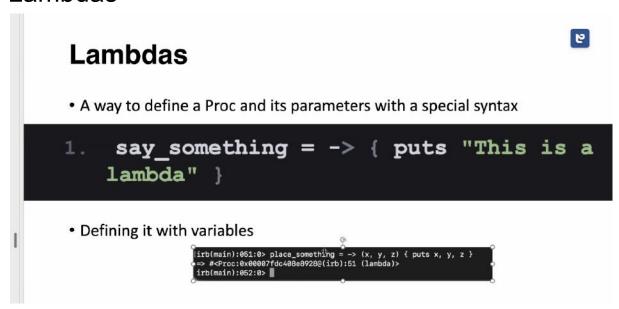
### Lambdas



#### Procedure and lambda

Lamba is a procedure, a procedure with special syntax and context.

With lambdas you use the context of the own lambda.

### **Procs**

• Different than lambdas when there are arguments.

- If you use procs, you use the context of the method where you are calling it.
- If you use procs, you create your own context

### **Procs**

• In the following example, we return in the <u>call\_proc</u> method, not the proc itself.

```
1. def call_proc
2. puts "Before proc"
3. my_proc = Proc.new { return 2 }
4. my_proc.call
5. puts "After proc"
6. end
7.
8. p call_proc
9. # Prints "Before proc" but not "After proc"
```

If you are using a procedure inside a function and you call a return, this return is going to end the function.

You can still return something if you simply put the variable or value without return as the last line of the procedure.

<u>procedures -> outside context</u> <u>lambda -> inner context.</u>

## Ruby Introspection & GC Agenda

- Introspection
  - What is introspection
  - The Object Space
  - Looking inside objects
  - Calling methods dynamically
  - A Note about performance
- The Garbage Collector
  - Introduction
  - Mark and Sweep Algorithm
  - Heap Compation
  - Performance Issues and managing our own memory

# What can a program see within itself?

- · What objects contain
- Current class hierarchy
- Contents and behaviors of objects
- Information on methods

## Introspection

2

 With the information the program knows about themselves, they can call methods at runtime and modify themselves while running

## The Object Space



- Where all objects inhabit in Ruby
- · Objects alive and not yet collected by the Garbage Collector
  - This means an object is alive as long as it has any references pointing to it

In the object space is the list of all objects that are alive.

## **The Object Space**

- What does it contain by default?
- Useful Tip: You can also use the method .each\_object without an argument (class)

The object space is a module/class.

To access the object, we use the each\_object method.

# **The Object Space**

 Please note that the ObjectSpace doesn't know about Fixnum, true, false, or nil.

```
irb(main):099:0> ObjectSpace.count_objects[:TOTAL]
=> 35054
irb(main):100:0> b = nil
=> nil
irb(main):101:0> ObjectSpace.count_objects[:TOTAL]
=> 35054
irb(main):102:0>
```



## **Looking Inside Objects**

 You can get the methods available for any variable with .methods

```
1    r = 1..10 #=> 1..10
2    list = r.methods #=> begin, end, bsearch.........
3    list.length #=> 125
4    list[0..3] #=> [begin end bsearch to_a]
5
```

### To know if a class has a method:

```
?, :respond_to?, :object_id, :send, :__send__, :!, :!=

[2.7.5 :008 > a.respond_to? :delete_prefix
=> true

2.7.5 :009 >
```

# Calling methods dynamically

Determining the objects direct class and class hierarchy

```
class Example

MYCONSTANT = 'hello this is a test'

def self.this_is_a_class_method

puts 'ok'
end

def this_is_a_public_method

puts 'ok'
end

protected

def this_is_a_protected_method

puts 'ok'
end

private

def this_is_a_private_method

puts 'ok'
end

def this_is_a_private_method

puts 'ok'
end

def this_is_a_private_method

puts 'ok'
end

end
```

```
Example.private_instance_methods
#=> [:this_is_a_private_method, :irb_binding, ....]
Example.protected_instance_methods
#=> [:this_is_a_protected_method]
Example.public_instance_methods
#=> [:this_is_a_public_method, :instance_variable_defined?, ...]
Example.singleton_methods #=> [:this_is_a_class_method]
Example.constants #=> [:MYCONSTANT]
```

(singleton methods are the class methods).

### Calling the methods dynamically

- Send
- Eval
- Method.call

### Send

- Sending a symbol (as best practice, althought it also accepts strings)
- Remember: Symbols represent methods (identifiers)

```
"Ruby on Rails!".send(:length) #=> 14
"Ruby on Rails!".send("gsub", /Rails/, "wheels") #=> "Ruby on wheels!"
"Ruby on Rails!".send(:gsub, /Rails/, "wheels") #=> "Ruby on wheels!"
```

Send is calling the method itself, but it receives a symbol

```
=> true

[2.7.5 :009 > a.send(:reverse)

=> "raks0"

2.7.5 :010 >
```

## The Method Object

- 2
- Method class that creates a method that is not bound to a class
- This means it can be used anywhere

Llamar métodos que no están dentro de una clase.

Passing a method with the method name as a symbol, allows you to pass it as a variable.

```
2.7.5 :002 >
2.7.5 :003 >
2.7.5 :004 >
2.7.5 :005 > def example_function
2.7.5 :006 > puts "ok"
2.7.5 :007 > end
=> :example_function
2.7.5 :008 > method(:example_function).call
ok
=> nil
2.7.5 :009 > method(:example_function)
=> #<Method: main.example_function() (irb):1>
2.7.5 :010 > v = method(:example_function)
=> #<Method: main.example_function() (irb):1>
2.7.5 :011 > class Example
2.7.5 :012 > def self.example(arg)
2.7.5 :013 > arg.call
2.7.5 :013 >
                 arg.call
2.7.5 :013 > al
2.7.5 :014 > end
2.7.5 :015 > end
=> :example
2.7.5 :016 > Example.example(v)
ok
=> nil
2.7.5 :017 >
```

It is similar to passing it as a procedure.

# Eval Takes a string and execute it Explicitly evaluates it into the current context a = '2.send(:class)' eval a

Receives a string and evaluates it with the current context.

### THIS IS CALLED METAPROGRAMMING

```
require 'benchmark'
              include Benchmark
   A fe
                                                                       iing
   ben
              test = "My not so long string"
              method_var = test.method(:length)

    Click

             n = 10000000
              Benchmark.bm(10) { |x|
               x.report("call") { n.times { method_var.call } }
                x.report("send") { n.times { test.send(:length) } }
               x.report("eval") { n.times { eval("test.length") } }
               x.report("norm") { n.times { test.length } }
              }.compact
              =begin
                          0.648847 0.002007 0.650854 ( 0.653695)
                         0.636859 0.000771 0.637630 ( 0.638562)
                         53.932909 0.083692 54.016601 ( 54.091896)
                         0.334988 0.000221 0.335209 ( 0.335498)
              norm
NROUTE UNIVERSIT
```

The performance of the other methods is slower.

### The benchmark is very used

### Want to know about Benchmarks?



- Benchmark is a great Ruby Module for managing and knowing about performance
- Visit https://ruby-doc.org/stdlib-3.0.0/libdoc/benchmark/rdoc/Benchmark.html

# System Hooks

- Technique that lets you trap events
- · Most common event is time of creation
- · Technique allowed by the alias method
  - · Creates new aliases of the methods

# System Hooks: An example

```
class Object

def timestamp
    return @timestamp end
    def timestamp=(aTime)
    @timestamp = aTime
    end
    end

class Class
    alias_method :old_new, :new
    def new(*args)
    result = old_new(*args)
    result.timestamp = Time.now
    return result
    end

class Test
    end

Test.new.timestamp #=> Dependable of execution time but mine was 2022-08-12 14:25:13 -0500
```

How to edit a class?

With alias\_method it creates almost a sa shadow copy

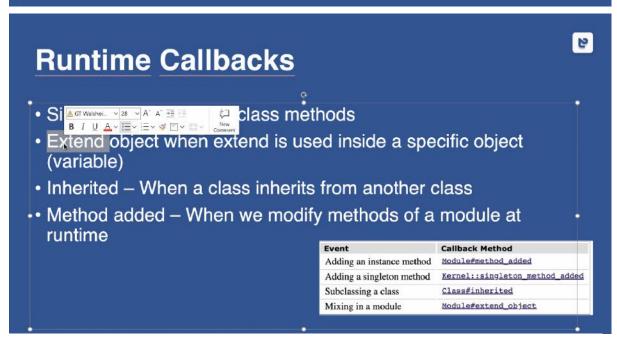
System hook, a way to add additional properties to the variables we create. How do we add a timestamp to a class?

......

## **Runtime Callbacks**

Being notified when one of the following happens

Event	Callback Method
Adding an instance method	Module#method_added
Adding a singleton method	<pre>Kernel::singleton_method_added</pre>
Subclassing a class	Class#inherited
Mixing in a module	Module#extend_object



A way to adding properties to an object. USED THE MOST FOR THAT

```
module Test
  def self.extend_object(o)
   puts "Added Test to #{ o.class } type"
  end
  end
end

(_3).extend(Test) #=> Added Test to Integer type
```

• Multiple inheritance in a class through Modules



We cannot do multiple inheritance.

A module can have a class inside of it, but it can only be used inside of the module.

# **Ruby Garbage Collector**

 Works through malloc and free, later on calloc and realloc were also used

Removes things that are not useful from the memory automatically.

### **The Mark Phase**

- Ruby Crawls through the objects and marks the ones still in use
- Colors
  - · White: Unmarked Still in use
  - · Grey: Marked but may point to white objected. Live but not scanned
  - · Black: Marked and don't point to any white objects

# The Sweep Phase

- Unmarked objects are returned to the free list
  - Separate spaces by young and old objects
  - Incremental garbage collection since Ruby 2.2
    - Now, it works in blocks
    - Spend same time in GC but in shorted and more frequent bursts
  - Protected and unprotected objects concept (from write)

End of garbage collector phases...

# Heap Compaction since Ruby 2.7

- Thanks to Mark Evans using Tmalloc
  - Tmalloc Fast memory allocation implementation of C's malloc by Google

t

# **Memory Loops: Options**

- Debugging: BEST PRACTICE. ALWAYS
- Using Jemalloc Always the second option

### **Jemalloc**

useful gem to debug memory loops, and memory usage?

**MEMORY LOOP EQUALS MEMORY LEAK** 

```
zu rouozo r4 ru ri_adu_autri_type_to_quovo_brokerage
      class MyClass
          def public_method
              my_method
          end
          protected
          def my_method
              puts "im protected"
          end
      end
      module MyModule
          class Hello
              def initialize
                  puts "new class"
          def method
              Hello.new
          end
      def ok
          a = 2
          b = Proc.new { a *= 2 }
          b. call
      def oki
          c = -> { return 3 }
          c.call
      end
37
                                            I
     MyClass.public_instance_methods[0]
```

### Manera correcta

Para llamar un método de un módulo, solo se puede llamar dentro de un módulo. Estas están protegidas dentro del scope del módulo.