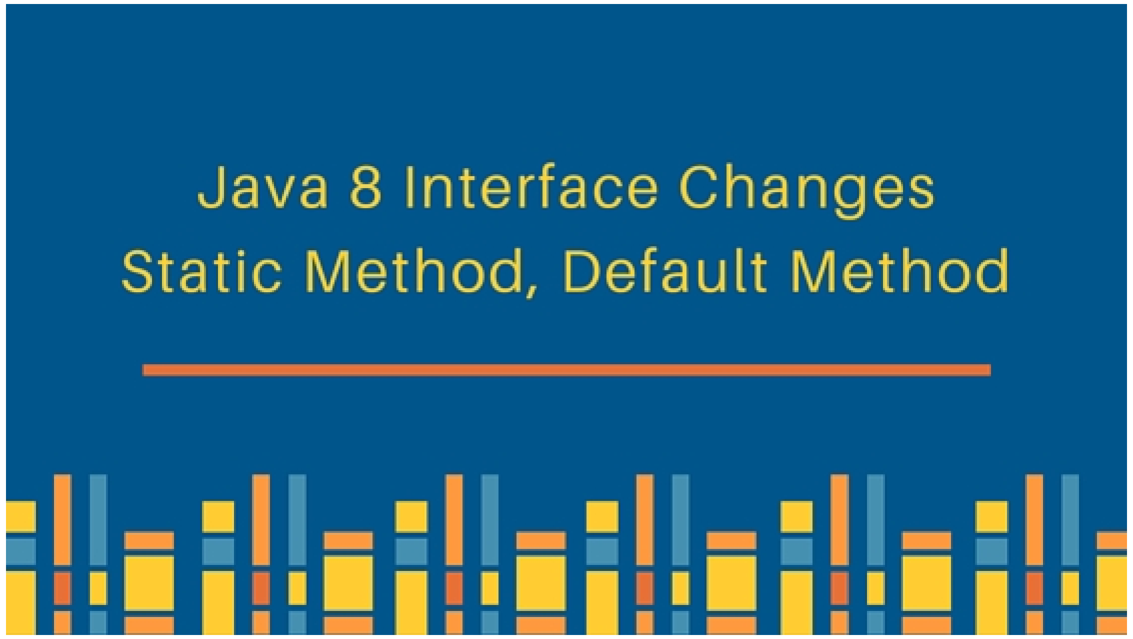
Java 8 Interface Changes – static method, default method

APRIL 2, 2018 BY PANKAJ — 44 COMMENTS

Java 8 interface changes include static methods and default methods in interfaces. Prior to Java 8, we could have only method declarations in the interfaces. But from Java 8, we can have **default methods** and **static methods** in the interfaces.

Table of Contents [hide]



Designing interfaces have always been a tough job because if we want to add additional methods in the interfaces, it will require change in all the implementing classes. As interface grows old, the number of classes implementing it might grow to an extent that it's not possible to extend interfaces. That's why when designing an application, most of the frameworks provide a base implementation class and then we extend it and override methods that are applicable for our application.

Let's look into the default interface methods and static interface methods and the reasoning of their introduction in Java 8 interface changes.

Java Interface Default Method

For creating a default method in java interface, we need to use "**default**" keyword with the method signature. For example,

Java Arrays   Annotation and Enum
Java Collections   Java IO Operations
Java Exception Handling
MultiThreading and Concurrency
Regular Expressions   Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial

```
package com.journaldev.java8.defaultmethod;

public interface Interface1 {

        void method1(String str);

        default void log(String str){
                System.out.println("I1 logging::"+str);
        }
}
```

Notice that log(String str) is the default method in the `Interface1`. Now when a class will implement Interface1, it is not mandatory to provide implementation for default methods of interface. This feature will help us in extending interfaces with additional methods, all we need is to provide a default implementation.

Let's say we have another interface with following methods:

```
package com.journaldev.java8.defaultmethod;

public interface Interface2 {

        void method2();

        default void log(String str){
                System.out.println("I2 logging::"+str);
        }

}
```

We know that Java doesn't allow us to extend multiple classes because it will result in the "Diamond Problem" where compiler can't decide which superclass method to use. With the default methods, the diamond problem would arise for interfaces too. Because if a class is implementing both `Interface1` and `Interface2` and doesn't implement the common default method, compiler can't decide which one to chose.

Extending multiple interfaces are an integral part of Java, you will find it in the core java classes as well as in most of the enterprise application and frameworks. So to make sure, this problem won't occur in interfaces, it's made mandatory to provide implementation for common default methods of interfaces. So if a class is implementing both the above interfaces, it will have to provide implementation for `log()` method otherwise compiler will throw compile time error.

A simple class that is implementing both `Interface1` and `Interface2` will be:

```
package com.journaldev.java8.defaultmethod;

public class MyClass implements Interface1, Interface2 {

        @Override
        public void method2() {
        }

        @Override
        public void method1(String str) {
        }

        @Override
        public void log(String str){
```

```
                    Interface1.print("abc");
            }
    }
```

Important points about java interface default methods:

1. Java interface default methods will help us in extending interfaces without having the fear of breaking implementation classes.
2. Java interface default methods has bridge down the differences between interfaces and abstract classes.
3. Java 8 interface default methods will help us in avoiding utility classes, such as all the Collections class method can be provided in the interfaces itself.
4. Java interface default methods will help us in removing base implementation classes, we can provide default implementation and the implementation classes can chose which one to override.
5. One of the major reason for introducing default methods in interfaces is to enhance the Collections API in Java 8 to support lambda expressions.
6. If any class in the hierarchy has a method with same signature, then default methods become irrelevant. A default method cannot override a method from `java.lang.Object`. The reasoning is very simple, it's because Object is the base class for all the java classes. So even if we have Object class methods defined as default methods in interfaces, it will be useless because Object class method will always be used. That's why to avoid confusion, we can't have default methods that are overriding Object class methods.
7. Java interface default methods are also referred to as Defender Methods or Virtual extension methods.

Java Interface Static Method

Java interface static method is similar to default method except that we can't override them in the in implementation classes. Let's look into this with a simple example.

```
package com.journaldev.java8.staticmethod;

public interface MyData {

	default void print(String str) {
		if (!isNull(str))
			System.out.println("MyData Print::" + str);
	}

	static boolean isNull(String str) {
		System.out.println("Interface Null Check");

		return str == null ? true : "".equals(str) ? true : false;
	}
}
```

Now let's see an implementation class that is having isNull() method with poor implementation.

```
package com.journaldev.java8.staticmethod;

public class MyDataImpl implements MyData {

	public boolean isNull(String str) {
		System.out.println("Impl Null Check");

		return str == null ? true : false;
```

```
        public static void main(String args[]){
                MyDataImpl obj = new MyDataImpl();
                obj.print("");
                obj.isNull("abc");
        }
}
```

Note that `isNull(String str)` is a simple class method, it's not overriding the interface method. For example, if we will add @Override annotation to the isNull() method, it will result in compiler error.

Now when we will run the application, we get following output.

```
Interface Null Check
Impl Null Check
```

If we make the interface method from static to default, we will get following output.

```
Impl Null Check
MyData Print::
Impl Null Check
```

Java interface static method is visible to interface methods only, if we remove the isNull() method from the `MyDataImpl` class, we won't be able to use it for the `MyDataImpl` object. However like other static methods, we can use interface static methods using class name. For example, a valid statement will be:

Important points about java interface static method:

1. Java interface static method is part of interface, we can't use it for implementation class objects.
2. Java interface static methods are good for providing utility methods, for example null check, collection sorting etc.
3. Java interface static method helps us in providing security by not allowing implementation classes to override them.
4. We can't define interface static method for Object class methods, we will get compiler error as "This static method cannot hide the instance method from Object". This is because it's not allowed in java, since Object is the base class for all the classes and we can't have one class level static method and another instance method with same signature.
5. We can use java interface static methods to remove utility classes such as Collections and move all of it's static methods to the corresponding interface, that would be easy to find and use.

Java Functional Interfaces

Before I conclude the post, I would like to provide a brief introduction to Functional interfaces. An interface with exactly one abstract method is known as Functional Interface.

A new annotation @FunctionalInterface has been introduced to mark an interface as Functional Interface. @FunctionalInterface annotation is a facility to avoid accidental addition of abstract methods in the functional interfaces. It's optional but good practice to use it.

Functional interfaces are long awaited and much sought out feature of Java 8 because it enables us to use **lambda expressions** to instantiate them. A new package `java.util.function` with bunch of functional interfaces are added to provide target types for lambda expressions and method references. We will look into functional interfaces and lambda expressions in the future posts

FILED UNDER: JAVA

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

« JDBC Tutorial Java 8 Functional Interfaces »

Comments

sharan says

MAY 23, 2018 AT 1:00 AM

Strange !! but I can write main method in interface and execute it without writing main method in a class.

I would like know about this behavior

Reply

Varsha Beedkar says

MARCH 20, 2018 AT 12:50 AM

great explanation

sensibles says

MARCH 9, 2018 AT 5:09 AM

awesome article, thanks man!

Reply

Paru says

JANUARY 20, 2018 AT 9:47 PM

Interface1.print("abc"); there is no such method in Interface1.

What is the main use of default method in interface? how can we access or make use of it?

Reply

Anil Gupta says

JANUARY 18, 2018 AT 12:02 AM

Very well explained. Great job. Thanks very much for sharing.

Reply

priya says

NOVEMBER 21, 2017 AT 9:54 PM

static is a keyword which makes any variable or method a class level variable or class level method ,means we can access it by our class name without making object of class. Thanks for sharing this.

Reply

Rajkumar says

JUNE 30, 2017 AT 9:39 AM

Can we create a static method in an interface and can i call class object with it's static method.

Reply

Niteesh Bhargava says

MAY 7, 2017 AT 6:24 AM

Hi Pankaj,

Thanks again for useful content.

Reply

Sumit says

SEPTEMBER 1, 2017 AT 9:31 PM

Dear

nitish , it is good to see

Dear

pankaj thanks again

Reply

Aruna says

APRIL 23, 2017 AT 1:25 PM

Hi Pankaj,

Can you please explain the below conflict.

Journaldev says

"Designing interfaces have always been a tough job because if we want to add additional methods in the interfaces, it will require change in all the implementing classes. As interface grows old, the number of classes implementing it might grow to an extent that it's not possible to extend interfaces. That's why when designing an application, most of the frameworks provide a base implementation class and then we extend it and override methods that are applicable for our application."

But

"By adding a default method to an interface, every implementing class automatically inherits this behavior. Some of these classes might have not been designed with that new functionality in mind, and this can cause problems. For instance, if someone adds a new default method default void foo() to an interface Ix, then the class Cx implementing Ix and having a private foo method with the same signature does not compile."

As per this again any changes to interfaces will effect the implementation classes right?

thanks in advance,

Aruna.

Reply

Ankit says

JANUARY 29, 2018 AT 4:35 AM

Reply

Patan says

APRIL 28, 2018 AT 10:50 AM

It won't compile as it can result into reducing the visibility of the method, which can result into the compile time error.

Reply

Aruna says

APRIL 23, 2017 AT 12:14 PM

Hi Pankaj,
Is it mandatory for the implementation classes to use the default methods
(or)
is it optional to use default methods.
I mean does the rule of abstract methods of interface apply to default methods making it mandatory for the implementation classes to have default methods in their code.
Thankyou.

Reply

Bhanu says

Its not mandatory as the interface is already providing a default definition and that's the sole purpose.
Reply

Aruna says
APRIL 23, 2017 AT 12:02 PM
Hi Pankaj,
thank you so much for the tutorials.
And I couldn't under stand the 6th point from default methods.
"If any class in the hierarchy has a method with same signature, then default methods become irrelevant. A default method cannot override a method from java.lang.Object. The reasoning is very simple, it's because Object is the base class for all the java classes. So even if we have Object class methods defined as default methods in interfaces, it will be useless because Object class method will always be used. That's why to avoid confusion, we can't have default methods that are overriding Object class methods."
So what exactly does this explain?
does it say that "if any class in the hierarchy has a method with same signature as of the default method of base interface."
what does the compiler use:
a) the implementing class method
(or)
b) the default method of the base interface?.
(or)
c)will the compiler throw an error

Thanks,
Aruna.
Reply

Niket says
FEBRUARY 2, 2018 AT 8:24 AM
It means if we have a method say void foo() in interface X and void foo() in Object class and a class Y implements X then using foo() method in Y will inherit it's features from Object and not from X
Reply

Tapan Singh says
MARCH 26, 2018 AT 9:15 PM
Its not that complicated. Take your example.
Interface I has default method foo. Now Class X implements Interface I and provides implementation for foo, overriding essentially. Then Class Y extends Class X. So, we now have I –> X –> Y. If Y uses method foo, implementation of X is invoked; thereby rendering the default implementation in Interface I useless.
Clear so far, with me?
Now, second point.
If Interface I provides default implementation of equals (a method in Object class); and now Class X implements Interface I, whose equals method will it use: Interface I's or Object class'?
Remember that Interface I and Object class have no connection. To avoid this confusion, the Java language architects added this check in the compiler to not allow default implementation of methods provided in the Object class already.
Hope that clarifies.

sumit says

MARCH 22, 2017 AT 12:36 AM

@Override
public void log(String str){
System.out.println("MyClass logging::"+str);
Interface1.print("abc");
}
it should be Interface1.super.log("abc");

Reply

sumit says

MARCH 22, 2017 AT 12:01 AM

@Override
public void log(String str){
System.out.println("MyClass logging::"+str);
Interface1.print("abc");
}
it should be Interface1.log("abc");

Reply

NJ says

MARCH 4, 2018 AT 7:15 AM

It must be Interface1.super.log("abc") instead of Interface1.log("abc")

Reply

udit sharma says

SEPTEMBER 20, 2016 AT 11:28 AM

I didn't got the point number 4 for static method points :
We can't define interface static method for Object class methods, we will get compiler error as "This static method cannot hide the instance method from Object".

```
public interface Interface {
default void play(){
System.out.println("this is play method");
}
static boolean equals() {
System.out.println("Interface Null Check");
return true;
}
}
```

Reply

static boolean equals() is not object equals method
try below:
static boolean equals(Object obj)
Reply

IRS says
JUNE 9, 2016 AT 1:27 AM
Hi Pankaj – Thanks for such a concise and self-explanatory blogpost.
Quick question – I am trying to understand why is it that Functional Interfaces can have ONLY one method ?
Why not overloaded methods ? For example –
operate(int a, int b)
operate(float a, float b)
Reply

Pankaj says
JUNE 9, 2016 AT 6:40 AM
I guess that is to keep things simple because method names are not used while calling them.
Reply

OCTOBER 25, 2016 AT 11:07 AM

The underlying answer to your question is the fact that Functions are not [yet] “First-class Citizens” in Java. This essentially means that while lambdas provide a nice, convenient syntax, there are still being wrapped within a interface under the hood [in other words, lambdas are backward compatible with these interfaces]. And because of this, Functional Interfaces can have only one method. Hope that helps.

Reply

Alok says

FEBRUARY 28, 2017 AT 1:28 AM

Functional interfaces can have only one ‘Abstract method’. Other methods like default or static may or may not exist.

Reply

Paula says

JUNE 7, 2016 AT 5:50 AM

hi,

indeed i would also like to know what is with that print from line 9, Interface1, default method.

Thanks in advance.

regards

Reply

Pankaj says
JUNE 11, 2016 AT 10:47 PM
Nice catch, i have removed that code and updated the post.
Reply

JJ says
MAY 23, 2016 AT 1:50 AM
Excellent explanation and useful replies. Thanks Pankaj for your blog entries!
I suggest we replace line 8 of the MyDataImpl class to the shorter version shown below. It is more concise and common usage, while providing the same result as the current version. Thanks!
return str == null;
Reply

Pankaj says
JUNE 11, 2016 AT 10:49 PM
yes make sense, but to convey the idea easily I think it's better to have ternary operator. It's more like developers choice of using the code. Thanks!
Reply

SEPTEMBER 24, 2015 AT 12:16 PM

Could you please explain how this got printed if we remove static from the interface method
MyData Print::

Reply

Paul Ostrowski says

OCTOBER 26, 2015 AT 9:40 AM

When he removed the static qualifier, then the implementation class overrode the now non-static method isNull, so when he called print(""), the interface called the implementation's version of isNull, which returned 'false', allowing the print to occur.

Reply

Pankaj says

JUNE 11, 2016 AT 10:50 PM

Thanks Paul for the explanation.

Reply

DRC says

JUNE 23, 2015 AT 6:05 AM

Very good explanation indeed ! !

One very minor observation : In the class Interface1, line 9 -> is the print(str) a typo ?

Reply

Pankaj **says**

JUNE 11, 2016 AT 10:50 PM

yes it was typo, i have removed it and updated the post.

Reply

Anshudeep **says**

MAY 4, 2015 AT 9:31 AM

With interface default methods there may also be a scenario where an interface is extending another interface and both of those interfaces have same default method.

Reply

Pankaj **says**

JUNE 11, 2016 AT 10:51 PM

It will be the case of overriding super class method.

Reply

APRIL 8, 2015 AT 10:08 PM

Very nice explanation…Thanks..

Reply

Rafael Chaves says

NOVEMBER 29, 2014 AT 6:19 AM

> We can use static interface methods to remove utility classes such as Collections and move all of it's static methods to the corresponding interface, that would be easy to find and use.

I like the new static interface methods feature, but I don't think what you suggested here is a good idea – an interface is meant to be implemented, to expand the protocol of an object, whereas utility classes such as Collections are just containers for utility functions, which is a bit of a hack, as Java does not provide support for standalone functions.
I think classes are a better choice. Given the newfound power of interfaces, we should leave them for what they are meant for, to define partial protocols (and implementations) for objects in an object-oriented program, and continue to use classes as a pattern for holding non-OO functionality.
is that better than using classes?

Reply

Rafael Chaves says

NOVEMBER 29, 2014 AT 6:21 AM

Oh, I see now, you are talking about helper classes that were creared to complement an interface. In that case, I do see merit in your suggestion.

Pankaj says
JUNE 11, 2016 AT 10:52 PM
Thanks for the comment and understanding. 🙂
Reply

Ranjith says
APRIL 24, 2014 AT 4:04 PM
Superb explanation , we are awaited for next post .
Reply

Dilip Kumar Pandey says
APRIL 21, 2014 AT 6:21 PM
great tutorials
cleatly explains the concept without deviating
from the topic.
Awaiting for your future posts on
functional interfaces and lambda expressions.
Reply

Pankaj says

JUNE 11, 2016 AT 10:53 PM

it's already posted, search for it and you will find it. 🙂

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

POST COMMENT