Does an interface by default extend Object?

Ask Question

If you define a interface like below

```
interface I1{

}
```

The in any code section you can write like

```
I1 i1;
i1.equals(null);
```

Then from where the equals method come, is the interface also extends the super class Object ?, if that true how an interface can extends a class?

Suppose let the interface extends the super class Object , then if you see why the collection interface like Set thave define the equals() and hashCode() method ?. All the class extends the Object class so if you define any abstract method in a interface which present in Object class then who implement the interface they no need to implements those method. Like in below code

```
interface I1{
String toString();
}

class A implements I1{

}
```

Here the class A no need to implements the method toString() as it's present in Object class. Then what is the objective of defining those method in collection interface as they can't force there implementation class to implement those method.

java

edited Jul 12 '14 at 7:51
tereško

asked Dec 8 '12 at 9:02
Krushna

50.8k 18 75 134   **1,251** 2 15 37

Possible duplicate of Do interfaces inherit from Object class in java – proudandhonour Jan 24 '17 at 5:02

2 Answers

> Then from where the equals method come, is the interface also extends the super class Object ?, if that true how an interface can extends a class?

The Java Language Specification deals with this explicitly.

From section 9.2:

> 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.

Basically, this is so that you *can* use `equals` , `hashCode` etc - because the way that the Java language is specified means that any concrete implementation of the interface *will* be a class, and that class *must* ultimately be a subclass of `Object` , so the members will definitely be present.

To put it another way, while the interface itself doesn't extend `Object` , it is known that any implementation will.

> Here the class A no need to implements the method toString() as it's present in Object class. Then what is the objective of defining those method in collection interface as they can't force there implementation class to implement those method.

Usually this is just done for clarity, e.g. to document what is expected of an implementation in terms of the members declared in `Object` .

answered Dec 8 '12 at 9:04

Jon Skeet

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

TEAMS

Create Team

1035k 639 7656 8251

Thank you very much So we can say that interfcae do not extend the object class, What is the answer of my second questions about Set interface why they have define those method. – Krushna Dec 8 '12 at 9:11

@KrushnaCh.Dash: See the end of my answer. The documentation for `Set.hashCode` and `Set.equals` provide more details about the expected behaviour. – Jon Skeet Dec 8 '12 at 9:16

Every class implicitly extends `Object` and so inherits every (non-private) method of the Object class.

Every instance has a class and therefore has all of the method of Object.

Whether an instance implements an interface or not is completely irrelevant to this point.

Object is a class, and interfaces can not extend classes, so "no" - the interface doesn't inherit anything from any class.

edited Dec 8 '12 at 9:41

answered Dec 8 '12 at 9:05

Bohemian ♦

278k 57 384 517

Well, OP is talking about interfaces. – Rohit Jain Dec 8 '12 at 9:07

@RohitJain better now? I thought what I was dayin – Bohemian ♦ Dec 8 '12 at 9:12