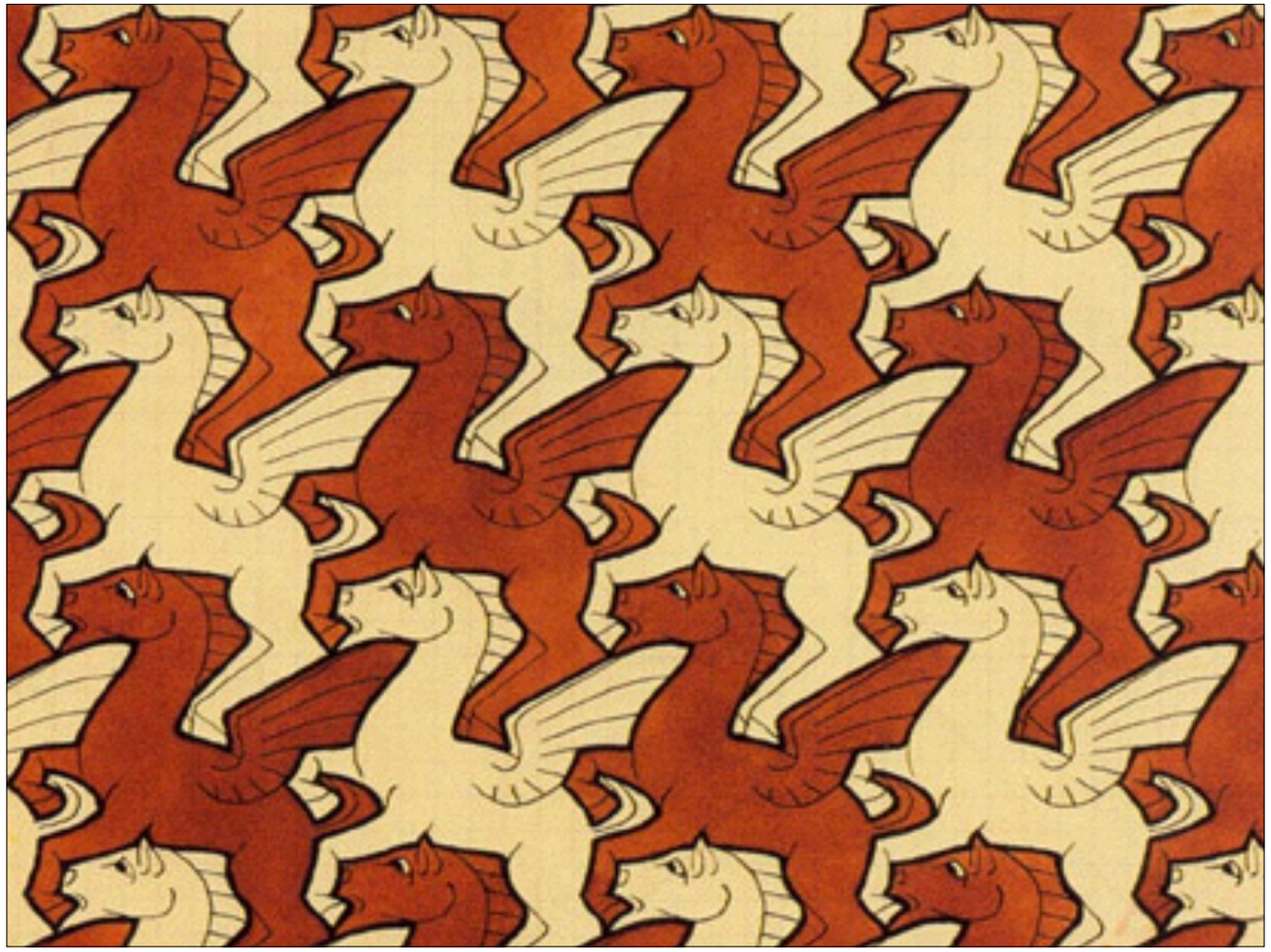
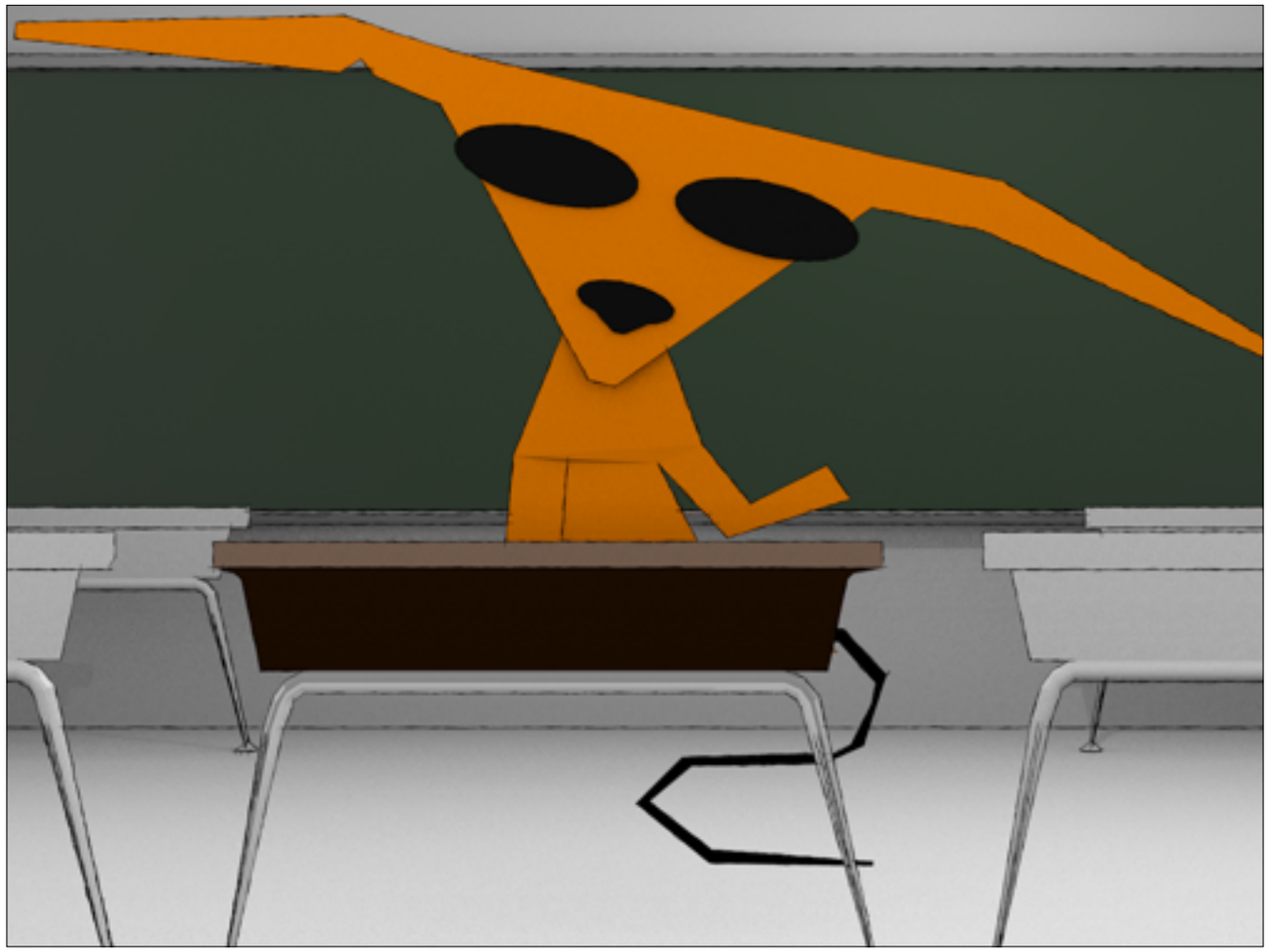


Does anyone like repeating themselves? Yeah, me neither. Repetition can be quite nice in art….



and it is super useful in learning….

But when you’re writing code, repetition is the worst! Nobody has time for that. Well, maybe this guy…..

Imagine you've been hired to help a local animal preserve write software that will improve operations. They need to track vehicles, employees and hyenas.

```ruby
class Employee
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def pay
    "Wooohooooo!"
  end
end
```

```ruby
class Hyena
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def feed(food)
    if food == "crunchy bones"
      "yipyipyip"
    else
      "grrrrrrrr"
    end
```

```ruby
class Vehicle
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def fuel
    "Vroooooooom!"
  end
end
```

So you make a class for each and realize - to your horror - that your classes are FILLED with repetitive code.

vehicle.rb

```ruby
module Memorable
  module ClassMethods
    def self.extended(base)
      base.class_eval do
        @all = []
      end
    end

    def all
      @all
    end

    def reset_all
      self.all.clear
    end

    def count
      @all.count
    end
  end

  module InstanceMethods
    def initialize
      self.class.all << self
      super
    end
  end
end
```

Type:
Module

Code Name:
Memorable

Secret Power:
Saves the World from Repetitive Code

So you bust out this guy - the module - and you move all of the repeat code to him.

employee.rb

```ruby
class Employee
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def pay
    "Wooohooooo!"
  end
end
```

hyena.rb

```ruby
class Hyena
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def feed(food)
    if food == "crunchy bones"
      "yipyipyip"
    else
      "grrrrrrrr"
    end
```

vehicle.rb

```ruby
class Vehicle
  attr_accessor :all
  @all = []

  def initialize
    self.class.all << self
  end

  def self.all
    @all
  end

  def self.reset_all
    self.all.clear
  end

  def self.count
    @all.count
  end

  def fuel
    "Vroooooooom!"
  end
end
```

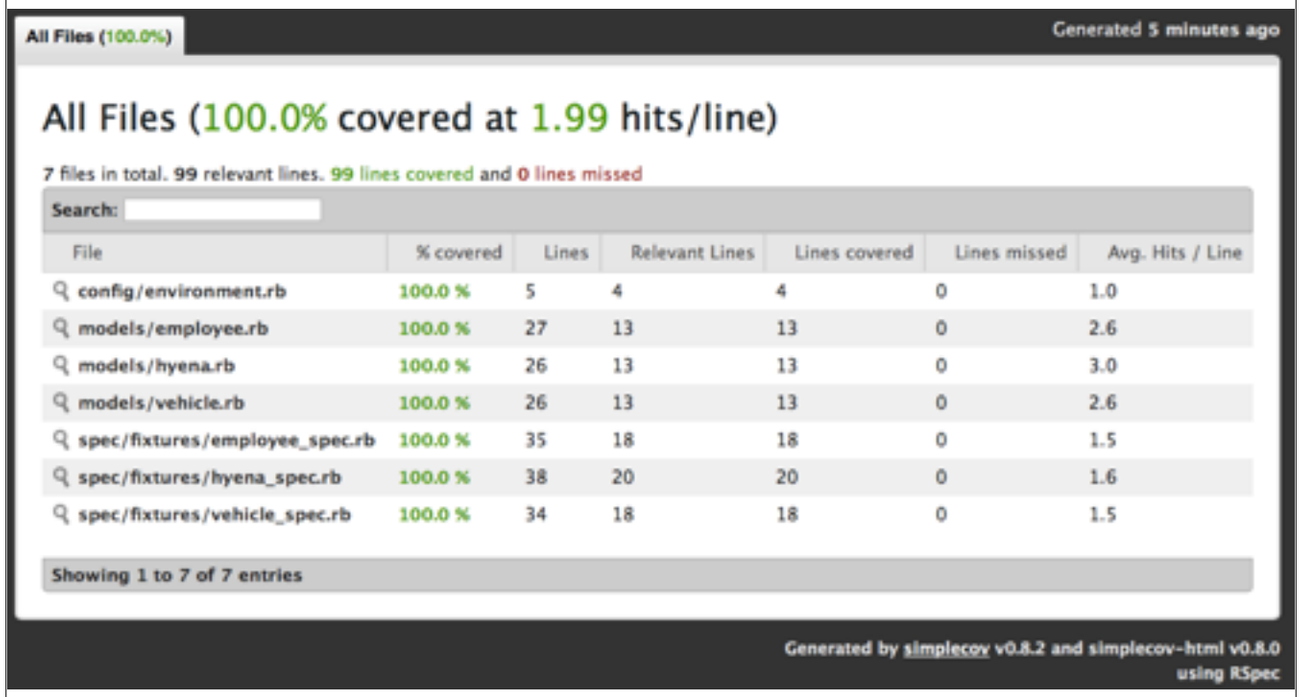
Transforming this…..

```ruby
require_relative '../models/co

class Employee
  extend Memorable::ClassMetho
  prepend Memorable::Instance

  def pay
    "Wooohooooo!"
  end
end
```

```ruby
require_relative '../models/co

class Hyena
  extend Memorable::ClassMetho
  prepend Memorable::Instance

  def feed(food)
    if food == "crunchy bones"
      "yipyipyip"
    else
      "grrrrrrrrr"
    end
  end
end
```

```ruby
require_relative '../models/cor

class Vehicle
  extend Memorable::ClassMethoc
  prepend Memorable::InstanceMe

  def fuel
    "Vroooooooooom!"
  end
end
```

into this!!!



All Files (100.0%)

Generated 5 minutes ago

All Files (100.0% covered at 1.99 hits/line)

7 files in total. 99 relevant lines. 99 lines covered and 0 lines missed

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
config/environment.rb	100.0 %	5	4	4	0	1.0
models/employee.rb	100.0 %	27	13	13	0	2.6
models/hyena.rb	100.0 %	26	13	13	0	3.0
models/vehicle.rb	100.0 %	26	13	13	0	2.6
spec/fixtures/employee_spec.rb	100.0 %	35	18	18	0	1.5
spec/fixtures/hyena_spec.rb	100.0 %	38	20	20	0	1.6
spec/fixtures/vehicle_spec.rb	100.0 %	34	18	18	0	1.5

Showing 1 to 7 of 7 entries

Generated by simplecov v0.8.2 and simplecov-html v0.8.0 using RSpec

You run your spec tests and BOOM! You have 100% coverage….but something's still bothering you……you got rid of the repetition in your code, but

```ruby
# employee_spec.rb
require_relative '../spec_

describe Employee do
  before(:each) do
    Employee.reset_all
  end

  let!(:employee){Employee

  describe "Class methods"
    it "keeps track of the
      expect(Employee.all)
    end

    it "can count how many
      expect(Employee.coun
    end

    it "can reset the empl
      Employee.reset_all
      expect(Employee.coun
    end
  end

  describe "Instance metho
    it "can be initialized
      expect(employee).to
    end
```

```ruby
# hyena_spec.rb
require_relative '../spec_

describe Hyena do
  before(:each) do
    Hyena.reset_all
  end

  let!(:hyena){Hyena.new}

  describe "Class methods"
    it "keeps track of the
      expect(Hyena.all).to
    end

    it "can count how many
      expect(Hyena.count).
    end

    it "can reset the empl
      Hyena.reset_all
      expect(Hyena.count).
    end
  end

  describe "Instance metho
    it "can be initialized
      expect(hyena).to be_
    end
```

```ruby
# vehicle_spec.rb
require_relative '../spec_h

describe Vehicle do
  before(:each) do
    Vehicle.reset_all
  end

  let!(:vehicle){Vehicle.ne

  describe "Class methods" d
    it "keeps track of the
      expect(Vehicle.all).t
    end

    it "can count how many
      expect(Vehicle.count)
    end

    it "can reset the emplo
      Vehicle.reset_all
      expect(Vehicle.count)
    end
  end

  describe "Instance method
    it "can be initialized"
      expect(vehicle).to be
    end
```

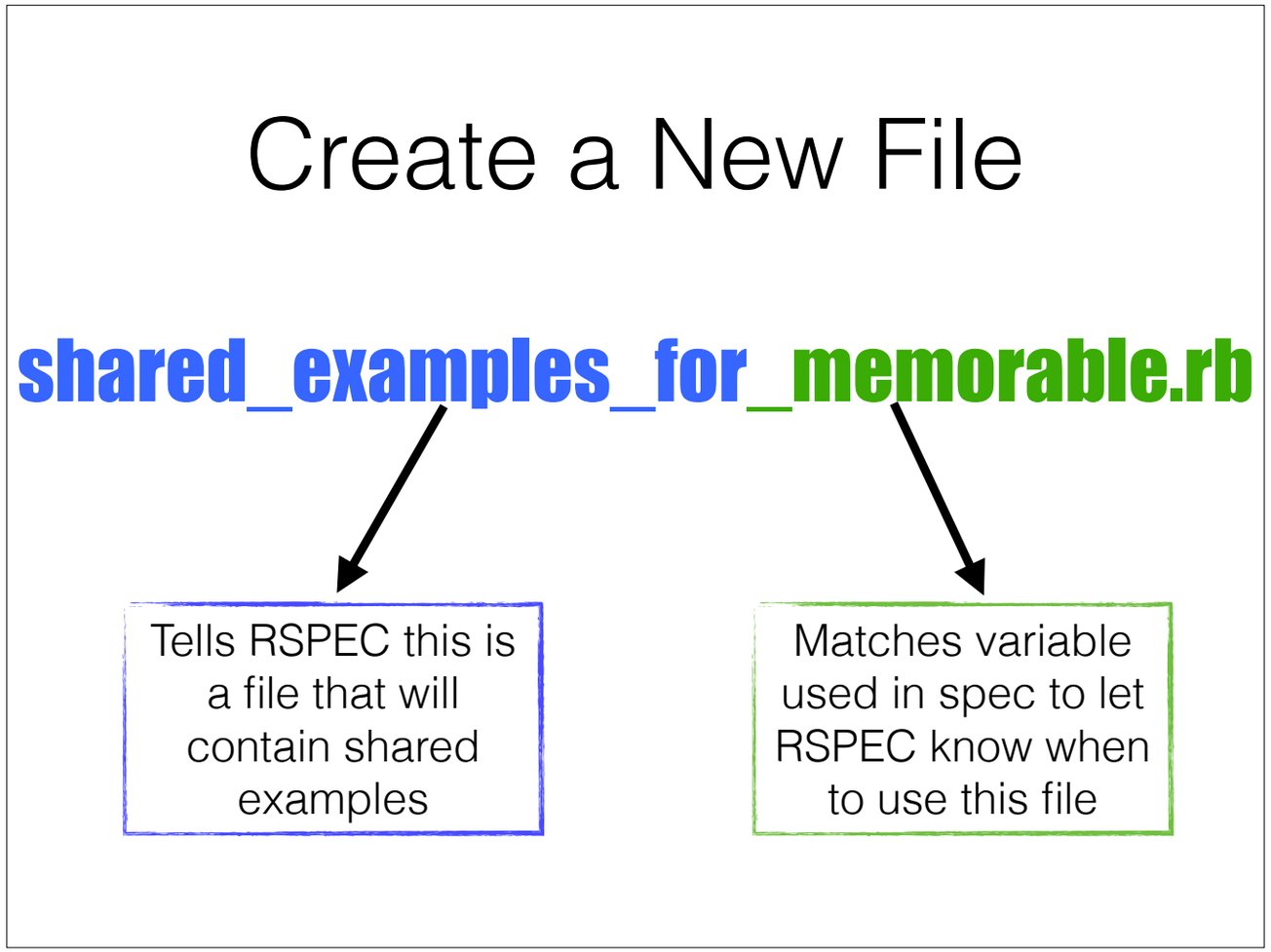
…your RSPEC tests are still full of line after line of repetitive code. Wouldn't it be great if there were a way to avoid repeating yourself in your test code???

Keeping RSPEC Tests DRY

The DRYist Talk You Will Hear Today

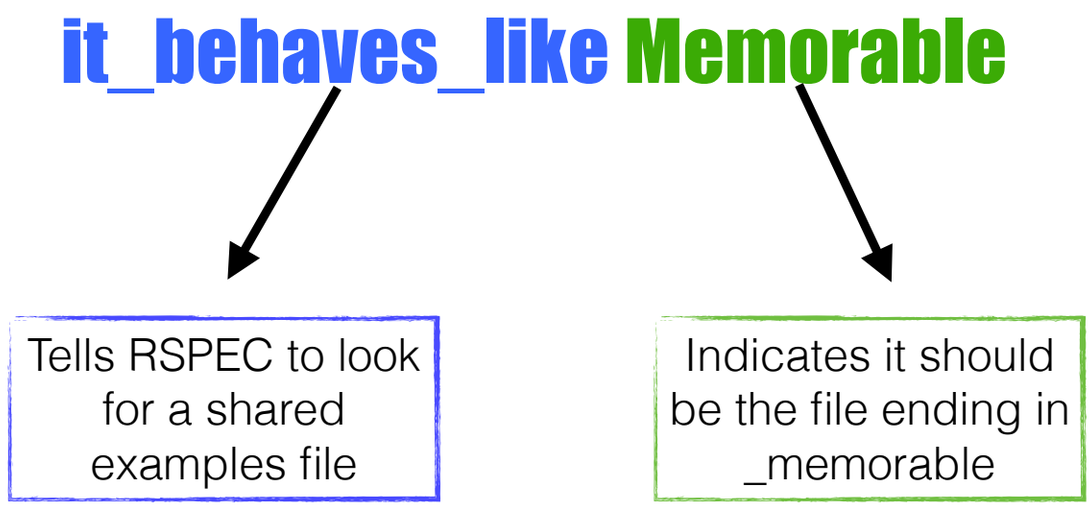
By: David Coronado and Jessica Rudder

There is! And we're going to teach you how to do it.



The first step is the create a new file. The name is important here. You need to start the file name with shared_examples_for and then indicate which tests are in this file - in this case, the Memorable tests. Memorable will now be a variable you can use in your other spec files to indicate these examples should be applied.

Add Line to Class Spec File



Next add it_behaves_like Memorable to the class-specific spec file. During testing, when RSPEC reaches this line, it will jump to a file named "shared_examples_for_memorable.rb" and start running the tests in that file.

Move Memorable Tests to New File

```ruby
describe Hyena do
  before(:each) do
    Hyena.reset_all
  end

  let!(:hyena){Hyena.new}

  describe "Class methods" do
    it "keeps track of the hyenas that have been created" do
      expect(Hyena.all).to include(hyena)
    end

    it "can count how many hyenas have been created" do
      expect(Hyena.count).to eq(1)
    end

    it "can reset the hyenas that have been created" do
      Hyena.reset_all
      expect(Hyena.count).to eq(0)
    end
  end

  describe "Instance methods" do
    it "can be initialized" do
      expect(hyena).to be_an_instance_of(Hyena)
    end
  end
end
```

Next move all the tests that are duplicated between classes into the shared_examples file. You're almost there, but first you need to make these tests less specific. Otherwise you'll keep testing the hyena class over and over.

Update to Implicit Subject

```
describe Hyena do
  before(:each) do
    described_class.reset_all
  end

  let!(:hyena) {described_class.new}

  describe "Class methods" do
    it "keeps track of the hyenas that have been created" do
      expect(described_class.all).to include(hyena)
    end

    it "can count how many hyenas have been created" do
      expect(described_class.count).to eq(1)
    end

    it "can reset the hyenas that have been created" do
      described_class.reset_all
      expect(described_class.count).to eq(0)
    end
  end

  describe "Instance methods" do
    it "can be initialized" do
      expect(hyena).to be_an_instance_of(described_class)
    end
  end
end
```

Change to an implicit subject using “described_class”

Replace all references to a specific class (in this case, Hyena) with “described_class”. This allows the class being referenced to vary depending on which class is being tested at that point in time.

Update Variable Names

```
describe Hyena do
  before(:each) do
    described_class.reset_all
  end

  let!(:memorable) {described_class.new}

  describe "Class methods" do
    it "keeps track of the hyenas that have been created" do
      expect(described_class.all).to include(memorable)
    end

    it "can count how many hyenas have been created" do
      expect(described_class.count).to eq(1)
    end

    it "can reset the hyenas that have been created" do
      described_class.reset_all
      expect(described_class.count).to eq(0)
    end
  end

  describe "Instance methods" do
    it "can be initialized" do
      expect(memorable).to be_an_instance_of(described_class)
    end
  end
end
```

Change from specific “:hyena” to “:memorable”

Update the variable names so it’s no longer class specific. Obviously this is not required as variables can be anything, but it will help make the code more readable.

Update Your Test Labels

```ruby
shared_examples_for Memorable do
  before(:each) do
    described_class.reset_all
  end

  let!(:memorable) {described_class.new}

  describe "Memorable class methods" do
    it "keeps track of the instances that have been created" do
      expect(described_class.all).to include(memorable)
    end

    it "can count how many instances have been created" do
      expect(described_class.count).to eq(1)
    end

    it "can reset the instances that have been created" do
      described_class.reset_all
      expect(described_class.count).to eq(0)
    end
  end

  describe "Memorable instance methods" do
    it "can be initialized" do
      expect(memorable).to be_an_instance_of(described_class)
    end
  end
end
```

Make it clear that these are tests for Memorable and be sure to remove any class specific references from the test labels

Make sure the tests are labeled accurately. These tests are for Memorable, not just hyena.

```ruby
require_relative './spec_helper'

shared_examples_for Memorable do
  before(:each) do
    described_class.reset_all
  end

  let!(:memorable) {described_class.new}

  describe "Memorable class methods" do
    it "keeps track of the instances that have been created" do
      expect(described_class.all).to include(memorable)
    end

    it "can count how many instances have been created" do
      expect(described_class.count).to eq(1)
    end

    it "can reset the instances that have been created" do
      described_class.reset_all
      expect(described_class.count).to eq(0)
    end
  end

  describe "Memorable instance methods" do
    it "can be initialized" do
      expect(memorable).to be_an_instance_of(described_class)
    end
  end
end
```

Type:
Shared Examples

Code Name:
shared_examples_for_memorable

Secret Power:
Multiplies the Power of 1 Test Across Infinite Classes

And now, thanks to this shared example…

employee_spec.rb

```
1  require_relative '../spec_
2
3  describe Employee do
4    before(:each) do
5      Employee.reset_all
6    end
7
8    let!(:employee){Employee
9
10   describe "Class methods"
11     it "keeps track of the
12       expect(Employee.all)
13     end
14
15     it "can count how many
16       expect(Employee.coun
17     end
18
19     it "can reset the empl
20       Employee.reset_all
21       expect(Employee.coun
22     end
23   end
24
25   describe "Instance metho
26     it "can be initialized
27       expect(employee).to
28     end
```

hyena_spec.rb

```
1  require_relative '../spec_
2
3  describe Hyena do
4    before(:each) do
5      Hyena.reset_all
6    end
7
8    let!(:hyena){Hyena.new}
9
10   describe "Class methods"
11     it "keeps track of the
12       expect(Hyena.all).to
13     end
14
15     it "can count how many
16       expect(Hyena.count).
17     end
18
19     it "can reset the empl
20       Hyena.reset_all
21       expect(Hyena.count).
22     end
23   end
24
25   describe "Instance metho
26     it "can be initialized
27       expect(hyena).to be_
28     end
```

vehicle_spec.rb

```
1  require_relative '../spec_h
2
3  describe Vehicle do
4    before(:each) do
5      Vehicle.reset_all
6    end
7
8    let!(:vehicle){Vehicle.ne
9
10   describe "Class methods" d
11     it "keeps track of the
12       expect(Vehicle.all).t
13     end
14
15     it "can count how many
16       expect(Vehicle.count)
17     end
18
19     it "can reset the emplo
20       Vehicle.reset_all
21       expect(Vehicle.count)
22     end
23   end
24
25   describe "Instance method
26     it "can be initialized"
27       expect(vehicle).to be
28     end
```

This mess of code…

```ruby
require_relative '../fixtures

describe Employee do
  it_behaves_like Memorable

  before(:each) do
    Employee.reset_all
  end

  let!(:employee){Employee.ne

  describe "Instance methods"
    it "can be payed" do
      expect(employee.pay).to
    end
  end

end
```

```ruby
require_relative '../fixtures

describe Hyena do
  it_behaves_like Memorable

  before(:each) do
    Hyena.reset_all
  end

  let!(:hyena){Hyena.new}

  describe "Hyena instance me
    it "can be fed foods it d
      expect(hyena.feed("appl
    end
    it "can be fed foods it l
      expect(hyena.feed("crun
    end
  end
end
```

```ruby
require_relative '../fixtures/

describe Vehicle do
  it_behaves_like Memorable

  before(:each) do
    Vehicle.reset_all
  end

  let!(:vehicle){Vehicle.new}

  describe "Instance methods"
    it "can be fueled up" do
      expect(vehicle.fuel).to
    end
  end
end
```

…becomes this!

```
Employee
  behaves like Memorable
    Memorable class methods
      keeps track of the instances that have been created
      can count how many instances have been created
      can reset the instances that have been created
    Memorable instance methods
      can be initialized
  Instance methods
    can be payed

Hyena
  behaves like Memorable
    Memorable class methods
      keeps track of the instances that have been created
      can count how many instances have been created
      can reset the instances that have been created
    Memorable instance methods
      can be initialized
  Hyena instance methods
    can be fed foods it doesn't like
    can be fed foods it loves

Vehicle
  behaves like Memorable
    Memorable class methods
      keeps track of the instances that have been created
      can count how many instances have been created
      can reset the instances that have been created
    Memorable instance methods
      can be initialized
  Instance methods
    can be fueled up
```

By running RSPEC with the documentation on you can see that each class underwent the tests that were unique to their class as well as the tests for the memorable method in the shared_examples_for_memorable file.

All Files (100.0%) Generated 3 minutes ago

All Files (100.0% covered at 3.92 hits/line)

8 files in total. 77 relevant lines. 77 lines covered and 0 lines missed

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
config/environment.rb	100.0 %	8	6	6	0	1.0
models/concerns/memorable.rb	100.0 %	28	15	15	0	11.3
models/employee.rb	100.0 %	10	6	6	0	1.0
models/hyena.rb	100.0 %	14	8	8	0	1.1
models/vehicle.rb	100.0 %	10	6	6	0	1.0
spec/fixtures/hyena_spec.rb	100.0 %	20	11	11	0	2.0
spec/fixtures/support/shared_examples_for_memorable.rb	100.0 %	30	16	16	0	4.1
spec/fixtures/vehicle_spec.rb	100.0 %	17	9	9	0	2.0

Showing 1 to 8 of 8 entries

Generated by simplecov v0.8.2 and simplecov-html v0.8.0 using RSpec

Double check in coverage and BOOM! You're still at 100% test coverage but you achieved it with far fewer lines of code.

Leaving you with more time to hang out with this guy.

@davidjcoronado
@jessrudder

david.coronado@flatironschool.com
jessica.rudder@flatironschool.com