Ruby trunk - Feature #7986

Custom case statement comparison method

02/28/2013 10:09 PM - trans (Thomas Sawyer)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:	Next Major	

Description

=begin
Case statements use #=== to handle matching, but sometimes this can be limiting. It such cases it would be helpful to be able to specify the comparison method. So I wondered if that could be done by hanging the method off the case keyword, e.g.

```
class Bar < Foo; end

case == obj.class
when Foo
  p "Foo, not Bar!"
when Bar
  p "Bar!"
end
```

And

```
case.include? name
when a
  p "a includes name"
when b
  p "b includes name, not a"
else
  p "neither a or b includes name"
end
```

=end

History

#1 - 03/01/2013 12:56 PM - drbrain (Eric Hodel)

=begin
Why not use the features of case statements properly? For classes, place the subclass above the superclass. For array inclusion, use splat:

```
class Foo; end
class Bar < Foo; end

def case_klass obj
case obj
when Bar
p "Bar!"
when Foo
p "Foo, not Bar!"
end
end

case_klass Foo.new
case_klass Bar.new

def case_include obj
case obj
when *%w[b]
p "b includes name, not a"
when *%w[a]
p "a includes name"
else
p "neither a or b includes name"
end
```

end

case_include 'a'
case_include 'b'
case_include 'c'

=end

#2 - 03/01/2013 05:42 PM - trans (Thomas Sawyer)

=begin
One does not necessarily have the luxury of putting the classes in order. Consider a case that uses inheritance:

class Handler
def handle(obj)
case obj
when Foo
"Foo"
end
end
end

class SubHandler < Handler
def handle(obj)
result = super(obj)
return result if result

```
  case obj
  when Bar
    "Bar"
  end
end
```

end

Now add a dozen or so additional classes and that is basically the use case I presently have.

As for the splat, one can do that for this particular example. (Though I wonder how efficient splatting in this way will be). I was just trying to give another example off the top of my head. But the point is that any method can be used, which is nice for it's flexibility.

```
case.foo? name
when a
  p "a foos name"
when b
  p "b foos name, not a"
else
  p "neither a nor b foos name"
end
```

=end

#3 - 03/01/2013 06:53 PM - duerst (Martin Dürst)

On 2013/03/01 17:42, trans (Thomas Sawyer) wrote:

> One does not necessarily have the luxury of putting the classes in order. Consider a case that uses inheritance:

[snip]

> Now add a dozen or so additional classes and that is the use case I presently have.

I'm of course not familiar with your problem or program, but a proliferation of case statements, or case alternatives, is usually taken as a sign that there is something more fundamental that needs to be improved.

Regards, Martin.

#4 - 03/01/2013 10:45 PM - trans (Thomas Sawyer)

=begin
@martin Being able to rely on inheritance and case statements I am able to remove hundreds of lines of complex DSL code. For example, old code:

```
emit Foo, via: foo

def emit_foo(obj)
   ...
end
```

The new code:

```
def emit(obj)
  case obj
  when Foo
    ...
  end
end
```

While the first might look a bit prettier, the latter is far more flexible. This is a good example of the YADSL principle actually --don't create a DSL when regular code can do the job just as well or better. So that's the particular case I am presently looking at.

Regardless of my particular case though, doesn't being able to select the case operator seem like a nice bit of flexibility in itself?

Another example that comes to mind:

```
case.start_with? uri
when "https:"
  ...
when "http:"
  ...
when "ftp:"
  ...
else
  ...
end
```

Think of the efficiency improvements over the typical use of regular expressions in such a case.
=end

#5 - 03/02/2013 05:43 AM - drbrain (Eric Hodel)

=begin
It is puzzling that you must always super first, then if unhandled execute the case expression, but do nothing if the object is unknown. It seems very error prone.

Why not execute the subclass case expression first, then super in its else?

As such, I agree with Martin that there is something wrong with your design. If you study design patterns you'll find that dispatching to methods is preferred (such as in the visitor pattern) to using comparison constructs due to the inflexibility that your design displays.

A case expression can already handle URI matching just fine:

case uri
when /\Ahttps:/i then …
when /\Ahttp:/i then …
when /\Aftp:/i then …
else …
end

or if you use URI objects:

case uri
when URI::HTTPS then …
when URI::HTTP then …
when URI::FTP then …
else …
end

or:

case uri.scheme.downcase
when 'https' then …
when 'http' then …
when 'ftp' then …
else …
end

=end

#6 - 03/02/2013 05:59 AM - trans (Thomas Sawyer)

=begin

> "It is puzzling that you must always super first, then if unhandled execute the case expression, but do nothing if the object is unknown. It seems very error prone."

Why? That's not uncommon. In this particular case you don't "always". It depends on which you want to take precedence, the superclass or the subclass. Recall my case is in place of a DSL --it's up to the end developer to decide which. Super may even be called in the middle somewhere. It is not at all error prone in the least.

> A case expression can already handle URI matching just fine

Of course, and I mentioned that. And what did I say? Quote: "think of the efficiency improvements over the typical use of regular expressions". Moreover the point has nothing whatsoever to do with URIs.

I could sit here and give 100 different examples and for everyone, someone could say, well you could also do it this way. In the end you can replace every case statement with if/else too.
=end