Self-testing Code in Ruby

Giovanni Sakti

Starqle

What is Self-testing code?

Self-testing code

- Code that have built-in tests
- The tests serve as a binding contract
- The tests can be run arbitrarily

What is TDD? How it differs from self-testing code?

TDD

- Practices of writing **tests** before the code
- Ensure that the code is self-tested
- It is, however, **optional** to do TDD to write self-testing code

TDD

But some companies enforce TDD because TDD enforces **YAGNI** principle

TDD

We'll see why...

TDD Steps

- Write a test
- Run the test, it should fail
- Write code just enough to pass the test
- Run the test
- Repeat

TDD & YAGNI

Because we only write just enough code to pass the test, there will be no unnecessary codes

Test in Ruby

There are several tools for doing testing in ruby

Test in Ruby

- RSpec
- Minitest
- test-unit

Test in Ruby

Let's try using RSpec

RSpec Install

```
% gem install rspec
```

RSpec Help

```
% rspec --help
```

Now let's do TDD practice using RSpec

TDD with RSpec (1)

Create a simple test of program that we want to create

```
# game_spec.rb

RSpec.describe Game do
  describe "#score" do
    it "returns 0 for new game" do
      game = Game.new
      expect(game.score).to eq(0)
    end
  end
end
```

TDD with RSpec (2)

Run the example and watch it fail

```
% rspec game_spec.rb
  uninitialized constant Object::Game (NameError)
```

TDD with RSpec (3)

Now write just enough code to make it pass

```ruby
# game.rb

class Game
  attr_reader :score

  def initialize
    @score = 0
  end
end
```

TDD with RSpec (3) cont'd

Now write just enough code to make it pass

```
# game_spec.rb

require './game'
...
```

TDD with RSpec (4)

Run the example and the test shall pass

```
% rspec game_spec.rb --color --format doc

Game
  #score
    returns 0 for all gutter game

Finished in 0.00057 seconds
1 example, 0 failures
```

TDD with RSpec (5)

Repeat with new features

Some important RSpec APIs

Basic Matchers

```ruby
# equality
expect('x'+'y').to eq('xy')       # a == b
expect('x'+'y').to eql('xy')      # a.eql?(b)
expect('x'+'y').not_to be('xy')   # a.equal?(b)

# strings
expect('abcd').to include('bc')
expect('abcd').to start_with 'ab'
expect('abcd').to end_with 'cd'
expect('abcd').to match /[a-z]+/

# collections
expect([1, 2, 3]).to include(1, 3)
expect([1, 2, 3]).to contain_exactly(3, 2, 1) # order not important
expect({ a: 1, b: 2 }).to include(b: 2)
```

Basic Matchers cont'd

```ruby
# booleans and nil
expect(true).to be true
expect(false).to be false
expect('abc').to be_truthy
expect(nil).to be_falsey
expect(nil).to be_nil

# numeric
expect(5).to be > 4
expect(5).to be >= 4
expect(5).to be < 6
expect(5).to be <= 6
expect(5).to be_between(4, 6).exclusive
expect(5).to be_between(5, 6).inclusive
expect(4.99).to be_within(0.02).of(5)

# errors (exceptions)
expect{ 5 / 0 }.to raise_error(ZeroDivisionError)
expect{ 5 / 0 }.to raise_error("divided by 0")
expect{ 5 / 0 }.to raise_error(ZeroDivisionError, "divided by 0")
```

Predicate Matchers

Predicate matchers are a little DSL for calling predicate methods. Predicate methods are methods that:

- return a boolean value; and
- have a name that ends with ?

Predicate Matchers cont'd

```
# array
expect([]).to be_empty          # [].empty?

# hash
expect({a: 1}).to have_key(:a)   # {a: 1}.has_key?(:a)
expect({a: 1}).to have_value(1)  # {a: 1}.has_value?(1)

# object
expect(5).not_to be_nil             # 'hi'.nil?
expect(5).to be_instance_of Fixnum  # 5.instance_of?(Fixnum)
expect(5).to be_kind_of Numeric     # 5.kind_of?(Numeric)
```

Predicate Matchers cont'd

Predicate matchers work on all objects, including custom classes

Now let's do some exercises...

TDD Exercises (1)

Create a Sentence from Words

Without using "Array#each" iterator, create a method that will return a sentence when given an array of words.

```
create_sentence(["hello", "world"])
# will return: "hello world"
```

TDD Exercises (2)

Find Palindromes

Write a method that receives two positive integers "m" and "n" and returns an array of "n" palindrome numbers after "m" (including "m" itself).

```
find_palindrome(100, 2)
# will return [101, 111]

find_palindrome(22, 3)
# will return [22, 33, 44]
```

TDD Exercises (3)

Descending Order

Create a method that receives an integer as an argument and rearrange it to generate biggest possible value.

```
descending(21445) # will return 54421
descending(145263) # will return 654321
descending(1254859723) # will return 9875543221
```

TDD Exercises (4)

Deep Count

Create a method called deep_count that will return the number of elements in an array, including the number of elements of its sub arrays.

TDD Exercises (4) cont'd

```
deep_count([]) # will return 0
deep_count([1, 2, 3]) # will return 3

deep_count(["x", "y", ["z"]])
# will return 3 elements ("x", "y", ["z"]) in main array
# plus 1 element ("z") in sub array
# total = 4 elements

deep_count([1, 2, [3, 4, [5]]])
# total = 7 elements

deep_count([[[[[[[[[]]]]]]]]])
# total = 8 elements
```

TDD Exercises (5)

Letter Count

Create a method that receives a string as its argument and returns a hash that shows the number of occurrence of each letter in that string.

TDD Exercises (5) cont'd

```
letter_count("gojek")
# will return {:g=>1, :o=>1, :j=>1, :e=>1, :k=>1}

letter_count("kolla")
# will return {:k=>1, :o=>1, :l=>2, :a=>1}

letter_count("scholarship")
# will return {:s=>2, :c=>1, :h=>2, :o=>1, :l=>1, :a=>1, :r=>1, :i=>1, :p=>1}
```

Thanks