Ruby's Eigenclass

Introduction

In Ruby, there are three types of methods that can be applied to a class:

1. Instance methods
2. Singleton methods
3. Class methods

The first are available to all "instances" that are created from the class; the second are only available to a specific instance of a class and finally, the last are only available to the class itself.

Let's see an example of each:

```
# Instance methods

class Foo
  def speak
    puts "hello"
  end
end

foo = Foo.new
bar = Foo.new

foo.speak # => hello
bar.speak # => hello

# Singleton methods

class Foo; end
```

```
foo = Foo.new
bar = Foo.new

def foo.speak
  puts "hello"
end

foo.speak # => hello
bar.speak # => NoMethodError: undefined method `speak' for #<Foo:0x007f97b60c

# Class methods

class Foo
  def self.speak
    puts "hello"
  end
end

foo = Foo.new

foo.speak # => NoMethodError: undefined method `speak' for #<Foo:0x007f97b3af

Foo.speak # => hello
```

The above code (both singleton methods and class methods) demonstrates what is known in the Ruby community as an "Eigenclass". This is effectively an anonymous class that Ruby creates and inserts into the inheritance hierarchy to hold the class methods (thus not interfering with the instances that are created from the class).

Another way of accessing an Eigenclass is with the following syntax (`class <<`):

```
# Singleton methods

class Foo; end

foo = Foo.new

class << foo
  def speak
    puts "hello"
  end
end
```

```
foo.speak # => hello

# Class methods

class Foo
  class << self
    def speak
      puts "hello"
    end
  end
end

Foo.speak # => hello
```

Singleton and Class method visibility

The reason understanding the Eigenclass is important is because it helps to clarify how to make some methods private when they otherwise would seem impossible to make private.

Let's see an example of what I'm referring to:

```
class Foo
  def self.bar
    p "im public"
  end

  private
  def self.baz
    p "im private"
  end
end

Foo.bar # => im public
Foo.baz # => im private
```

Notice that although we've specified `def self.baz` to be `private` it is still accessible directly. This is because `private` only applies to instance methods; and although Singleton/Class methods are instance methods (of the Eigenclass!), in the above example the `private` method actually applies to the `Foo` class and

not the Eigenclass, and so the class level methods aren't affected by the `private` method.

> Note: `private` is a method and not a directive or special syntax.

Now that we've seen the problem, let's see how to fix this by taking advantage of the Eigenclass:

```
# Singleton method example

class Foo; end

foo = Foo.new

class << foo
  def speak
    implementation_details
  end

  private

  def implementation_details
    puts "hello!!!"
  end
end

foo.speak # => hello!!!
foo.implementation_details # => NoMethodError: private method `implementation

# Class method example

class Foo
  class << self
      def speak
        implementation_details
      end

      private

      def implementation_details
        puts "hello!!!"
      end
  end
end
```

```
Foo.speak # => hello!!!
Foo.implementation_details # => NoMethodError: private method `implementation
```

In the above code we're demonstrating how to make some methods public (i.e. `speak`) while keeping other implementation specific functions private (these types of methods we don't want to be public as there is no reason for an end user to call the method directly).

In the above code the `private` method (for both Singleton and Class method examples) goes through the Eigenclass, and as the class methods are technically instance methods on the Eigenclass, so the `private` directive is enforced.

Meta Programming Version

> Note: thanks to Peter Cooper for the following tip

We can utilise a little Ruby meta programming to achieve the same result:

```
class Foo
  def self.bar
    p "im public"
  end

  def self.baz
    p "im private"
  end

  private_class_method :baz
end

Foo.bar # => im public
Foo.baz # => NoMethodError: private method `baz' called for Foo:Class
```

Links

- Home
- About Me

- GitHub
- Twitter
- Resume