Ruby trunk - Feature #12281

Allow lexically scoped use of refinements with `using {}` block syntax

04/14/2016 03:48 AM - danielpclark (Daniel P. Clark)

Status:	Assigned
Priority:	Normal
Assignee:	shugo (Shugo Maeda)
Target version:	

Description

In Ruby 2.2.3 a refinement could be used in a begin/end block.

```
module Moo
  refine Fixnum do
    def to_s
      "moo"
    end
  end
end

begin # valid Ruby 2.2.3 and NOT Ruby 2.3
  using Moo
  1.to_s
end
# => "moo"
```

Since this use case has been removed I would like to propose an alternative.

```
using Moo do
  1.to_s
end
# => "moo"
```

I would like to propose allowing refinements to take a block and perform the refinement within the block and work just as if it were in it's own lexically scoped class.

I've been writing a lot of Rust lately and have found that their way of implementing Traits is just like Ruby's refinements except for that you can use Rust's version of refinements anywhere. Since Ruby's implementation is strictly lexically scoped I merely suggest a block syntax for using to allow greater expansion of refinements.

```
// Rust
impl MyCapitalize for String {
  fn my_capitalize(&self) -> Self {
    // code here
  }
}

use MyCapitalize;
String::from("hello").my_capitalize()
```

Rust lets you use the "refinement" of the trait implementation anywhere you use use just like Ruby's using. But currently Ruby restricts where using can be used. I would like that restriction to be lifted by allowing using to take a block.

```
# Ruby
module MyCapitalize
  refine String do
    def my_capitalize
      # code here
    end
  end
end

using MyCapitalize do
  "hello".my_capitalize
```

```
end
# => "Hello"
```

This way we keep Ruby's strict lexical scope behavior and at the same time allow refinement usage anywhere we need it.

Related issues:

Related to Ruby trunk - Feature #12086: using: option for instance_eval etc.	**Open**

History

#1 - 04/14/2016 03:56 AM - danielpclark (Daniel P. Clark)

I would also like the block for using to have access to local variables.

```
def example(thing)
  using MyCapitalize do
    thing.my_capitalize
  end
end

example "hello"
# => "Hello"
```

#2 - 04/17/2016 07:34 AM - shevegen (Robert A. Heiler)

Not having any pro or contra opinion here but I would like to just briefly chime in that I find the syntax quite heavy.

```
module Foo
  refine String do
```

It feels a bit ... odd with other ruby code that I would use or write, to suddenly have a constant after a method or keyword, and then a block. Perhaps I am just not used to it but my brain seems to take longer. I wonder if we could have some other way for refinements but I digress - sorry for the semi off-topic part from me here.

#3 - 04/19/2016 10:02 AM - danielpclark (Daniel P. Clark)

I found a way to use refinements in a block anywhere! Yay :-)

```
module Moo
  refine Fixnum do
    def to_s
      "moo"
    end
  end
end

class << Class.new # valid Ruby 2.3.0
  using Moo
  1.to_s
end
# => "moo"
```

Since this is valid lexical scope and lets me use Ruby's refinements anywhere I'm relatively happy with this. The one down side to this approach is there is no access to local variables. So the feature request is still valid IMHO. I'd like a way to use refinements in a block with access to local variables.

#4 - 06/13/2016 07:15 AM - shyouhei (Shyouhei Urabe)

- Assignee set to shugo (Shugo Maeda)

- Status changed from Open to Assigned

#5 - 06/13/2016 07:30 AM - matz (Yukihiro Matsumoto)

- Related to Feature #12086: using: option for instance_eval etc. added

#6 - 06/13/2016 07:44 AM - shugo (Shugo Maeda)

Daniel P. Clark wrote:

> In Ruby 2.2.3 a refinement could be used in a begin/end block.
>
> ```
> module Moo
>   refine Fixnum do
>     def to_s
>       "moo"
> ```

```
    end
  end
end

begin # valid Ruby 2.2.3 and NOT Ruby 2.3
  using Moo
  1.to_s
end
# => "moo"
```

Even in Ruby 2.2.3, it does't work as you expect:

```
begin
  using Moo
  p 1.to_s #=> "moo"
end
p 1.to_s #=> "moo", not "1"
```

I'd like to introduce #12086 instead, because it's more useful to implement internal DSLs.