Do interfaces inherit from Object class in java

Ask Question

Do interfaces inherit from `Object` class in Java?

If no then how we are able to call the method of object class on interface instance

```
public class Test {
    public static void main(String[] args) {
        Employee e = null;
        e.equals(null);
    }
}

interface Employee {
}
```

java inheritance interface

edited Jan 4 '13 at 13:29

aioobe

308k 71 669 726

asked May 19 '11 at 8:58

ponds

779 4 9 11

44 +1, Excellent question. – aioobe Jun 3 '11 at 12:19

@EJP, technically speaking it doesn't matter what java/io/Serializable.class contains. I think you're confusing the Java Lang Spec with the JVM spec. – aioobe Jun 20 '12 at 8:13

@aioobe As I haven't mentioned either of those specifications I don't understand your point. `Serializable` is an interface, the simplest possible; running `javap` on it tells you what it inherits from; and that is dictated by the Java Language Specification. If you think the JVM Spec comes into it somewhere please enlighten us. – EJP Jun 21 '12 at 8:49

2 @EJP, the question is about the Java language (i.e. the Java Language Specification). What ever java/io/Serializable.class contains is related to what the JVM spec says. Technically speaking there is no guarantee that there is a one-to-one correspondence between features of the two specifications. – aioobe Jan 4 '13 at 13:29

I elaborated on this in a recent blog post. – aioobe Sep 16 '16 at 11:28

7 Answers

> ***Do interfaces inherit from `Object` class in Java?***

No, they don't. And there is no common "root" interface implicitly inherited by all interfaces either (as in the case with classes) for that matter.(*)

> ***If no then how we are able to call the method of object class on interface instance***

An interface implicitly declared one method for each public method in `Object`. Thus the `equals` method is implicitly declared as a member in an interface (unless it already inherits it from a superinterface).

This is explained in detail in the Java Language Specification, § 9.2 Interface Members.

> ### 9.2 Interface Members
>
> [...]
>
> - If an interface has no direct superinterfaces, **then the interface implicitly declares a public abstract member method *m* with signature *s*, return type *r*, and throws clause *t* corresponding to each public instance method m with signature *s*, return type *r*, and throws clause *t* declared in `Object`**, unless a method with the same signature, same return type, and a compatible throws clause is explicitly declared by the interface.
>
> [...]

This post has been rewritten as an article here.

(*) Note that the notion of being a *subtype of* is not equivalent to *inherits from*: Interfaces with no super interface are indeed subtypes of `Object` (§ 4.10.2. Subtyping among Class and Interface Types) even though they do not inherit from `Object`.

edited Oct 30 '16 at 14:10

answered Jun 3 '11 at 12:18

aioobe

308k 71 669 726

6 reading it even in 2013, this answer should have got way more upvotes. Good stuff – happybuddha May 23 '13 at 20:31

1 @aioobe If we implement any interface, then why don't we give the implementation of "equals" method in the class which is implementing that interface. According to my concepts,we have to implement the methods of interface in implementing class otherwise the class will be abstract. – Vikas Mangal Jul 2 '14 at 7:38

1 You don't need to (re)implement inherited methods. Have a look at this example. In other words, equals is already defined and inherited to the class implementing the interface. – aioobe Jul 2 '14 at 8:34

3 I got the point here. But one question- why do we need this? What difference it would have made if the methods of `Object` class would not have been declared in the interface ? – Vikas Mangal Dec 11 '14 at 18:07

2 If we didn't have this, the program in the question would not compile. There is on `equals` method in the `Employee` interface. – aioobe Dec 11 '14 at 18:44

There is actually a superclass field in every `.class` file, including those that represent interfaces.

For an interface it always points to `java.lang.Object`. But that isn't used for anything.

Another way to look at it is:

```
interface MyInterface {
    // ...
}

public myMethod(MyInterface param) {
    Object obj = (Object) param;
    // ...
}
```

Here the cast `(Object) param` is always valid, which implies that every interface type is a subtype of `java.lang.Object`.

answered May 19 '11 at 9:07

finnw

37k 15 122 194

1 The .class file is an artifact of the .java file. To argue why something works in Java language by looking at the resulting .class file is backward reasoning. – aioobe Dec 9 '14 at 17:12

`Object` is a supertype of any interface [1]

However, an interface does not `implements` , `extends` , or, *"inherit from"* `Object` .

JLS has a special clause to add `Object` methods into interfaces [2]

[1] http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.10.2

[2] http://java.sun.com/docs/books/jls/third_edition/html/interfaces.html#9.2

answered May 19 '11 at 10:24

irreputable

37.8k 6 52 81

That's because `employee e = ...` reads that there is a class that **implements** `employee` , and is assigned to variable `e` . Every class that implements an interface extends Object implicitly, hence when you do `e.equals(null)` , the language knows that you have a class that is a subtype of `employee` .

The JVM will do runtime checking for your code (i.e. throw `NullPointerException`).

answered May 19 '11 at 9:16

Buhake Sindi

67.7k 22 142 201

Is intreface inherits Object class, how can we able to access the methods of object class through a interface type reference

No Interface does not inherits `Object` class,but it provide accessibility to all methods of `Object` class. The members of an interface are:

```
Those members declared in the interface.
Those members inherited from direct superinterfaces.
If an interface has no direct superinterfaces, then the interface implicitly
```

declares a public abstract member method corresponding to each public instance method declared in `Object` class, .
It is a compile-time error if the interface explicitly declares such a method m in the case where m is declared to be `final` in `Object` .

Now it is clear that all superinterface have `abstract` member method corresponding to each `public` instance method declared in `Object` .
source: http://ohmjavaclasses.blogspot.com/2011/11/is-intreface-inherits-object-clashow.html

edited Dec 12 '17 at 9:54
Ashish Kumar
477 2 7 23

answered Dec 10 '11 at 11:28

Sheo
675 9 21

Any class implementing any interface is also derived from `Object` as well by definition.

answered May 19 '11 at 9:05

jabal
5,927 7 34 79

"**Reference types all inherit from java.lang.Object**. Classes, enums, arrays, and **interfaces are all reference types.**"

Quoted from: http://docs.oracle.com/javase/tutorial/reflect/class/index.html Second sentence to be clear.

answered Dec 22 '12 at 20:11

dalvarezmartinez1
803 1 12 23

`Classes, enums, and arrays (which all inherit from java.lang.Object) as well as interfaces are`

`all reference types` : it does not say interface inherits from Object. Only Classes , enums and arrays. – Breaking Benjamin Apr 6 at 7:30

They changed it :) – dalvarezmartinez1 Apr 11 at 9:41