Ruby trunk - Feature #5448

Singleton module's ::instance method should forward parameters

10/14/2011 11:30 PM - Quintus (Marvin Gülker)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	2.6

Description

Hi there,

Classes mixing in the Singleton module currently aren't allowed to have parameters for their initialize method. This should be changed, because sometimes it's necessary to give some initial state information to the single(ton) instance.

Example code (no way to create the instance):

```
require "singleton"

class Foo
include Singleton

def initialize(arg)
puts "arg is: #{arg}"
end

end

f = Foo.instance(1) #=> ArgumentError 1 for 0
```

f = Foo.instance #=> ArgumentError 0 for 1

The arguments given to the ::instance method should be forwarded to #initialize, which in turn may raise the appropriate ArgumentError if necessary.

ruby -v: ruby 1.9.2p290 (2011-07-09 revision 32553) [x86_64-linux]
OS: Arch Linux

Marvin

History

#1 - 10/14/2011 11:54 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Feedback

```
=begin
Singleton has only one instance, why do you need parameters?

What's wrong with:

class FooSingleton < Foo
include Singleton
def initialize
super(1)
end
end
=end
```

#2 - 10/16/2011 12:24 AM - Quintus (Marvin Gülker)

Nobuyoshi Nakada wrote:

> Singleton has only one instance, why do you need parameters?

Maybe you're right, because one could do it with accessors as well. However, beside that a Singleton can just have one instance, there's nothing special about them, therefore I personally think that there shouldn't be anything special about it's #initialize method as well. "Normal" classes can have parameters for #initialize which are respected by the ::new method of the respecitive class. For symmetry I think it should work the same way with the ::instance method.

Plus: It causes a hard to understand exception message. As I've shown in the initial example, an #initialize with parameters renders the class unusable in "normal" terms. You cannot instanciate it, not even the single instance allowed by the Singleton module. The exception message, "wrong number of arguments (1 for 0) (ArgumentError)", caused by the code that one expected to forward it's arguments to #initialize makes one think that there's something wrong with the call, but comparing the call to ::instance and the parameters #initialize takes, there's nothing wrong. Then, on the second glance, one realizes that the exception is actually caused from *within* Ruby's standard library and the ArgumentError is not directly a result of instanciating the Foo class, but rather the result of Singleton's incapability of forwarding the arguments to the #initialize method.

> What's wrong with:
>
> class FooSingleton < Foo
> include Singleton
> def initialize
> super(1)
> end
> end

It adds an extra layer of complexity where none is needed. The Foo class itself is unusable, and just to instanciate it I have to make a subclass of it which in turn I can instanciate then. Subclassing is not meant to make classes instanciatable and I didn't define an abstract class (Ruby doesn't need things like that)--and this subclass isn't a subclass in the logical "kind-of" way. It's just a helper construct to circumvent a problem. As stated earlier, I would use an accessor if the situation stays like it is now, because that appears a bit cleaner to me. I happen to think that classes mixing in the Singleton module shouldn't be more special than normal classes beside their "singletonness".

Vale,
Marvin

#3 - 11/09/2012 06:12 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#4 - 11/20/2012 11:48 AM - nobu (Nobuyoshi Nakada)

- Status changed from Feedback to Rejected

I think that "an extra layer of complexity where none is needed" is the parameter here.
If it were really needed, you may want to use multiton instead.
What you want doesn't sound singleton.