Ruby master - Bug #6087

How should inherited methods deal with return values of their own subclass?

02/26/2012 06:02 AM - marcandre (Marc-Andre Lafortune)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	3.0	
ruby -v:	trunk	**Backport:**

Description

Just noticed that we still don't have a consistent way to handle return values:

```
class A < Array
end
a = A.new
a.flatten.class # => A
a.rotate.class  # => Array
(a * 2).class   # => A
(a + a).class   # => Array
```

Some methods are even inconsistent depending on their arguments:

```
a.slice!(0, 1).class # => A
a.slice!(0..0).class # => A
a.slice!(0, 0).class # => Array
a.slice!(1, 0).class # => Array
a.slice!(1..0).class # => Array
```

Finally, there is currently no constructor nor hook called when making these new copies, so they are never properly constructed.

Imagine this simplified class that relies on @foo holding a hash:

```
class A < Array
  def initialize(*args)
    super
    @foo = {}
  end

  def initialize_copy(orig)
    super
    @foo = @foo.dup
  end
end
a = A.new.flatten
a.class # => A
a.instance_variable_get(:@foo) # => nil, should never happen
```

I feel this violates object orientation.

One solution is to always return the base class (Array/String/...).

Another solution is to return the current subclass. To be object oriented, I feel we must do an actual dup of the object, including copying the instance variables, if any, and calling initialize_copy. Exceptions to this would be (1) explicit documentation, e.g. Array#to_a, or (2) methods inherited from a module (like Enumerable methods for Array).

I'll be glad to fix these once there is a decision made on which way to go.

Related issues:

Related to Ruby master - Bug #10845: Subclassing String	**Open**

History

#1 - 02/26/2012 06:21 AM - trans (Thomas Sawyer)

I would think these methods should be using self.class.new for constructors thus returning the subclass. Although, that might not always possible.

#2 - 03/02/2012 10:50 AM - marcandre (Marc-Andre Lafortune)

- Assignee set to matz (Yukihiro Matsumoto)

Hi,

Thomas Sawyer wrote:

> I would think these methods should be using self.class.new for constructors thus returning the subclass. Although, that might not always possible.

This has two problems:
1) It imposes an API on the constructor of subclasses (i.e. that they accept one parameter which would be an instance of the base class)
2) The builtin classes constructors doesn't even respect that, i.e.
Hash.new({1 => 2}).has_key?(1) # => false

--
Marc-André

#3 - 03/02/2012 11:22 AM - marcandre (Marc-Andre Lafortune)

I apparently forgot to mention that I prefer the second approach, i.e. the equivalent of calling dup on the receiver.

I believe Aaron Patterson seconds this in [ruby-core:43030]

If this approach is accepted, the last remaining question is what of cases of instances of Array/String/... in which instance variables where set using instance_variable_set. Should the instance variables copied over?

```
b = []
b.instance_variable_set(:@foo, 42)
b.flatten.instance_variable_get(:@foo) # => nil or 42?
```

I think that to be consistent, they should be copied (again, assuming we decide to return an instance of subclasses). In the discussion of #4136, Charles Nutter thinks it could hinder performance to do so, but I feel that cases where such objects happen to have instance variables set should be extremely rare, so I don't think it would have much effect in practice.

--
Marc-André

#4 - 03/02/2012 12:31 PM - tenderlovemaking (Aaron Patterson)

Yes, I do second this.

#5 - 03/02/2012 07:35 PM - trans (Thomas Sawyer)

> This has two problems:
> 1) It imposes an API on the constructor of subclasses (i.e. that they accept one parameter which would be an instance of the base class)
> 2) The builtin classes constructors doesn't even respect that, i.e.
> Hash.new({1 => 2}).has_key?(1) # => false

You took me a bit too literally. I only meant it should be <u>equivalent</u> too calling self.class.new. In other words, it should return an instance of the subclass, not the base class. I did not mean to imply the necessary use of the constructor in this way --which (perhaps unfortunately) is not possible in some notable cases, as you point out.

#6 - 03/18/2012 06:46 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#7 - 05/11/2012 06:33 AM - headius (Charles Nutter)

I never noticed this before, so I'm jumping in a couple months late.

Duping the original object or copying its instance vars is wrong. Instance variables are state of an individual object, and should not be carried on to a *new* object as in these messages. There's no precedent for doing that other than dup'ing, which is explicitly for making a copy of the target object.

flatten et al are not returning "copies"...they're returning new instances with a different arrangement of the same elements. Therefore, those new objects should not automatically inherit instance variables from their parents.

It would be a good idea to design a formal way by which subclasses that *want* to propagate instance vars to new instances can do so. It just shouldn't be the default.

For the pattern that keeps coming up, where A < Array...you're doing it wrong anyway. Favor composition over inheritance :)

#8 - 07/14/2012 04:44 PM - matz (Yukihiro Matsumoto)

- Target version changed from 2.0.0 to 3.0

Array methods should return consistent values.
But we keep the behavior for now to maintain compatibility.
We will fix this (to consistently return Arrays) in 3.0.

Matz.

#9 - 12/16/2019 03:35 PM - mame (Yusuke Endoh)

- Related to Bug #10845: Subclassing String added

#10 - 04/04/2020 07:33 AM - nobu (Nobuyoshi Nakada)

- Description updated

#11 - 04/09/2020 03:12 AM - mame (Yusuke Endoh)

Recently this ticket was discussed at dev-meeting, and matz changed his mind. I remember that matz said:

- A method that seems to return a new array that is directly related to the receiver, should return an instance of the receiver's class.
- A method that seems to return a new array that is not directly related to the receiver, should return an Array.

So, we need to decide the behavior for each method.

#12 - 04/10/2020 07:17 AM - matz (Yukihiro Matsumoto)

I used to think methods should honor subclasses, but I changed my mind that the behavior made things too complex.
So if possible I want to make every method return Array instead of instance of a subclass. I just worry about the size of the incompatibility.

Matz.

#13 - 04/10/2020 07:18 AM - matz (Yukihiro Matsumoto)

Should we do an experiment in 3.0?

Matz

#14 - 04/10/2020 04:44 PM - Eregon (Benoit Daloze)

Much like all Enumerable methods return Array and (of course) do not copy instance variables, I think Array methods should do the same.

This seems particularly important since Array overrides a few methods from Enumerable for optimization but that should be entirely transparent.
For example, returning a subclass in e.g. Array#map would make it inconsistent with Enumerable#map.

So I'm in favor of no subclass handling here.
We're creating a new instance, and copying the entire state from the receiver doesn't seem reasonable to me.

If people want to keep receiver state like class and @ivars, they can always use mutating methods + #dup if needed.

#15 - 05/07/2020 07:23 AM - ko1 (Koichi Sasada)

Eregon (Benoit Daloze) wrote in #note-14:

> Much like all Enumerable methods return Array and (of course) do not copy instance variables, I think Array methods should do the same.

+1

#16 - 05/13/2020 03:14 PM - Dan0042 (Daniel DeLorme)

> - A method that seems to return a new array that is directly related to the receiver, should return an instance of the receiver's class.
> - A method that seems to return a new array that is not directly related to the receiver, should return an Array.

So this is the old thinking?

> I used to think methods should honor subclasses, but I changed my mind that the behavior made things too complex.

And this is the new thinking? In that case +1

If a subclass needs a method to return an instance of the subclass, it can easily and safely opt-in to this behavior (similar to Hash)

```
class A < Array
  def select(...)
    A.new(super) #or e.g. dup.replace(super) depending on specifics of the subclass
  end
end
```

On the other hand returning a subclass by default opens the door to all kinds of complexity and bugs depending on how the subclass is implemented. In particular if it has any state/ivars. ary.select is not the same as ary.dup.select! in that case.

Is there somewhere a complete list of methods that currently return a subclass?
For Array I think there's only this: drop, drop_while, take, take_while, flatten, uniq, slice