CommonRuby - Feature #10477

Implicit interfaces

11/04/2014 11:16 PM - sobrinho (Gabriel Sobrinho)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Hello guys,

I would to suggest us to discuss about implementing implicit interfaces on Ruby like Go.

> Go does not have classes. However, you can define methods on struct types. The method receiver appears in its own argument list between the func keyword and the method name.

This means you can specify a implicit interface where the implementation packages and packages that define the interfaces neither depends on the other.

That keeps the concept of duck typing but adds a extra layer of interface security to the language instead of relying on NoMethodError exceptions.

Go usage example:

```
type Vertex struct {
    X, Y float64
}

func (v *Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

In Ruby it could something like that:

```
interface Cache
  def get(key, default = nil)
  def set(key, value, ttl = nil)
  def delete(key)
end

class App
  def initialize(cache_store Cache)
    @cache_store = cache_store
  end

  delegate :get, :set, :delete, :to => :@cache_store
end
```

In this case a NoMethodError would never occur on App#get, App#set and App#delete.

If you think about service objects, you may have things like that:

```
class Buy
  def self.finish(object)
    object.store(
      fine: FineCalculator.calculate(object.value, Date.current)
      interest: InterestCalculator.calculate(object.value, Date.current)
      expedient: ExpedientCalculator.calculate(object.value, Date.current)
    )

    BuyMailer.deliver(to: object.buyer, object: object)
  end
```

```
end
```

In a case of a failure on calling object.buyer, the object.store has already happened and may affect the system in a bad way, which may not be acceptable.

Using a implicit interface it would never happen:

```
interface Purchasable
  def store(attrs)
  def value
  def buyer
end

class Buy
  def self.finish(object Purchasable)
    object.store(
      fine: FineCalculator.calculate(object.value, Date.current)
      interest: InterestCalculator.calculate(object.value, Date.current)
      expedient: ExpedientCalculator.calculate(object.value, Date.current)
    )

    BuyMailer.deliver(to: object.buyer, object: object)
  end
end
```

I think it's a great idea of Go that would be of benefit in Ruby.

Probably there is better usage cases, sorry about that, but the concept is to have implicit interfaces on libraries that we publish for everyone (gems).

Think about complex interfaces like capybara drivers, active support cache drivers and etc.

How it sounds?

Reference:
http://programmers.stackexchange.com/questions/197356/how-does-go-improve-productivity-with-implicit-interfaces-and-how-does-that-c