Ruby - Feature #12334

Final/Readonly Support for Fields / Instance Variables

04/30/2016 06:28 PM - wied03 (Brady Wied)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

This sort of relates to https://bugs.ruby-lang.org/issues/11911

C# through 'readonly' and Java through 'final' variables/fields allow me to only allow assigning a field in the initializer. It might be nice to embrace some controlled mutation by having this feature in Ruby. Sometimes its tempting in other methods to reassign a field but you really want to control that from the initializer.

Freezing targets a different problem by controlling what I can mutate within that field's object. The two can compliment each other but I see them as each solving a different problem. I know it's possible to freeze an entire instance of a class and not allow reassigning the field that way, but then I'm in an all or none situation where I can't have controlled mutation.

Related issues:

Related to Ruby - Feature #12021: Final instance variables	**Open**

History

#1 - 05/01/2016 04:29 PM - shevegen (Robert A. Heiler)

What is a "field"?

#2 - 05/01/2016 04:38 PM - wied03 (Brady Wied)

- Subject changed from Final/Readonly Support for Fields to Final/Readonly Support for Fields / Instance Variables

Sorry, I should have used correct terminology. By fields, I meant instance variables.

#3 - 05/02/2016 02:06 AM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Rejected

In ruby, instance variables are private to objects. You can *never* access instance variables from outside of an object, unless explicitly exported via a method.

Mutation of instance variables are already controlled: you cant.

#4 - 05/02/2016 02:15 AM - wied03 (Brady Wied)

This feature request is not about being external to the object.

I can have multiple Ruby methods in the same class besides the initialize method that all do @some_ivar = 123 or @some_ivar=456. This is about trying to guard against that (if the developer chooses).

#5 - 05/02/2016 02:20 AM - wied03 (Brady Wied)

Here is an example:

```
class Foo
  attr_reader :foo

  def initialize
    @foo = 123
  end

  def accidental_mutate
    @foo = 456 # Ruby doesn't give me any way to prevent this as of now
  end
end

class Foo
```

```
  attr_readonly :foo # one way, see initializer

  def initialize
    # another way
    readonly @foo = 123
  end

  def accidental_mutate
    @foo = 456 # Ruby could throw an error here and prevent me from doing this
  end
end
```

#6 - 05/02/2016 02:21 AM - wied03 (Brady Wied)

If the feature should be rejected, I can live with that, but I don't think we had the same idea of what I was requesting here.

#7 - 05/02/2016 04:06 AM - shyouhei (Shyouhei Urabe)

- Status changed from Rejected to Open

OK, I see your proposal now. I'm not for it though ("you write a class and you can't control when to mutate its instance variables" rather sounds like a design issue to me).

#8 - 05/02/2016 05:29 AM - duerst (Martin Dürst)

I agree with Shyouhei. If you are afraid of changing an instance variable in one of your classes' methods, maybe just name it @final_foo to remind you that you didn't want to change it.

There are many other checks, too, that could be introduced into Ruby, but the Ruby way is to leave it to the programmer.

#9 - 05/02/2016 02:51 PM - wied03 (Brady Wied)

Maybe leaving it to the developer or design/code reviewing is the best way. I do think there is something to the idea of controlled mutation though. It's a good middle ground between what the functional language people are pushing for (total immutability everywhere) and a more wide open setup.

#10 - 05/02/2016 02:54 PM - wied03 (Brady Wied)

If the developer had to take care of all of this, we probably would not have freezing right? You could make the case that it should be up to the developer to not mutate a frozen object. I'm not sure what the exact thought process was behind implementing frozen in Ruby, but perhaps it was similar.

#11 - 05/02/2016 03:50 PM - sawa (Tsuyoshi Sawada)

It has nothing to do with freezing. Freezing is about a property of an object. What you are trying to do is about variable assignment.

I think what you want is a constant that belongs to an instance and can be defined in a method body (and further raises an error instead of warning when redefinition is attempted). But that does not make sense to me. It means that you can only call that method once. And it would bring many other impractical restrictions.

#12 - 05/02/2016 03:56 PM - wied03 (Brady Wied)

I know that it has nothing to do with freezing. I was just using that as an example of something where Ruby has decided to give the developer a tool to control mutation.

@Tsuyoshi - Yes a constant like that is sort of what I'm describing. That method though is initialize, which by definition is only called once.

#13 - 05/03/2016 09:13 AM - Eregon (Benoit Daloze)

- Related to Feature #12021: Final instance variables added

#14 - 05/19/2016 10:59 PM - dsferreira (Daniel Ferreira)

I'm struggling to understand the use cases.

For instance, can't we achieve the same thing by using a private method or a constant? Why do we need to use a readonly instance variable?

#15 - 05/19/2016 11:04 PM - wied03 (Brady Wied)

Daniel: It might be easiest to go read about how readonly works in C#.

This is quite different than a constant because it can very with each instance of an object since a "readonly ivar" is set in the initializer.

Private methods are also quite different because this 'readonly concept' is **not** about controlling mutation from outside the class. It's about controlling it inside the class.

#16 - 07/20/2016 01:49 AM - shyouhei (Shyouhei Urabe)

FYI we looked at this issue at yesterday's developer meeting. I tried to support this for being a potential optimization hint, but failed because another attendee claimed we should focus to the feature as a language, not its implementation. We didn't reach any consensus whether we should accept or reject this, though.