What is Default, Defender or Extension Method of Java 8 with Example

Java 8 now allows you to add non-abstract method implementations to interfaces by utilizing the `default` and `static` keywords. Methods with default keywords are known as default methods or defender methods in Java. Before Java 8, it was virtually impossible to change an interface once published. Any change like the addition of a new method would have broken all clients. That's why when Java 8 decided to switch to internal iterator implementation using the `forEach()` method, they face the daunting challenge of breaking all implementation of the `Iterable` interface. Since backward compatibility is a top priority for Java engineers, and it wasn't practical to break all clients, they came up with the idea of the *default method*.

This is an amazing and very powerful change because now you can evolve your existing interface with all the knowledge you have gained after using them. JDK itself is utilizing default methods in a big way, `java.util.Map` interface is extended with several new default methods e.g. `replaceAll()`, `putIfAbsent(Key k, Value v)`, and others.

By the way, Since the default method allows extension of the existing interface, it's also known as the *Extension method*. You are also free to define any number of default methods in your interface. I think after this change, you unlikely need an *abstract class* to provide skeletal implementation as described in Effective Java, like, `List` comes with `AbstractList`, `Collection` comes with `AbstractCollection`, `Set` comes with `AbstractSet`, and `Map` comes with A`bstractMap`.

Instead of creating a new abstract class with default implementation, you can define them as default methods inside the interface itself. Similarly, the introduction of static methods inside the interface will make a pattern of an interface utility class redundant e.g. `Collections` for Collection interface, `Paths` for Path, and so on.

You can directly define the static utility method on the interface itself. If you want to learn more about all new features introduced in Java 8, I suggest taking a look at **these best core Java courses** on Udemy. It's one of my favorite Java courses and it covers different features from

both JDK 7 and JDK 8 in good detail.

Java 8 Example of Default Methods

Java 8 enables us to add non-abstract method implementations to interfaces by utilizing the default keyword. This feature is also known as Extension Methods. Here is our first example:

```
interface Multiplication{
    int multiply(int a, int b);

    default int square(int a){
        return multiply(a, a);
    }
}
```

Besides the abstract method `multiply()` the interface `Multiplication` also defines the default method `square()`. Any concrete classes of `Multiplication` interface only have to implement the abstract method `multiply()`. The default method `square()` method can be used directly.

```
Multiplication product = new Multiplication(){


    @Override
```

```
        public int multiply(int x, int y){
            return x*y;
        }
    };

    int square = product.square(2);
    int multiplication = product.multiply(2, 3);
```

The product sub-class is implemented using an anonymous class. The code is quite verbose: 6 lines of code for such a simple multiplication. You can reduce a lot of boilerplate code by using lambda expression, which is also introduced on Java 8. Since our interface contains only one abstract method and Java's lambda expression is of SAM type (Single Abstract method), we can replace anonymous class implementation with just one line of the lambda expression, as shown below :

```
Multiplication lambda = (x, y) -> x*y;

int product = lambda.multiply(3, 4);
int square = lambda.square(4);
```

Here is our complete Java program to demonstrate how you can use *default methods* inside the interface in Java 8. As I said, now, you can even extend your old interface to add new methods without any fear of breaking clients, provided those methods must be either default or static.

```
/**
* Java Program to demonstrate use of default method in Java 8.
 * You can define non-abstract method by using default keyword, and more
 * than one default method is permitted, which allows you to ship default
   skeletal
* implementation on interface itself.
*
* @author Javin Paul
*/
public class Java8DefaultMethodDemo{

    public static void main(String args[]) {

        // Implementing interface using Anonymous class
```

```
        Multiplication product = new Multiplication(){

            @Override
            public int multiply(int x, int y){
                return x*y;
            }
        };

        int squareOfTwo = product.square(2);
        int cubeOfTwo = product.cube(2);

        System.out.println("Square of Two : " + squareOfTwo);
        System.out.println("Cube of Two : " + cubeOfTwo);

        // Since Multiplication has only one abstract method, it can
        // also be implemented using lambda expression in Java 8

        Multiplication lambda = (x, y) -> x*y;

        int squareOfThree = lambda.square(3);
        int cubeOfThree = lambda.cube(3);

        System.out.println("Square of Three : " + squareOfThree);
        System.out.println("Cube of Three : " + cubeOfThree);

    }

}

interface Multiplication{
    int multiply(int a, int b);

    default int square(int a){
        return multiply(a, a);
    }

    default int cube(int a){
        return multiply(multiply(a, a), a);
    }
}
Output :
Square of Two : 4
Cube of Two : 8
```

```
Square of Three : 9
Cube of Three : 27
```

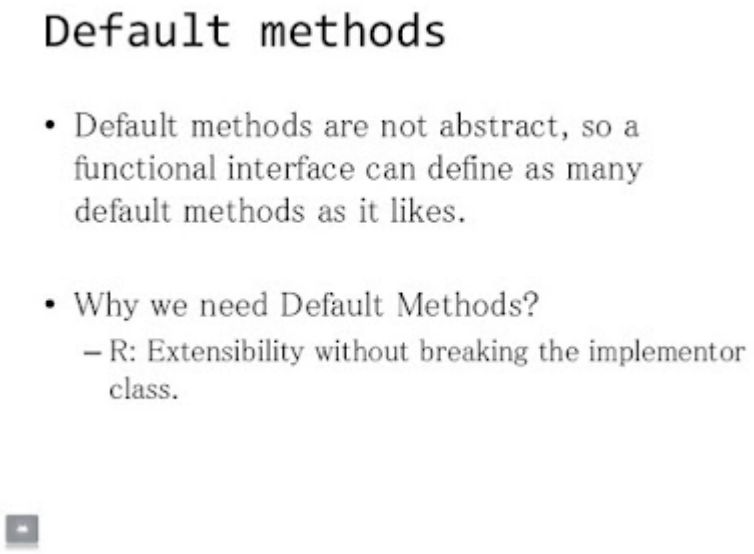
This code is an excellent example of how you can use default methods to add convenient methods to the interface itself. This is also an example of a template method pattern and avoids an extra helper class like Collections, which just provides a utility method to work on Collection.

You can now define such methods in the Collection class itself. In this example of the Java 8 default method, we have an interface Multiplication, which has its core abstract method called `multiply(a, b)`, which is supposed to multiply two numbers.

It has then created two concrete methods using the default keyword, called `square(a)` and `cube(a)`, which depends upon the `multiply(a, b)` method for their function. Now, the client just needs to implement a `multiply()` method, and he will get both `square(a)` and `cube(a)` for free.

Important points about Java 8 Default Methods

Now it's time to revise whatever we have learned so far and note down some of the important things about our new defender, extension, or default method of Java 8. You can take away all the knowledge in the form of these bullet points. It does not only help you to quickly revise the topic but also encourages you to explore further and discover more about those individual things.



1. You can add default methods either on the new interface or existing methods, provided they are compiled using the source version of Java 8.

2. Default methods have blurred the difference between abstract class and interface in Java. So next time while answering this question in the interview, don't forget to mention that you can do some of the things which were only possible with the abstract class using the default keyword. You can now define concrete methods on interfaces with the help of default methods.

3. `default` is not a new keyword, instead, it was reserved from JDK 1.1 for these kinds of evolution.

4. You are free to define any number of default methods in your interface. There is no restriction on the number of default methods an interface can contain in Java 8.

5. If an interface let's say C, extend two interfaces A and B, which has default method with same name and signature then the compiler will complain about this while compiling class C. It's not allowed in Java 8 to avoid ambiguity. So even after default methods, multiple inheritances is still not allowed in Java 8. You cannot extend multiple interfaces with conflicting Java interface default method implementation.

6) There are a lot of examples of Java 8 interface default methods are available in JDK 1.8 codebase, one of the most popular ones is the `forEach()` method. You can also open interfaces like `java.util.Map` to see new default methods e.g. `putIfAbsent()`, which was only available to `ConcurrentMap` prior to JDK 1.8 version.

That's all about the **default methods of Java 8**. This is one of the breakthrough changes, which will open a path for better and more convenient interfaces. The best way to remember the default method is to remember the problem of using the `putIfAbsent()` method of `ConcurrentMap` from JDK 1.7, which was not present in Map.

It was not possible to write methods that can directly operate on the Map interface because any time if a Map interface points to a `ConcurrentMap` object, you need to cast into `ConcurrentMap` just for sake of using the `putIfAbsent()` method.

With extension methods, now JDK 8's `java.util.Map` interface has got its own `putIfAbsent()` method. To learn more about what is new in Java 8, I suggest taking a look at Manning's Java 8 in Action, It is one of the best Java 8 books available in the market right now to guide you through features like lambdas and streams.