HowToDoInJava

Java Default Methods Tutorial

Last Updated: December 26, 2020 By: Lokesh Gupta Java 8 Default Methods, Java 8

In previous post, we learned about Lambda expressions and functional interfaces. Now, let's move on the discussion and talk about another related feature i.e. **default methods**. Well, this is truly revolutionary for java developers. Till java 7, we have learned a lot of things about interfaces and all those things have been in our mind whenever we wrote code or designed the applications. Some of these concepts are going to change drastically from java 8, after introduction of default methods.

I will discuss following points in this post:

What are default methods in java 8?
Why default methods were needed in java 8?
How conflicts are resolved while calling default methods?

What are default methods in java 8?

Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.

As name implies, default methods in java 8 are simply default. If you do not override them, they are the methods which will be invoked by caller classes. They are defined in interfaces.

Let's understand with an example:

```
public interface Moveable {
    default void move(){
        System.out.println("I am moving");
    }
}
```

Moveable interface defines a method move(); and provided a default implementation as well. If any class implements this interface then it need not to implement it's own version of move() method. It can directly call instance.move();

```
public class Animal implements Moveable{
    public static void main(String[] args){
        Animal tiger = new Animal();
        tiger.move();
    }
}

Output: I am moving
```

And if class willingly wants to customize the behavior then it can provide it's own custom implementation and override the method. Now it's own custom method will be called.

```
public class Animal implements Moveable{

    public void move(){
        System.out.println("I am running");
    }

    public static void main(String[] args){
        Animal tiger = new Animal();
        tiger.move();
    }
}

Output: I am running
```

This is not all done here. Best part comes as following benefits:

1. Static default methods: You can define static default methods in interface which will be available to all instances of class which implement this interface. This makes it easier for you to organize helper methods in your libraries; you can keep static methods specific to an interface in the same interface rather than in a separate class. This enables you to define methods out of your class and yet share with all child classes.
2. They provide you an highly desired capability of adding a capability to number of classes without even touching their code. Simply add a default method in interface which they all implement.

Why default methods were needed in java 8?

This is a good candidate for your next interview question. **Simplest answer is to enable the functionality of lambda expression in java.** Lambda expression are essentially of type of functional interface. To support lambda expressions seamlessly, all core classes have to be modified. But these core classes like java.util.List are implemented not only in JDK classes, but also in thousands of client code as well. Any incompatible change in core classes will back fire for sure and will not be accepted at all.

Default methods break this deadlock and allow adding support for functional interface in core classes. Let's see an example. Below is a method which has been added to java.lang.Iterable.

```
default void forEach(Consumer<? super T> action) {
  Objects.requireNonNull(action);
  for (T t : this) {
    action.accept(t);
  }
}
```

Before java 8, if you had to iterate on a java collection then your would get an iterator instance and call it's next method until hasNext() returns false. This is common code and have been used thousands of time in day to day programming by us. Syntax is also always same. So can we make it compact so that it takes only single line of code and still do the job for us as before. Above function does that.

Now to iterate and perform some simple operation on every item in list, all you need to do is:

```
import java.util.ArrayList;
import java.util.List;

public class Animal implements Moveable{
    public static void main(String[] args){
        List<Animal> list = new ArrayList();
        list.add(new Animal());
        list.add(new Animal());
        list.add(new Animal());

        //Iterator code reduced to one line
        list.forEach((Moveable p) -> p.move());
    }
}
```

So here, an additional method has been added to List without breaking any custom implementations of it. It has been very desired feature in java since long. Now it's with us.

How conflicts are resolved while calling default methods?

So far so good. We have got all basics well. Now move to complicated things. In java, a class can implement N number of interface. Additionally, a interface can also extend another interface as well. An if any default method is declared in two such interfaces which are implemented by single class. then obviously class will get confused which method to call.

Rules for this conflict resolution are as follows:

1) Most preferred are the overridden methods in classes. They will be matched and called if found before matching anything.
2) The method with the same signature in the "most specific default-providing interface" is selected. This means if class Animal implements two interfaces i.e. Moveable and Walkable such that Walkable extends Moveable. Then Walkable is here most specific interface and default method will be chosen from here if method signature is matched.

3) If Moveable and Walkable are independent interfaces then a serious conflict condition happen, and compiler will complain then it is unable to decide. The you have to help compiler by providing extra info that from which interface the default method should be called. e.g.

```
Walkable.super.move();
//or
Moveable.super.move();
```

That's all for this topic here. I will more on this next time when something interesting comes into my mind. DO not forget to drop your comments/thoughts or questions.

Happy Learning !!

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Spring boot change default port of embedded server
2. Java hashCode() and equals() Methods
3. Private Methods in Interface – Java 9
4. Immutable Collections with Factory Methods in Java 9
5. Java abstract keyword – abstract classes and methods
6. Hibernate get() vs load() Methods
7. Java WatchService API Tutorial

8. JMS Tutorial – Java Message Service Tutorial

9. Java Exact Arithmetic Operations Support in Math Class

0. Java Stream sorted()

14 thoughts on “Java Default Methods Tutorial”

Naga Parise

September 9, 2019 at 8:54 am

Hi Lokesh Gupta,

Thanks for info.

All your articles are extremely good.

Reply

Nishtha

July 15, 2019 at 4:31 pm

How can we define “Static default methods”.
I'm getting a Compile time error saying “Illegal combination of modifiers for the interface method run; only one of abstract, default, or static permitted”.
Please help

Reply

Shekhar

July 22, 2019 at 12:15 am

Any concrete static method in an interface is “default” by default. So we don’t write ” default static myStaticMethod() “. Just write
static myStaticMethod(){…;} and it will be static default method for that interface.
Its confusing , else they should have made it

Reply

Veerendhar

May 15, 2019 at 5:41 am

Nice useful explanation Lokesh…. Thank you

Reply

MANOHAR GNANAVEL

December 2, 2019 at 10:23 am

Yeas Thank you

Reply

Boitumelo J. Mahlong

March 25, 2019 at 12:52 am

A good explanation on Default methods. I have a question which I will also attempt to answer by playing with a few examples. If default methods are to enable/support functional interface implementations on core classes, does it then mean if I have 5 default methods in my interface, I can implement all through lambda expressions or will it only be the one method not declared "default"

Reply

Noopur

April 10, 2019 at 3:45 pm

lambda expressions are for defining body to abstract method of functional interface. The default methods already have definition in interface itself.

Reply

Shilpa

May 4, 2016 at 6:59 pm

Thank you Lokesh!

Reply

prathap

May 7, 2014 at 9:57 am

hi lokesh,

nice explanation and thanx

Reply

adarsh

April 2, 2014 at 4:04 pm

Does this mark the end of abstract classes.

Reply

Barnali

July 14, 2015 at 5:21 pm

Default method does not mark end of abstract classes.
Interface does not have state/field so default method can not use state like abstract class.

Reply

RAJ

March 31, 2014 at 5:33 am

Nice short narration Thanks.

Reply

Ankush

March 30, 2014 at 4:02 pm

Very nice and informative article. Thank you!

Reply

Saurabh

March 29, 2014 at 11:10 pm

Nice, precise and short article. Thanks

Reply

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial