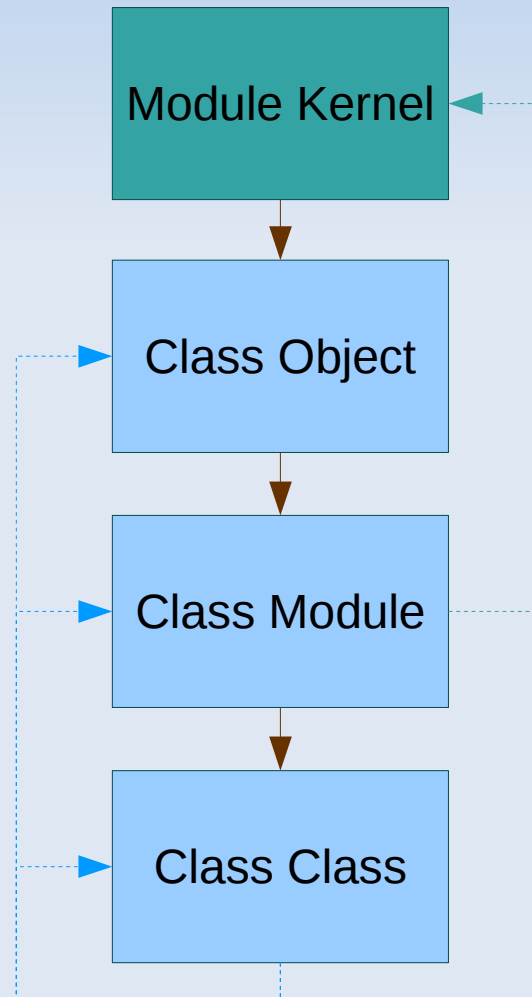


Core of the Core

A series on the **core** of the Ruby Object Model



Part 2: Reflection



Who am I?

What is Reflection?

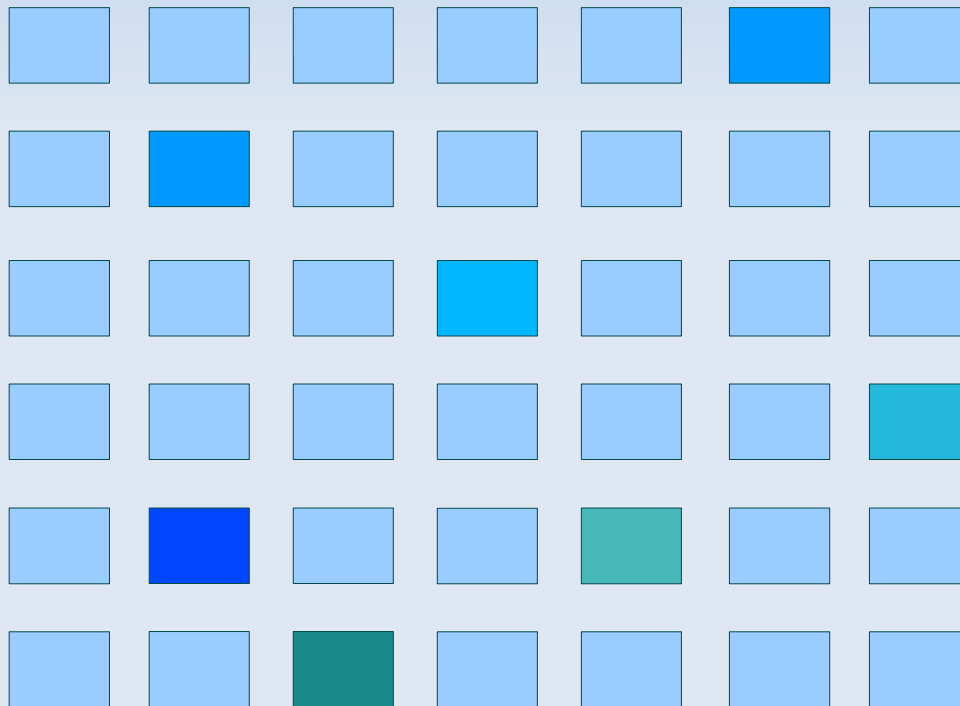
Reflection is the ability of a computer program to observe and modify itself.

What is in the **object space**?

What is a particular **object**?

Reflecting on a Ruby Process

What is in the **object space**?



Reflecting on a Ruby Process

`ObjectSpace.each_object(class_or_module, &blk)`

Calls the block once for each living, nonimmediate object in the Ruby process.

Immediate objects:

- fixnums
- symbols
- true/false
- nil

Using ObjectSpace.each_object

```
print_it = lambda {|obj| puts obj}

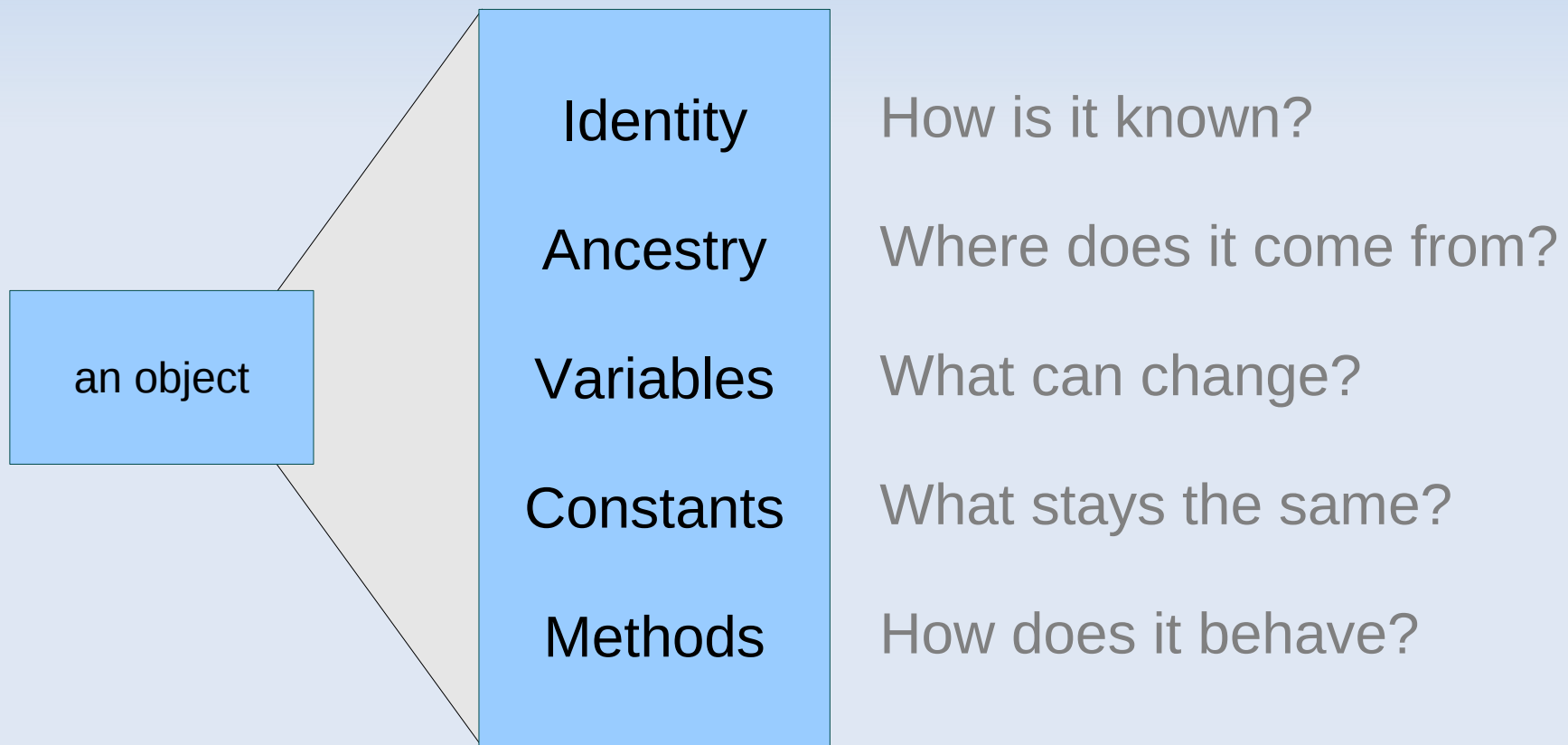
# Print each class
ObjectSpace.each_object(Class, &print_it)

# Print each module
ObjectSpace.each_object(Module, &print_it)

# Print everything (I don't recommend it)
ObjectSpace.each_object(&print_it)
```

Aspects of Reflection

What is a particular **object**?



The Identity of an Object

`object.object_id`

`object.__id__`

Returns an integer identifier for an object

```
> a = 'abc'
> a.object_id
] 70083886640800
> ('ab' + 'c').object_id
] 70083886621640
> String.object_id
] 70032390145320
```

The Identity of an Object

`object.inspect`

Returns a string containing a human-readable representation of `obj`. If not overridden, uses the `to_s` method to generate the string.

```
class Foo
  def inspect
    'foo'
  end
end
```

```
> Foo.new
] foo
```

The Identity of an Object

object.class

Returns the class of an object. It must be called with an explicit receiver.

```
> 'abc'.class
] String
> String.class
] Class
> Class.class
] Class
```

The Ancestry of an Object

`class.superclass`

Returns the superclass of class, or nil.

```
> 'abc'.superclass
] NoMethodError: undefined method `superclass'
for "abc":String
> String.superclass
] Object
> Object.superclass
] nil
```

The Ancestry of an Object

`module.ancestors => an array`

Returns a list of modules included, plus itself.

```
> String.ancestors
] [String, Enumerable, Comparable, Object,
Kernel]
> Object.ancestors
] [Object, Kernel]
> Module.new { include Math }.ancestors
=> [#<Module:0x7f6de1a0c638>, Math]
```

The Variables of an Object

Kernel.local_variables

Kernel.global_variables

```
> i = 'abc'
> local_variables
] ["_", i]
> global_variables
] ["$-w", "$:", "$.", "$KCODE", "$-F", "$*", "$stderr",
"$,", "$`", "$binding", "$-p", "$\\", "$$", "$<", "$@",
"$-v", "$-i", "$deferr", "$\\", "$=", "$;",
"$PROGRAM_NAME", "$stdout", "$&", "$-d", "$LOAD_PATH",
"$-a", "$VERBOSE", "$FILENAME", "$defout", "$-0", "$+",
"$0", "$stdin", "$~", "$DEBUG", "$-I", "$_", "$-K",
"$>", "$/", "$'", "$-l", "$LOADED_FEATURES", "$?",
"$SAFE", "$!"]
```

The Variables of an Object

module.class_variables

Returns an array of the names of class variables in the module/class and its ancestors.

```
class A
  @@a = 1
end

class B < A
  @@b = 2

  def self.later
    @@c = 3
  end
end
```

```
> A.class_variables
] [ "@@a" ]
> B.class_variables
] [ "@@b", "@@a" ]
> B.later
> B.class_variables
] [ "@@b", "@@c", "@@a" ]
```

The Variables of an Object

object.instance_variables

Returns an array of instance variable names for the receiver. Note: defining an accessor does not create the corresponding instance variable.

```
class Foo
  def self.in_class
    @class_instance = 'a'
  end

  def in_instance
    @instance = 'b'
  end
end
```

```
> Foo.instance_variables
] []
> Foo.in_class
> Foo.instance_variables
] ["@class_instance"]
> f = Foo.new
> f.instance_variables
] []
> f.in_instance
> f.instance_variables
] ["@instance"]
```

Get and Set Instance Variables

`object.instance_variable_get(sym)`

`object.instance_variable_set(sym, obj)`

Gets or sets an instance variable

```
class A
  def initialize
    @a = 1
  end
end
```

```
> a = A.new
> a.instance_variable_get(:@a)
] 1
> a.instance_variable_set(:@a, 'abc')
] 'abc'
> a.instance_variable_get(:@a)
] 'abc'
> a.instance_variable_get(:@b)
] nil
```

The Constants of an Object

module.constants

Returns an array of the names of the constants accessible. This includes the names of constants in any included modules.

```
module A
  B = 2
end
```

```
class C
  include A
  D = 4
end
```

```
> A.constants
] ["B"]
> C.constants
] ["D", "B"]
```

Get and Set Constants

```
module.const_get(sym)
```

```
module.const_set(sym, obj)
```

Gets or sets a named constant

```
> Math.const_get(:PI)
] 3.14159265358979
> Math.const_set(:CAKE, 4)
] 4
> Object.const_get(:Math).const_get(:CAKE)
] 4
> Math::CAKE
] 4
```

The Methods of an Object

`module.instance_methods(include_super=true)`

`object.singleton_methods(include_all=true)`

`object.methods(regular=true)`

```
class A
  def self.a_singleton
    'a singleton'
  end

  def an_instance
    'an instance'
  end
end
```

```
> A.instance_methods(false)
] ["an_instance"]
> A.singleton_methods(false)
] ["a_singleton"]
> A.new.methods
] [..., "a_singleton", ...]
```

Calling a Method on an Object

`object.send(symbol, args)`

`object.__send__(symbol, args)`

```
class A
  def self.a_singleton
    'a singleton'
  end

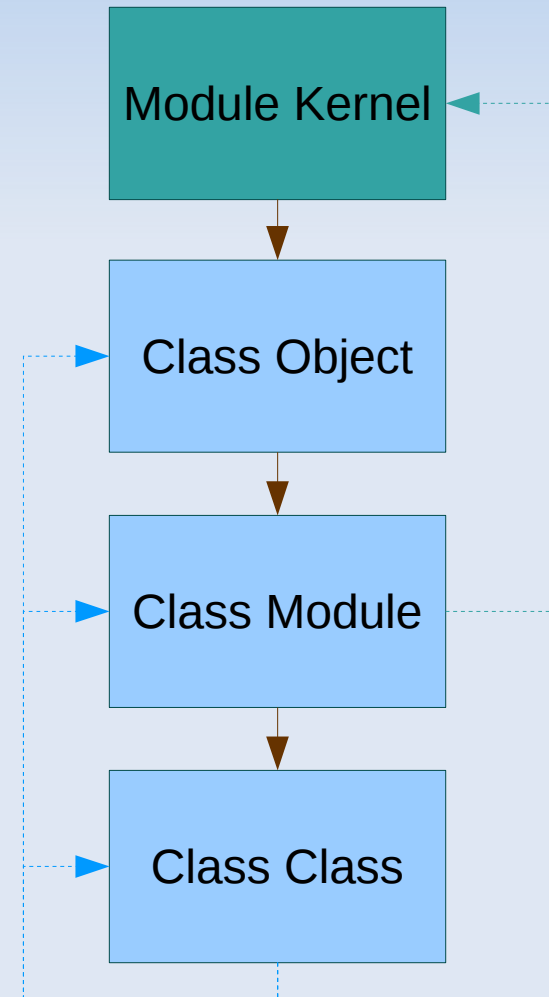
  def an_instance
    'an instance'
  end
end
```

```
> A.send(:a_singleton)
] "a singleton"
> A.new.send('an_instance')
] "an instance"
> 3.send(:+,4)
] 7
> 'abc'.send('*',3)
] "abcabcabc"
> [:a, :b, :c].send('[]',0)
] :a
```

Conclusion of Reflection

With great power comes great responsibility.

Choose wisely when to use the reflective features of Ruby.



Conclusion of Reflection

Essential Resources

- The Pickaxe
"Programming Ruby" by Dave Thomas
- Ruby Rdoc
- "Ruby Object Model" screencasts by Dave Thomas

Reflection on The Speaker

```
class TheSpeaker
  def self.speaker_name
    p 'Ben Wagaman'
  end

  def self.twitter
    p 'jamin4jc'
  end

  def self.email
    p 'benjamin.a.wagaman@chase.com'
  end
end

speaker_class = Object.const_get(:TheSpeaker)

speaker_class.methods(false).each do |message|
  speaker_class.send(message)
end
```