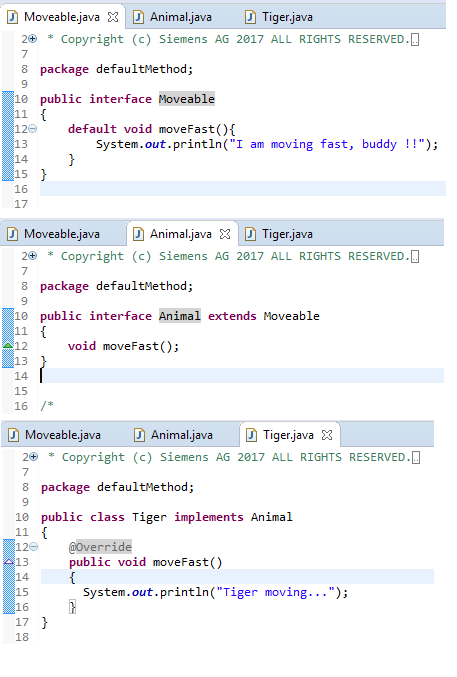
1. A class Developer implements two interfaces ‘MyService’ and ‘MyNewService’ having common default method. So if developer is not overriding this method in class then it’s a compile time error. So in this case he has to override and provide the implementation for this method. From inside the overridden method in the class, he can call the interface MyService or MyNewService default method implementation as well using **MyService.super.methodName()** or **MyNewService.super.methodName().**



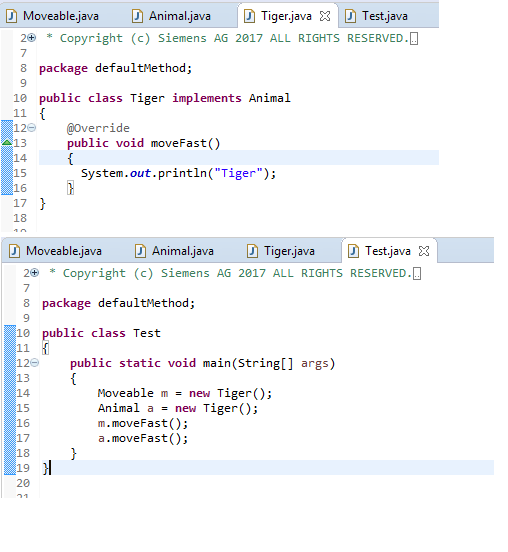
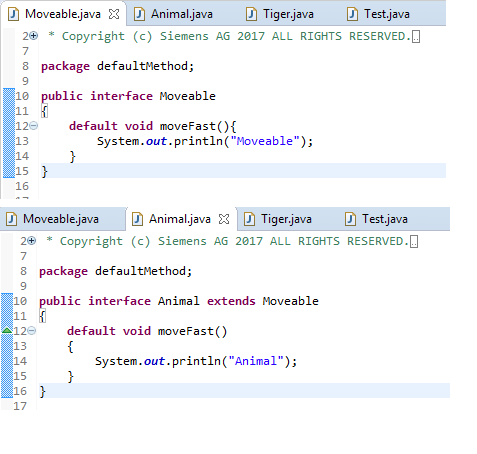




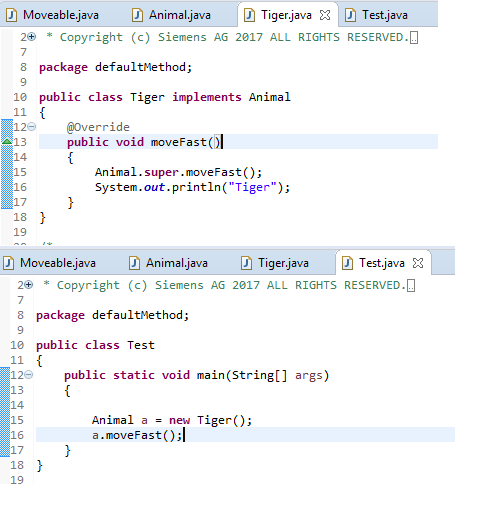
1. If interface Moveable is having a default method. Interface Animal extends this Interface Moveable and redeclares the default method and make it abstract then class Tiger which implement this interface Animal has to provide the implementation of the abstract method.



1. If an interface Moveable is having a default method. Interface Animal extends this interface Moveable and redefine the default method then class Tiger which implement interface Animal will always see the implementation of method in interface Animal.



1. Class always wins over interface. i.e. if class override the default method then defaults are irrelevant until call using **interface.super.methodName().**



This concept has been implemented in interface TrendCursor.java.

Please go through below links.

<http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/JavaSE8DefaultMethods/JavaSE8DefaultMethods.html>

<https://community.oracle.com/thread/3539951>