

# Deciphering the Ruby Object Model

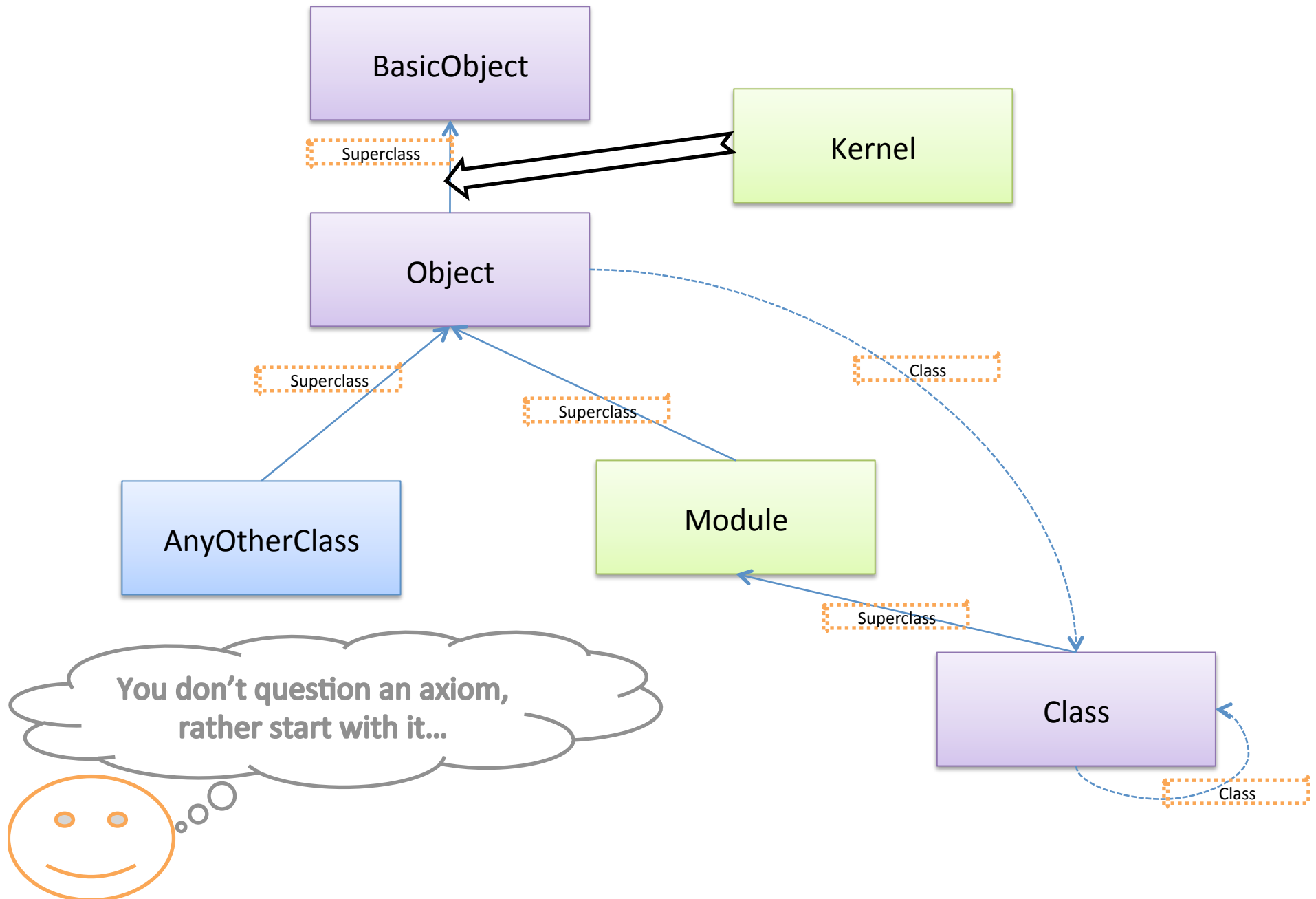
# Who am I?

- Karthik Sirasanagandla
- A Pragmatic Programmer @ ThoughtWorks
- VB freelance programmer for about a year
- Java/JEE apps. developer for about 5 years
- Ruby Enthusiast for over 8 months now
- Haskell newbie
- Occasionally blog at [kartzontech.blogspot.com](http://kartzontech.blogspot.com)
- Not so frequently tweet using [@kartzontech](https://twitter.com/kartzontech)

# Why Learn Ruby Object Model?

- Good if you know “syntax and semantics”
- Better if you know “Ruby’s Object Model”
- *“A language the does not affect the way you think about programming is not worth learning at all” – Alan Perlis*

# The Axiom



# Classes

~~Classes should be closed for modification  
and open to extension~~

```
class MyClass  
end
```

```
class MyClass  
  def im      # instance method  
    puts "Instance method"  
  end  
  
  def self.cm #class method  
    puts "Class method"  
  end  
end
```

# Classes

Open Classes ???



Monkey Patching ???



*“With more power comes more responsibilities” opines Matz, like uncle Ben in the spiderman movie*

# Classes

Class methods can be defined in number of ways:

```
class MyClass
  def MyClass.cm1
    puts "cm1() invoked..."
  end

  def self.cm2
    puts "cm2() invoked..."
  end

  class << self
    def cm3
      puts "cm3() invoked..."
    end
  end
end

end
```

# Classes

Classes are objects

```
class MyClass
  def im
    puts "Instance method"
  end
end
```

is the same as:

```
MyClass = Class.new do
  def im
    puts "Class method"
  end
end
```

# Classes

Quiz Time:

```
class MyClass < String.new
  def self.to_s
    puts "Class method"
  end
end
```

MyClass.to\_s

What is the output?

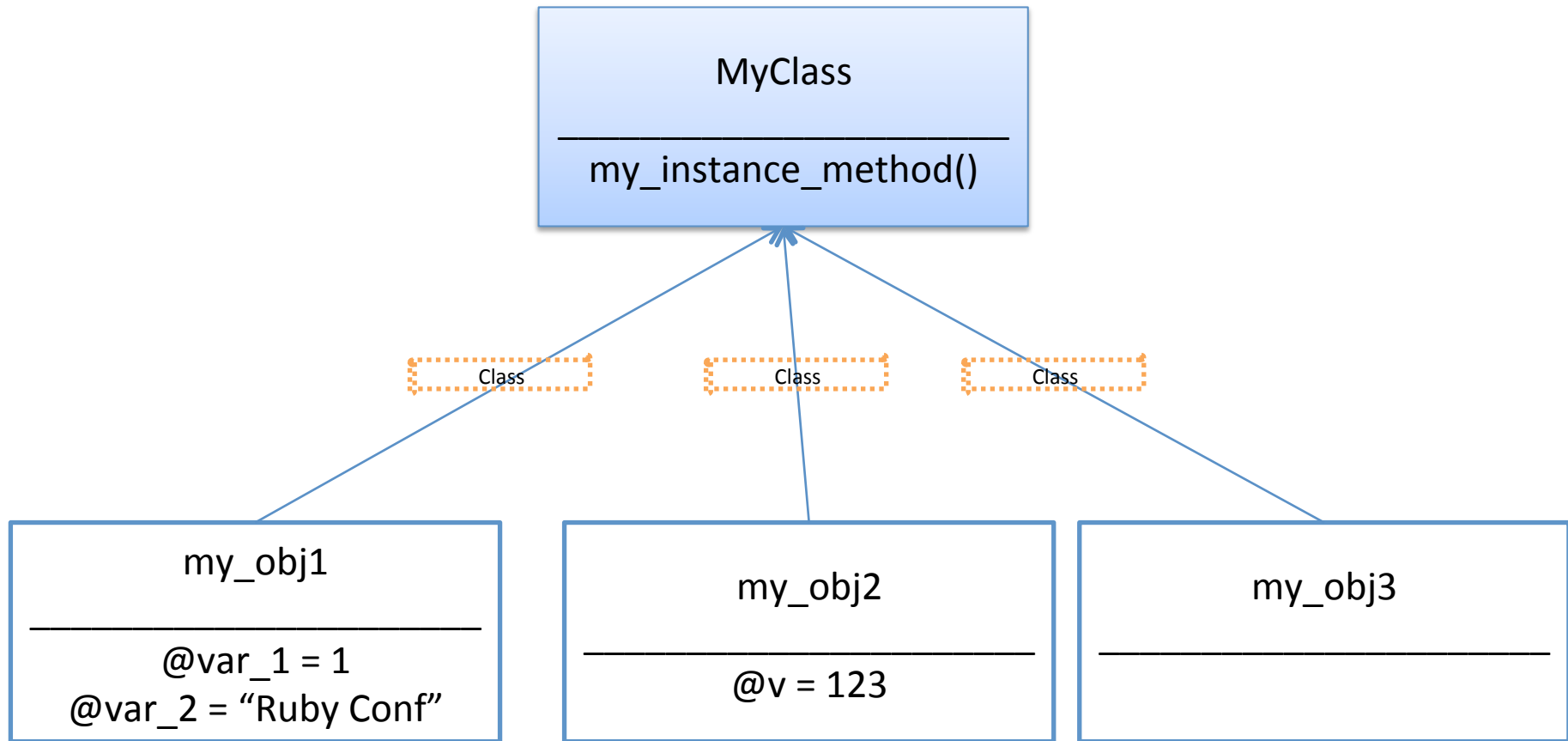
**TypeError: Wrong argument type String (expected Class)**

Lesson:

Classes inherit from classes only not from just any object

# Classes

No connection between a class and its objects' instance variables



# Classes

Quiz:

What is the Java/C# Interface equivalent in Ruby?

[FYI: This has nothing to do with what I have been talking about all through!!]

Answer:

**Interfaces are irrelevant in Ruby.**

But why?

Duck Typing is the answer.

~~Program to an interface~~

No need to inherit from a common interface.

# The joy of “**self**” discovery

- **self** is not the same as **this** in Java
- **self** is synonymous to **current/default object**
- Who gets to be **self**  
depends on  
Where **self** is

# The joy of “**self**” discovery

Code is an unbeatable teaching aid. What say?

```
p “At Topmost level, self is #{self}”
```

```
class MyClass
```

```
  p “In class definition block, self is #{self}”
```

```
  def self.my_class_method
```

```
    p “In my_class_method, self is #{self}”
```

```
  end
```

```
  def my_instance_method
```

```
    p “In my_instance_method, self is #{self}”
```

```
  end
```

```
end
```

# Class, Instance and Class Instance Variables

Experience is the best teacher.

```
class A
  y = 1
  @p = 2
  @q
  @@t = 3

  def initialize
    @@r ||= 4
    @s = 5
  end
end

puts "Class instance variables of A are #{A.instance_variables}"
[:@p]

puts "Class variables of A are #{A.class_variables}"
[:@@t]

a = A.new
puts "a, of type A, has instance variables #{a.instance_variables}"
[:@s]

puts "Class variables of A are #{A.class_variables}"
[:@@t, :@@r]
```

# Access modifiers

- Ruby has 3 access modifiers for methods:
  - Public (default)
  - Private
  - Protected
- Access modifiers apply until the end of scope, or until another access modifier pops-up

```
class MyClass
```

```
  #public is the default access modifier  
  def m1; end
```

```
  private  
  def m2; end;  
  def m3; end;
```

```
  protected  
  def m4; end;
```

```
end
```

# Access modifiers

- **private, public, protected are all methods**, so you can pass method name as symbol to it and change its visibility

```
class MyClass
```

```
  def m1; end
```

```
  def m2; end;
```

```
  def m3; end;
```

```
  def m4; end;
```

```
  public :m1
```

```
  private :m2, :m3
```

```
  protected :m4
```

```
end
```

# Access modifiers

- `public`
- `private`
  - only accessible within the scope of a single object in which it is defined (truly private)
- `protected`
  - accessible from the scope of a method belonging to any object that's an instance of the same class

```
class MyClass
```

```
  protected
```

```
  def my_protected_method  
  end
```

```
  public
```

```
  def my_public_method(other)  
    self.my_protected_method  
    other.my_protected_method  
  end
```

```
end
```

```
mc1 = MyClass.new
```

```
mc2 = MyClass.new
```

```
mc1.my_public_method(mc2)
```

```
mc1.my_protected_method
```

```
#=> Works ☺
```

```
#=> NoMethodError
```

# Access modifiers

- Quiz

```
class Speaker
```

```
  def talk
```

```
    self.confident? ? "lecture..." : "abscond!"
```

```
  end
```

```
private
```

```
  def confident?
```

```
    true
```

```
  end
```

```
end
```

```
Speaker.new.talk
```

What is the output?

**NoMethodError**

Lesson:

Private methods can be called only with implicit receiver.

No explicit receiver. Not even self.

# Access modifiers

- Quiz

We talked about access modifiers with methods as example.

**What about the access modifiers for instance variables?**

**Answer:**

Access modifiers don't apply to Instance variables (@inst\_vari).

**Instance variables always remain private.**

Access to instance variables are only through getter and setter methods!

# Singleton Methods

A method defined to be specific to a single object

```
statement = "Karthik is speaking at Ruby Conf India 2011"
```

```
def statement.really?  
  true  
end
```

```
statement.really?
```

What is the output?

```
#=> true
```

```
another_statement = "Karthik is bull shitting at Ruby Conf 2011"
```

```
another_statement.really?
```

What is the output?

```
#=> undefined method error
```

# Singleton Methods

How about a class method? Isn't that tied to a single object?

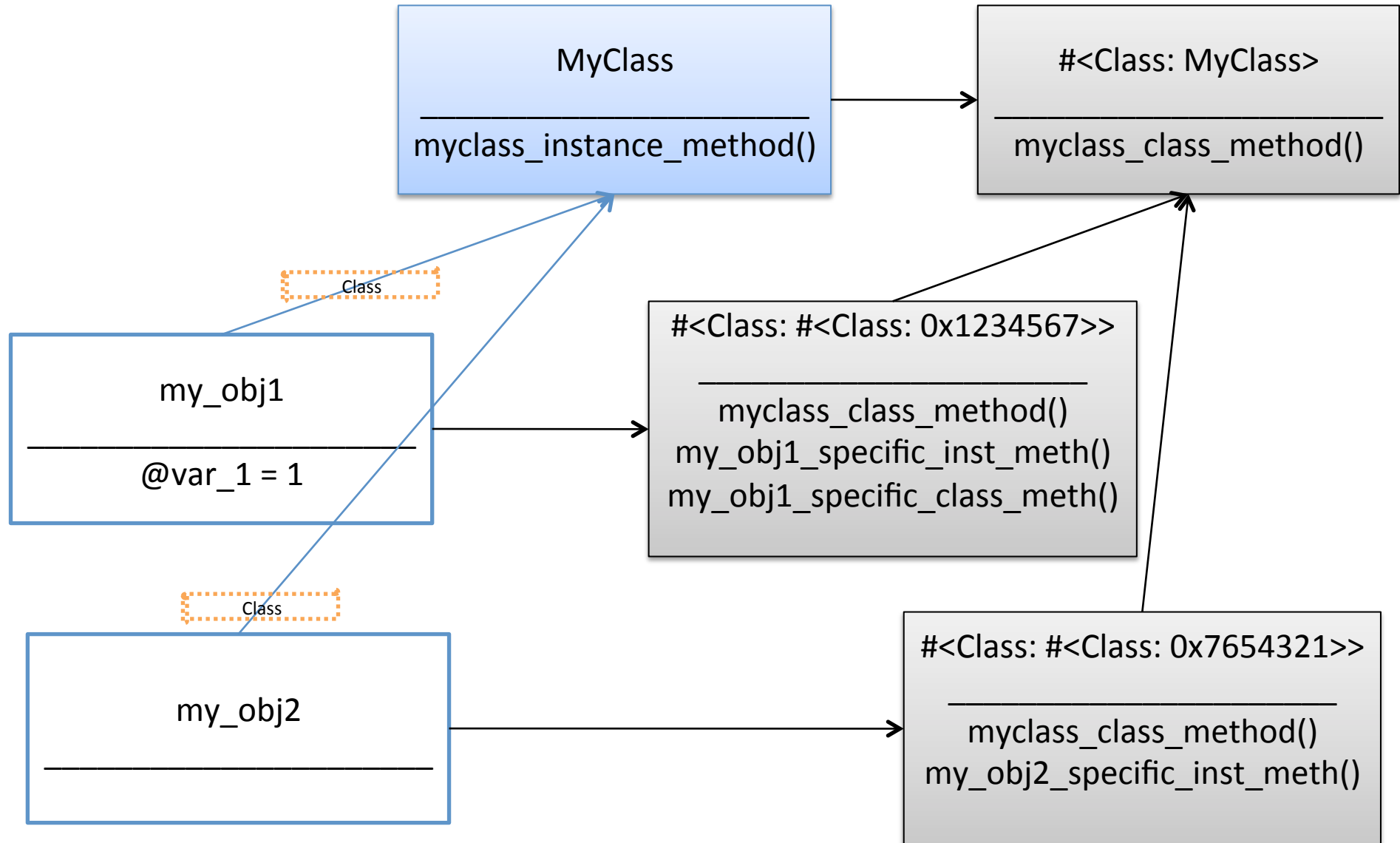
Eg:

```
def MyClass.my_class_method  
  #blah..  
end
```

**Class methods are indeed singleton methods!**

# Methods – where do they reside?

It depends!!!



# Methods -> Function call or Message???

- No function call. It's all messages!
- `duck.quack()`
  - As Java programmer, I see it as looking up for “`quack`” as member function in a table and call it.
  - As Ruby programmer, I see it as passing a message “`quack`” to the object “`duck`”.
- No Method Overloading.

# Module

- Used for namespacing

```
module MyModule
```

```
  MyConstant = "MyModule::Myconstant"
```

```
  class MyClass
```

```
    MyConstant = "MyModule::MyClass::MyConstant"
```

```
  end
```

```
end
```

```
puts MyModule::MyConstant      # => MyModule::MyConstant
```

```
puts MyModule::MyClass::MyConstant  #=> MyModule::MyClass::MyConstant
```

Another example:

```
ActiveRecord::Base.connection.execute(sql_query)
```

# Module

- Better alternate to multiple inheritance.
- DP: “Favor composition over inheritance”
- Code Examples:

```
module MyModule
  def my_meth
    puts “my_meth() of MyModule”
  end
end
```

```
class MyClass
end
```

Case 1: Include MyModule instance methods as instance methods of myClass

```
class MyClass
  include MyModule # ← ← Just include it...
end
```

# Module

```
module MyModule
  def my_meth
    puts "my_meth() of MyModule"
  end
end
```

```
class MyClass
end
```

Case 2: Include instance methods of MyModule  
as class methods of MyClass

```
class MyClass
  class << self
    include MyModule
  end
end
```

```
class MyClass
  extend MyModule
end
```

# Module

Quiz:

```
module MyModule
  def self.my_freakin_meth
    puts "my_freakin_meth() of MyModule"
  end
end
```

```
class MyClass
  include MyModule
end
```

```
MyClass.my_freakin_meth
```

What is the output?

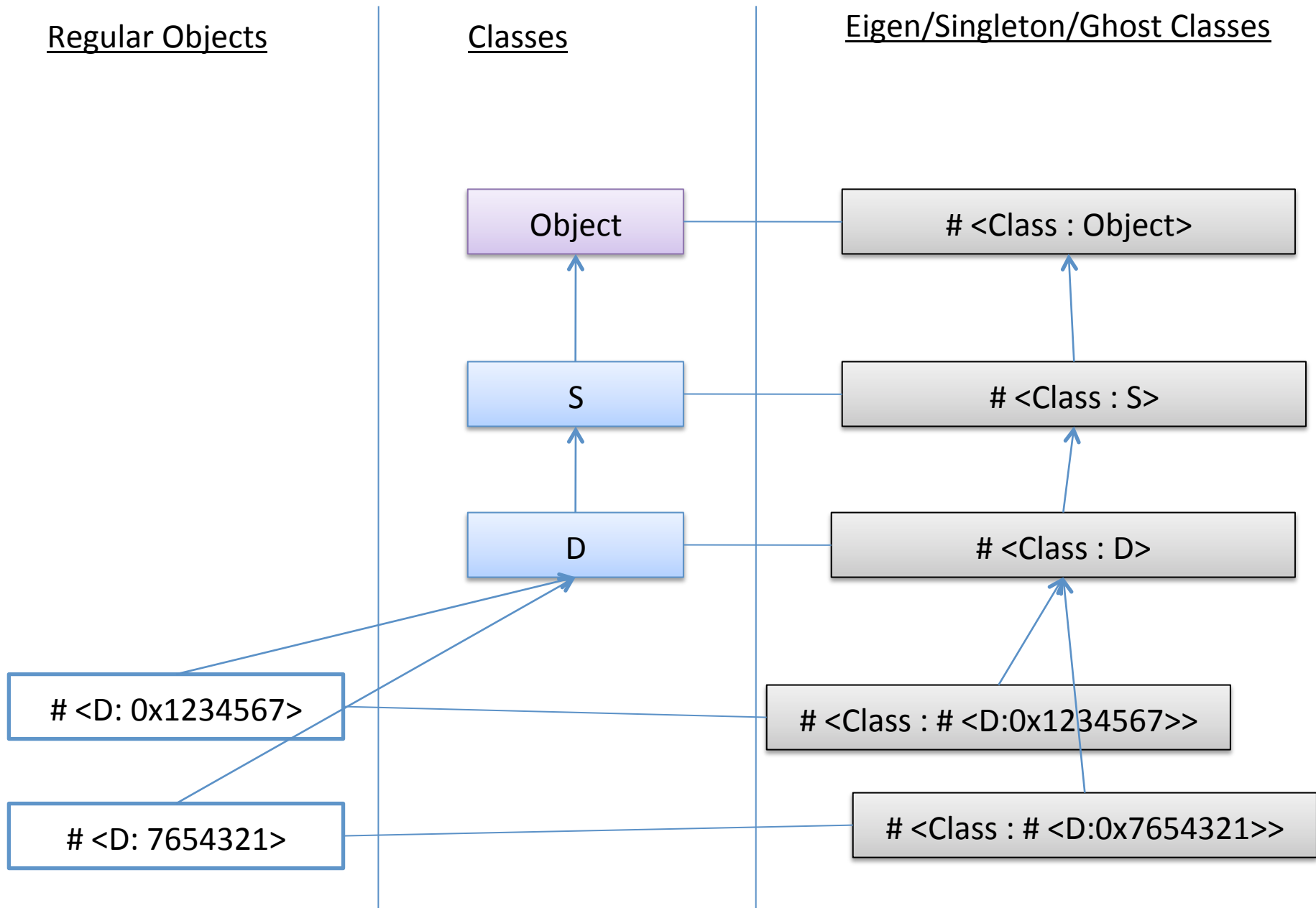
```
#NoMethodError
```

Lesson:

When an object includes a module

- Module's instance methods are *included*
- Module's class methods are *excluded*

# Method Lookup



# Resources

- [Programming Ruby](#) by Dave Thomas
- [Metaprogramming Ruby](#) by Paolo Perrotta
- [The Well-grounded Rubyist](#) by David A Black
- [OneStepBack.org](#) by Jim Weirich
- [blog.jayfields.com](#)

???

# Thanks.

Don't forget to whisper your feedback in my ears!