# BEWD
# Sharing Behaviour

Matt Heath
Tech Lead, Platform
Hailo

## AGENDA

- Reviewing Scope
- Sharing Code: Inheritance
- Sharing Code: Mixins
- Lab Time

# SCOPE

## METHOD SCOPE

```
class SuperHero
  def fly
    "Here we go!"
  end
end

def fly
  "I can't."
end

>> superman = SuperHero.new

>> superman.fly
=> "Here we go!"

>> fly
=> "I can't."
```

# SCOPE

## CLASS METHODS

- You don't need an instance to call a class method
- Below is an example of the SecretNumber class re-implemented to use a class method

```
class SecretNumber
  # gets a random number between 0-9
  # adds one so it's between 1-10
  def self.generate
    rand(10)+1
  end
end

>> number = SecretNumber.generate
```

# SCOPE

## KNOWING SELF

» self keyword is used when defining a method name to indicate a class method

» self is also used INSIDE a method definition to indicate the current object

» a common use of self is to call the current objects methods (such as one of its attr_accessors)

## SCOPE

### KNOWING SELF

```
class NewsPaper
  attr_accessor :stories

   def self.generate_random_story
     "This random event happened on day #{rand(28)} of
this month."
   end

   def add_story(story)
    # the below code is the same as: @stories << story
     self.stories << story
   end
end

>> story = NewsPaper.generate_random_story
=> "This random event happened on day 20 of this month."
>> paper = NewsPaper.new
>> paper.add_story(story)
>> paper.stories
=> ["This random event happened on day 20 of this month."]
```

# SHARING BEHAVIOUR

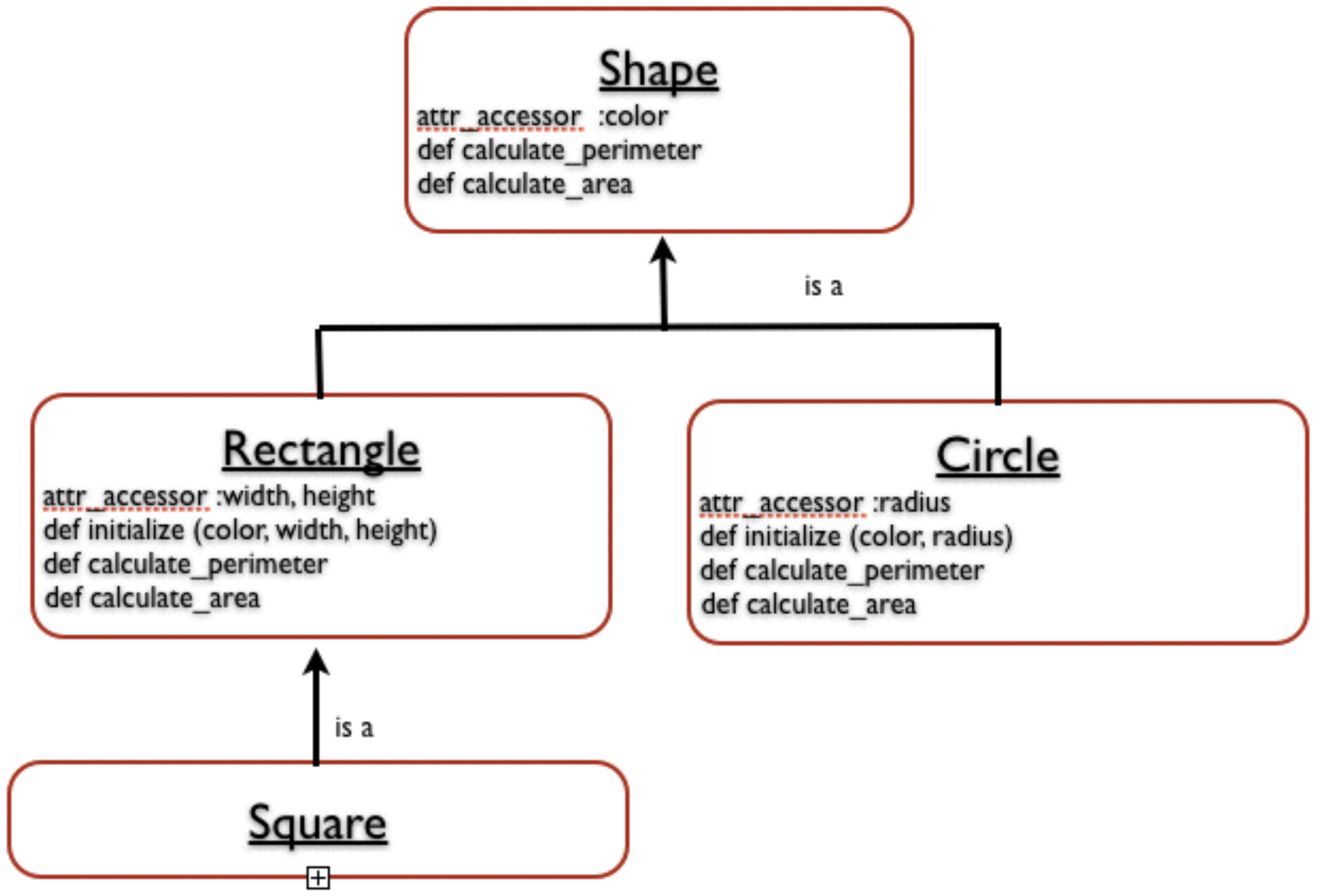
## SHARING IS CARING

- Inheritance
- Mixins
- Modules

## INHERITANCE

- Share properties & behaviour
- Keeps code DRY

# INHERITANCE



CODE ALONG

## Inheritance

# INHERITANCE

## GLIMPSE INTO RAILS

» Where you'll see it:

```
class User < ActiveRecord::Base

   # Interesting code…

end
```

# INHERITANCE

## RECAP

» One class can inherit the capabilities of another using the “ **<** ” operator.

» Sub-classes inherit from their super-class (child class inherits from parent class)

» A child can override a parent variable or method by re-using its name

» If defined in different physical files, a child must require its parent

## SHARING BEHAVIOUR

### GETTING READY FOR RAILS

- The following slides introduce other ways to share behavior.
- This is an introduction and we will see more when we start Rails.
- For now lets understand the basics.

## SHARING BEHAVIOUR

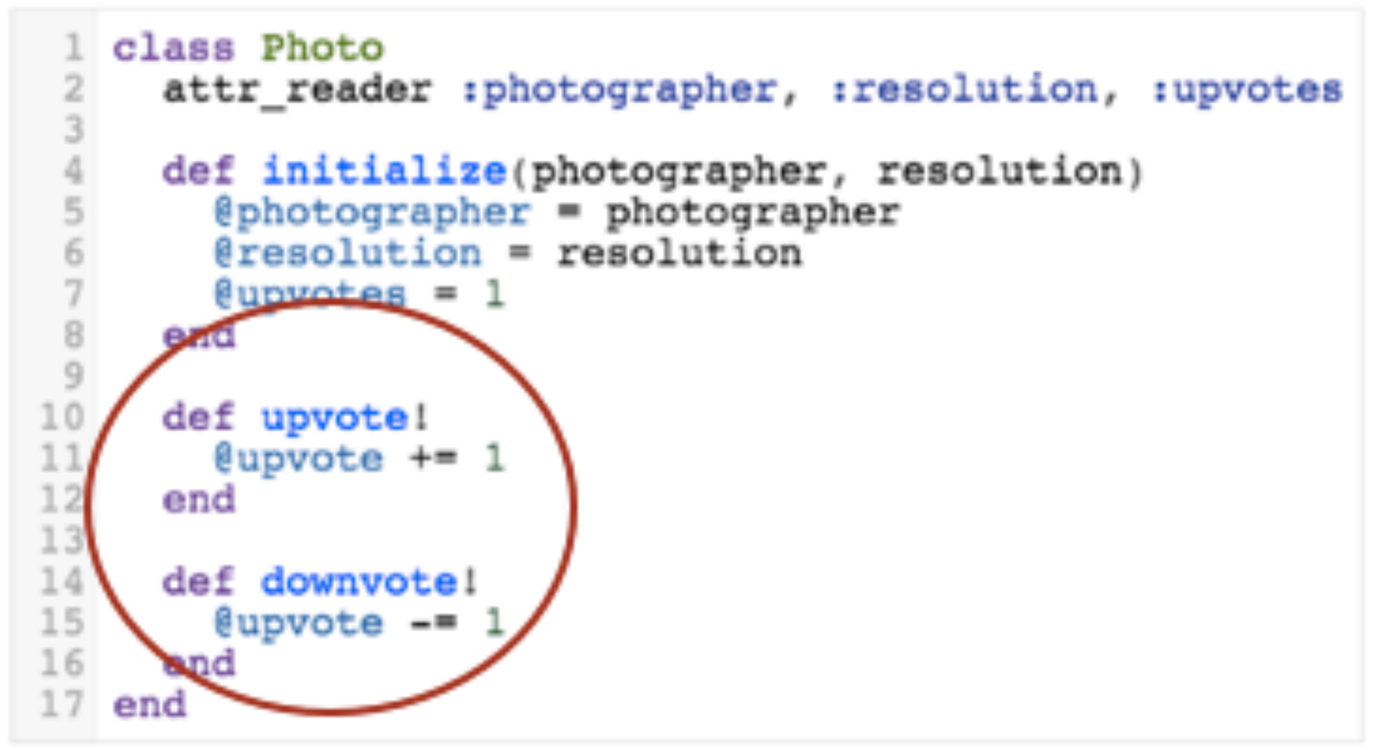
### MIXINS

- What if our classes don't have an "is a" relationship?
- "**Mixins**" are a facility to import code into a class
- Used in cases when we don't want to use inheritance
- Perhaps we only want a few methods from a small module, not the whole class
- A class may want to mixin many different modules, but you can only inherit from one class
- In Ruby, we use Modules to facilitate mixins

# MIXINS

## TEDDIT AS AN EXAMPLE

- Lets say teddit now accepts photos, videos and stories.
- You can up and down vote all of them.

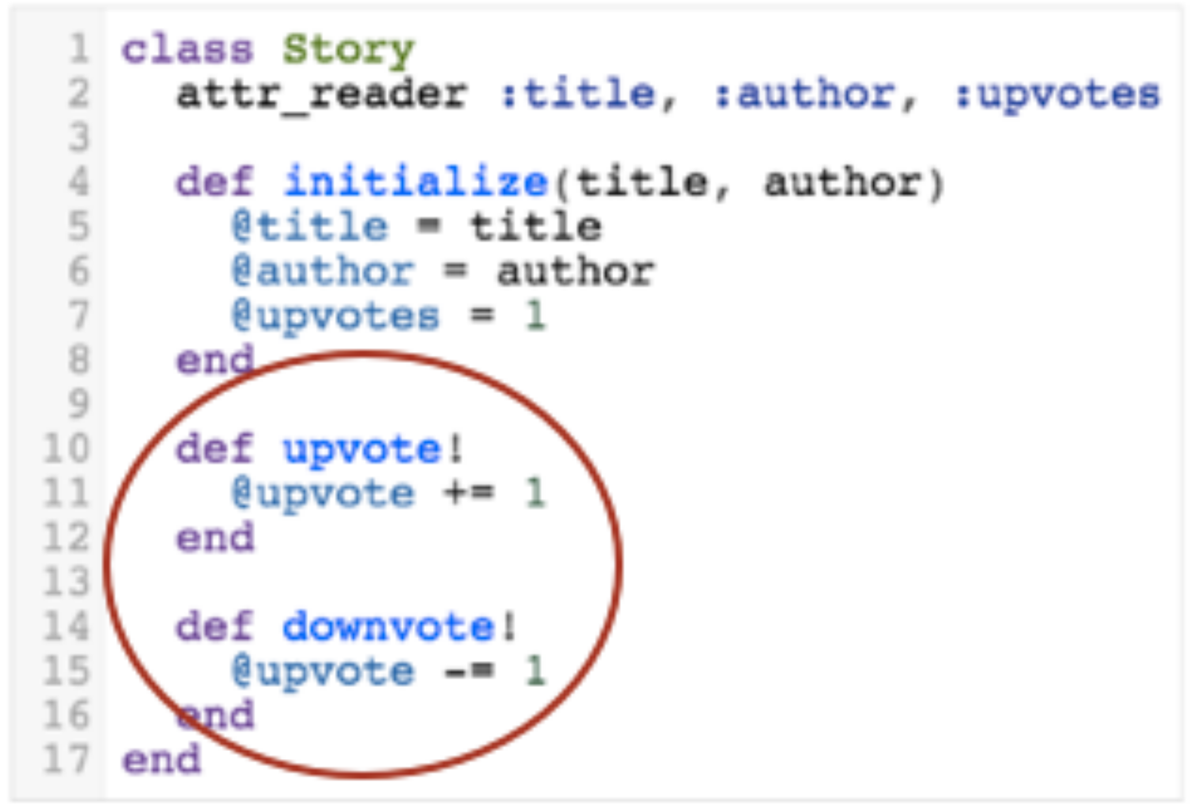


```ruby
class Photo
  attr_reader :photographer, :resolution, :upvotes

  def initialize(photographer, resolution)
    @photographer = photographer
    @resolution = resolution
    @upvotes = 1
  end

  def upvote!
    @upvote += 1
  end

  def downvote!
    @upvote -= 1
  end
end
```

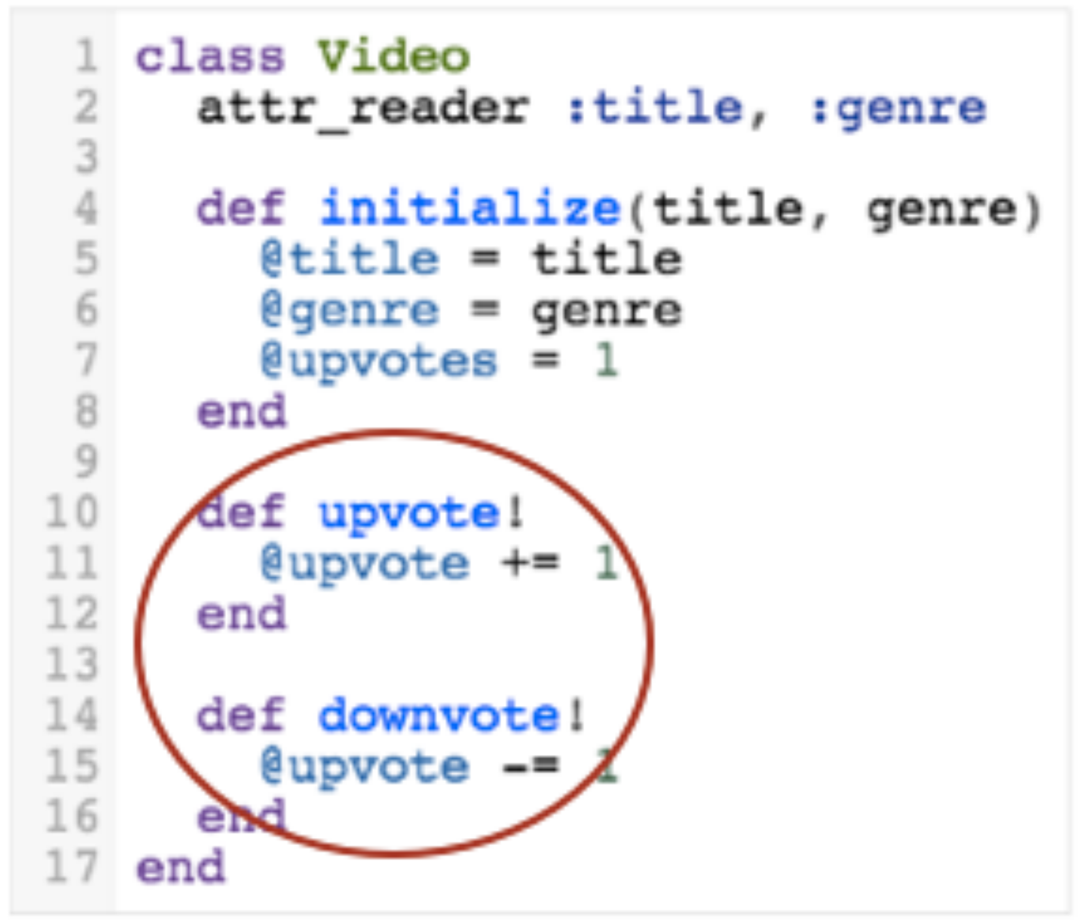


```ruby
class Story
  attr_reader :title, :author, :upvotes

  def initialize(title, author)
    @title = title
    @author = author
    @upvotes = 1
  end

  def upvote!
    @upvote += 1
  end

  def downvote!
    @upvote -= 1
  end
end
```



```ruby
class Video
  attr_reader :title, :genre

  def initialize(title, genre)
    @title = title
    @genre = genre
    @upvotes = 1
  end

  def upvote!
    @upvote += 1
  end

  def downvote!
    @upvote -= 1
  end
end
```

# MIXINS

## TEDDIT AS AN EXAMPLE

```
module Upvotable
    def upvote!
        @upvotes += 1
    end

    def downvote!
        @upvotes -= 1
    end
end
```

## MIXINS

### TEDDIT AS AN EXAMPLE

```
class Photo
    attr_reader :photographer, :resolution, :upvotes
    include Upvotable

    def initialize(photographer, resolution)
        @photographer = photographer
        @resolution = resolution
        @upvotes = 1
    end
end

class Story
    attr_reader :title, :author, :upvotes
    include Upvotable

    def initialize(title, author)
        @title = title
        @author = author
        @upvotes = 1
    end
end
```

# MIXINS

## TEDDIT AS AN EXAMPLE

```
>> story = Story.new
>> story.upvote!

>> photo = Photo.new
>> photo.downvote!
```

# INHERITANCE vs MIXINS

## WHAT'S THE DIFFERENCE?

» inheritance (`class SomeClass < OtherClass`) is used to **inherit** the methods from one class into another class

» include (`include SomeModule`) is used to **import** the methods from one module into a class

## Secret Number & Midterm

# RUBY

## SUCCESS!

Congrats! You're ready to start working with Rails!