

CREATING CLASSES AND OBJECTS IN RUBY


CLASS #4

AGENDA

- Everything is an object.
 - Numbers
 - Text
- Ruby Core Classes – objects with methods
 - Objects respond to messages – messages are method calls on the object
- Creating our own classes
- Using testing to help write lean clean classes
 - Write only enough code to make the test pass.



PROCEDURAL VS OBJECT-ORIENTED PROGRAMMING

- Procedural programming
 - Break down a programming task into variables, data structures and functions or subroutines.
 - Focus is on “proceeding” through the code statements
 - Procedural programming uses statements to operate on data structures
 - Object Oriented Programming
 - Break down a programming task into objects.
 - Objects represent abstractions of real-world objects
 - Objects bundle procedures and data together, so that an object operates on its own data.
 - Messages are set to objects; objects then respond to the messages by acting on their own data.
- 

RUBY IS OBJECT ORIENTED

- ***Everything*** is an object!
 - Fill in the blank using irb:
 - 1.class = _____
 - 0.0.class = _____
 - true.class = _____
 - false.class = _____
 - nil.class = _____



RUBY CORE CLASSES ARE OBJECTS

Home Classes Methods

In Files

- complex.c
- pack.c
- rational.c
- string.c
- transcode.c

Parent

Object

Methods

String

A `String` object holds and manipulates an arbitrary sequence of bytes, typically representing characters. `String` objects may be created using `String::new` or as literals.


Because of aliasing issues, users of strings should be aware of the methods that modify the contents of a `String` object. Typically, methods with names ending in “!” modify their receiver, while those without a “!” return a new `String`. However, there are exceptions, such as `String#[]=`.

Public Class Methods

RUBY CORE CLASSES ARE OBJECTS

Home Classes Methods

In Files

 hash.c

Parent

Object

Methods

::[]

Hash

A Hash is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an Array uses integers as its index, a Hash allows you to use any object type.

Hashes enumerate their values in the order that the corresponding keys were inserted.

A Hash can be easily created by using its implicit form:

```
grades = { "Jane Doe" => 10, "Jim Doe" => 6 }
```

SENDING A MESSAGE TO A RUBY CLASS

- Sending a message to a Ruby class is as simple as calling a method from that class:
 - `"Hello, World!".tr("aeiou", "*")`
 - `tr` is the message/method
 - `[1,5,4,7,4,8,0].sort!`
 - `sort!` is the message
- We send a message to the Ruby object by calling a method on the object. The object then acts on its own data.



CREATING OUR OWN CLASSES

We are going to create a simple Rectangle class.

- **What are the attributes of a rectangle?**
- **What kinds of messages should be sent to a rectangle?**



TEST DRIVEN DEVELOPMENT

- **A software development process:**
 - **Write an automated test case (that will fail).**
 - **Create just enough code for the test to pass.**
 - **Refactor the code**



RED-GREEN-REFACTOR!

- **Red:** the test fails.
- **Green:** the test passes because we have written just enough code.
- **Refactor:** consider ways to make your code clean and elegant.

RSPEC

Testing tool of choice for the Ruby programming language.

Domain Specific Language (DSL)

gem install rspec

CLASS RECTANGLE

Create a new file called "rectangle.rb"

Add the following line at the top of the file:

```
require "rspec"
```

At the command line, in the same directory as your rectangle.rb file, type:

```
rspec rectangle.rb
```

Among the errors you should see:

```
Finished in 0.00019 seconds (files took 0.05104 seconds to load)
0 examples, 0 failures
```

This means everything is hooked up and ready to go.



ADD A TEST

**Add the following Rspec statement to
rectangle.rb:**

```
describe Rectangle do
```

```
end
```

THE TEST FAILS

This time the error is near the top:

```
rspec rectangle.rb  
/Users/louiserains/source/rectangle.rb:3:in `<>top (required)>':  
uninitialized constant Rectangle (NameError)
```

The test is telling us that Rectangle hasn't been initialized.

ADD CODE TO CREATE A RECTANGLE CLASS

rectangle.rb now looks like this:

```
require 'rspec'
```

```
class Rectangle  
end
```

```
describe Rectangle do  
end
```

NO ERRORS, BUT NO EXAMPLES!

```
rspec rectangle.rb  
No examples found.
```

```
Finished in 0.00013 seconds (files took 0.08729 seconds to load)  
0 examples, 0 failures
```


ADD A TEST

Rectangles don't exist without a length and width:

```
describe Rectangle do
  subject { Rectangle.new(5,3) }

  context "should have length and width" do
    specify { expect(subject.length).to eq 5}
    specify { expect(subject.width).to eq 3}
  end
end
```

THE TEST FAILS

```
rspec rectangle.rb  
FF
```

Failures:

- 1) Rectangle should have length and width
Failure/Error: subject {Rectangle.new(5,3)}
ArgumentError:
wrong number of arguments (2 for 0)
./rectangle.rb:15:in `initialize'
./rectangle.rb:15:in `new'
./rectangle.rb:15:in `block (2 levels) in <top (required)>'
./rectangle.rb:18:in `block (3 levels) in <top (required)>'
- 2) Rectangle should have length and width
Failure/Error: subject {Rectangle.new(5,3)}
ArgumentError:
wrong number of arguments (2 for 0)
./rectangle.rb:15:in `initialize'
./rectangle.rb:15:in `new'
./rectangle.rb:15:in `block (2 levels) in <top (required)>'
./rectangle.rb:19:in `block (3 levels) in <top (required)>'

```
Finished in 0.00123 seconds (files took 0.09648 seconds to load)  
2 examples, 2 failures
```

Failed examples:

```
rspec ./rectangle.rb:18 # Rectangle should have length and width  
rspec ./rectangle.rb:19 # Rectangle should have length and width
```

READ THE ERRORS!

The errors drive the code that we write - they tell us what is wrong or missing.

ADD JUST ENOUGH CODE SO THE TEST PASSES

- We need to know some additional facts about classes:
 - The `initialize()` method is called when we send the 'new' message to a class
 - message: 'new'
 - method: 'initialize'
 - `Rectangle.new(5,3)` calls the initialize method
 - We need to write an initialize method.
 - Classes can have instance variables such as `length` and `width`.

THE RECTANGLE INITIALIZE METHOD

```
def initialize(length,width)
  @length = length
  @width = width
end
```

Using the @ sign in front of a variable name makes it an instance variable.

Every Rectangle now has two instance variables: @length and @width

THE TEST FAILS, BUT WITH A DIFFERENT ERROR!

```
rspec rectangle.rb  
FF
```

Failures:

1) Rectangle should have length and width

Failure/Error: specify {expect(subject.length).to eq 5}

NoMethodError:

undefined method `length' for #<Rectangle:0x007f9a321c3b98 @length=5, @width=3>

./rectangle.rb:18:in `block (3 levels) in <top (required)>'

2) Rectangle should have length and width

Failure/Error: specify {expect(subject.width).to eq 3}

NoMethodError:

undefined method `width' for #<Rectangle:0x007f9a321b3ba8 @length=5, @width=3>

./rectangle.rb:19:in `block (3 levels) in <top (required)>'

Finished in 0.0013 seconds (files took 0.0994 seconds to load)

2 examples, 2 failures

Failed examples:

```
rspec ./rectangle.rb:18 # Rectangle should have length and width
```

```
rspec ./rectangle.rb:19 # Rectangle should have length and width
```

ANOTHER FACT

We need methods to reveal the values of length and width.

```
class Rectangle
  def length
    @length
  end

  def width
    @width
  end
end
```

TO RETURN OR NOT?

- We normally leave out the return statement in a method unless we need to return something conditionally
- Example: the Fizz Buzz problem.

Write a program that prints the numbers from 1 to 100. But for multiples of 3 print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiple of both 3 and 5, print “FizzBuzz”.

```
def fizz_buzz(number)
  return "FizzBuzz" if number % 5 == 0 && number % 3 == 0
  return "Fizz" if number % 3 == 0
  return "Buzz" if number % 5 == 0
  number
end
```


FIZZ-BUZZ

```
(1..100).each{|n| puts fizz_buzz(n)}  
1  
2  
Fizz  
4  
Buzz  
Fizz  
7  
8  
Fizz  
Buzz  
11  
Fizz  
13  
14  
FizzBuzz  
16  
17  
Fizz  
19  
Buzz  
Fizz  
22  
...
```

THE TESTS PASS - GREEN!

```
rspec rectangle.rb
```

```
..
```

```
Finished in 0.00186 seconds (files took 0.10705 seconds to load)  
2 examples, 0 failures
```

REFACTOR

Convenience notation for the length and width methods:

```
attr_reader :length, :width
```

:length and :width are Ruby *Symbols*

<http://www.ruby-doc.org/core-2.1.2/Symbol.html>

Symbols are like immutable Strings.

```
require 'rspec'

class Rectangle
  attr_reader :length, :width

  def initialize(length,width)
    @length = length
    @width = width
  end
end

describe Rectangle do
  subject {Rectangle.new(5,3)}

  context "should have length and width" do
    specify {expect(subject.length).to eq 5}
    specify {expect(subject.width).to eq 3}
  end
end
```

OUR NEW CLEAN LEAN CLASS

THE TESTS STILL PASS AFTER REFACTORING!

```
rspec rectangle.rb
```

```
..
```

```
Finished in 0.00186 seconds (files took 0.10705 seconds to load)  
2 examples, 0 failures
```

WHAT NEXT?

- How about an area method?
- Write the test first!
 - Add a new test to our existing file

```
context "area" do  
  specify { expect(subject.area).to eq 15 }  
end
```



THE NEW TEST FAILS - RED!

```
rspec rectangle.rb  
..F
```

Failures:

1) Rectangle area

Failure/Error: specify { expect(subject.area).to eq 15 }

NoMethodError:

undefined method `area' for #<Rectangle:0x007fa0ec8cc4b0 @length=5, @width=3>
./rectangle.rb:22:in `block (3 levels) in <top (required)>'

Finished in 0.00121 seconds (files took 0.09798 seconds to load)
3 examples, 1 failure

Failed examples:

```
rspec ./rectangle.rb:22 # Rectangle area
```

DO JUST ENOUGH TO GET THE TEST TO PASS

- **Write a new method in the Rectangle class that calculates the area.**
 - **What is a good name for our method?**
 - **What should it do?**



THE NEW TEST PASSES - GREEN!

```
rspec rectangle.rb  
...
```

```
Finished in 0.00123 seconds (files took 0.10386 seconds to load)  
3 examples, 0 failures
```

RUNNING JUST OUR NEW TEST

```
>>rspec rectangle.rb:25
```

```
Run options: include {:locations=>{"../rectangle.rb"=>[25]}}
```

```
.
```

```
Finished in 0.00086 seconds (files took 0.09914 seconds to load)
```

```
1 example, 0 failures
```

```
25   context "area" do
26     specify { expect(subject.area).to eq 15 }
27   end
```

REFACTOR?

A method should only do one task for our class.

It should be clear and expressive.

TRY THIS

- Using **Red**, **Green**, Refactor, write a method for calculating the perimeter of our rectangle
 - Write the test first!

AND THIS

- Using **Red**, **Green**, Refactor, write a method called 'square?' which returns true if @length == @width and false otherwise.
- Write the test first!

SUMMARY

- Using Test Driven Development - the **Red**, **Green**, Refactor cycle - helps write lean and clean classes.
- rspec is the tool of choice for testing Ruby classes.
- Write tests for a specific instance of the class.
 - set specific values for class attributes
- Write methods in the class that work for any instance of the class.
 - use instance variables to represent class attributes
- Write just enough code to get the tests to pass.

HOMEWORK

- Using Test Driven Development, create a class: **RealEstateListing**.
 - The file name should be **real_estate_listing.rb**
 - Include **rspec** as in the class example.
 - Consider testing and adding some of the following attributes:
 - **listing_type** (townhouse, condo, apartment, house...)
 - **street_address**
 - **city**
 - **zip_code**
 - **number_of_bedrooms**
 - **number_of_baths**
 - **square_feet**
- Email your solutions to: Louise.Rains@meyouhealth.com



APPENDIX

Juicy Ruby Tips

WHAT'S NEXT?

- [Ruby in 20 Minutes Tutorial](#)
- [Ruby Koans](#)
 - Test-driven exercises which walk through the Ruby core classes, teaching you the language as you go.
- [CodeKata](#)
 - Exercise to review and practice test-driven skills with Ruby classes
- [CodeAcademy](#)
- [Launch Academy](#)

- [Agile Web Development with Rails](#)
- Michael Hartl's [Ruby on Rails Tutorial](#)
- [Thoughtbot's Upcase](#)

