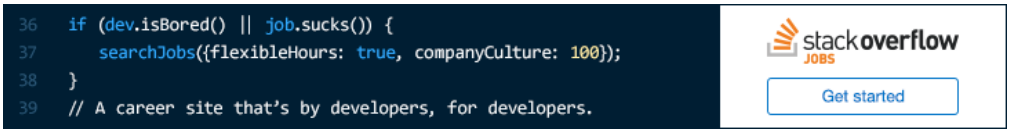
# Constructor in an Interface?




I know it's not possible to define a constructor in an interface. But I'm wondering why, because I think it could be very useful.

So you could be sure that some fields in a class are defined for every implementation of this interface.

For example consider the following message class:

```
public class MyMessage {

   public MyMessage(String receiver) {
      this.receiver = receiver;
   }

   private String receiver;

   public void send() {
      //some implementation for sending the mssage to the receiver
   }
}
```

If a define an interface for this class so that I can have more classes which implement the message interface, I can only define the send method and not the constructor. So how can I ensure that every implementation of this class really has an receiver set? If I use a method like `setReceiver(String receiver)` I can't be sure that this method is really called. In the constructor I could ensure it.

java interface

edited Feb 10 '14 at 19:10
daniel kullmann
5,097 3 31 46

asked May 10 '10 at 15:41
anon

3 possible duplicate of Why are we not allowed to specify a constructor in an interface? – matt b May 10 '10 at 15:42

2 You say "In the constructor I could ensure [every implementation of this class really has an receiver set]." But no, you couldn't possibly do that. Provided it was possible to define such a constructor, the parameter would only be a strong hint to your implementors – but they could chose to simply ignore it if they wanted to. – Julien Silland May 11 '10 at 3:08

@mattb Umm, that's a different language. – yesennes Jan 23 '16 at 20:21

## 7 Answers

Taking some of the things you have described:

> "So you could be sure that some fields in a class are defined for every implementation of this interface."
>
> "If a define a Interface for this class so that I can have more classes which implement the message interface, I can only define the send method and not the constructor"

...these requirements are exactly what abstract classes are for.

edited Oct 8 '15 at 13:41
zb226
4,570 1 20 45

answered May 10 '10 at 15:44
matt b
96.4k 46 224 294

but note that the use case @Sebi describes (calling overloaded methods from parent constructors) is a bad idea as explained in my answer. – rsp May 10 '10 at 15:57

24 Matt, that's clearly true, but abstract classes suffer from the single-inheritance limitation, which leads people to look at other ways of specifying hierarchies. – CPerkins May 10 '10 at 16:09

5 This is true and may solve Sebi's immediate problem. But one reason for using interfaces in Java is because you cannot have multiple inheritance. In a case where I cannot make my "thing" an abstract class because I need to inherit from something else, the problem remains. Not that I claim to have a solution. – Jay May 10 '10 at 16:10

5 @CPerkins while this is true, I am not suggesting that simply using an abstract class will solve Sebi's use case. If anything, it's best to declare a `Message` interface which defines the `send()` method, and if Sebi wishes to provide a "base" class for implementations of the `Message` interface, then provide an `AbstractMessage` as well. Abstract classes shouldn't take the place of interfaces, was never attempting to suggest so. – matt b May 10 '10 at 16:17

1 Understood, Matt. I wasn't arguing with you, more pointing out that it's not a *complete* replacement for what the op wants. – CPerkins May 10 '10 at 20:04



A problem that you get when you allow constructors in interfaces comes from the possibility to implement several interfaces at the same time. When a class implements several interfaces that define different constructors, the class would have to implement several constructors, each one satisfying only one interface, but not the others. It will be impossible to construct an object that calls each of these constructors.

Or in code:

```
interface Named { Named(String name); }
interface HasList { HasList(List list); }

class A implements Named, HasList {

  /** implements Named constructor.
   * This constructor should not be used from outside,
   * because List parameter is missing
   */
  public A(String name)  {
    ...
  }

  /** implements HasList constructor.
   * This constructor should not be used from outside,
   * because String parameter is missing
   */
  public A(List list) {
    ...
  }

  /** This is the constructor that we would actually
   * need to satisfy both interfaces at the same time
   */
  public A(String name, List list) {
    this(name);
    // the next line is illegal; you can only call one other super constructor
    this(list);
  }
}
```

edited Mar 19 '13 at 9:16

answered Mar 8 '12 at 9:21
daniel kullmann
5,097 3 31 46

Couldn't the language have done it by allowing things like `class A implements Named, HashList { A() {HashList(new list()); Named("name");} }` – mako Sep 10 '13 at 23:19

1 The most useful meaning for a "constructor in an interface", if allowed, would be if `new Set<Fnord>()` could be interpreted to mean "Give me something I can use as a `Set<Fnord>` "; if the author of `Set<T>` intended `HashSet<T>` to be the go-to implementation for things that didn't have a particular need for something else, the interface could then define `new Set<Fnord>()` could be considered synonymous with `new HashSet<Fnord>()` . For a class to implement multiple interfaces would not pose any problem, since

`new InterfaceName()` would simply construct a class *designated by the interface*. – supercat Feb 10 '14 at 23:45

---

An interface defines a contract for an API, that is a set of methods that both implementer and user of the API agree upon. An interface does not have an instanced implementation, hence no constructor.

The use case you describe is akin to an abstract class in which the constructor calls a method of an abstract method which is implemented in an child class.

The inherent problem here is that while the base constructor is being executed, the child object is not constructed yet, and therfore in an unpredictable state.

To summarize: is it asking for trouble when you call overloaded methods from parent constructors, to quote mindprod:

> In general you must avoid calling any non-final methods in a constructor. The problem is that instance initialisers / variable initialisation in the derived class is performed **after** the constructor of the base class.

answered May 10 '10 at 15:50

rsp
**17.9k** 4 38 52

---

There is only static fields in interface that dosen't need to initialized during object creation in subclass and the method of interface has to provide actual implementation in subclass .So there is no need of constructor in interface.

Second reason-during the object creation of subclass, the parent constructor is called .But if there will be more than one interface implemented then a conflict will occur during call of interface constructor as to which interface's constructor will call first

answered Nov 30 '15 at 9:25

Satyajit Gami
**51** 1

---

Dependencies that are not referenced in an interfaces methods should be regarded as implementation details, not something that the interface enforces. Of course there can be exceptions, but as a rule, you should define your interface as what the behavior is expected to be. Internal state of a given implementation shouldn't be a design concern of the interface.

answered May 10 '10 at 15:44

Yishai
**59.9k** 16 142 223

---

See this question for the *why* (taken from the comments).

If you really need to do something like this, you may want an abstract base class rather than an interface.

answered May 10 '10 at 15:45

JSBձոգ
**29.7k** 12 81 135

---

This is because interfaces do not allow to define the method body in it.but we should have to define the constructor in the same class as interfaces have by default abstract modifier for all the methods to define. That's why we can not define constructor in the interfaces.

edited Jul 12 '13 at 5:13

NoNaMe
**3,688** 17 48 81

answered Jul 12 '13 at 4:55

Aasif Ali
**11** 3

---