RUBY BITS

CLASSES

PRESS START

ENCAPSULATION

```
send_tweet("Practicing Ruby-Fu!", 14)
```

```
def send_tweet(status, owner_id)
  retrieve_user(owner_id)
  ...
end
```

should not be responsible for retrieving a user

RUBY BITS

ENCAPSULATION

- Passing around data as strings and numbers breaks encapsulation.
- Places using that data need to know how to handle it.
- Individual changes require updates at various places.

RUBY BITS

ENCAPSULATION

```ruby
tweet = Tweet.new
tweet.status = "Practicing Ruby-Fu!"
tweet.owner_id = current_user.id

send_tweet(tweet)
```

```ruby
class Tweet
  attr_accessor ...

  def owner
    retrieve_user(owner_id)
  end
end
```

one parameter!

```ruby
def send_tweet(message)
  message.owner
  ...
end
```

ENCAPSULATION

- May not be worth the overhead of a class if all you have is data.
- An option hash might suffice.
- When you have behavior to go with the data, it's time to introduce a class.

VISIBILITY

```ruby
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  def bump_karma
    puts "karma up for #{name}"
  end


end
```

```ruby
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

"karma up for joe"
"karma up for leo"

should not be part of the public API

RUBY BITS

VISIBILITY

```ruby
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  private

  def bump_karma
    puts "karma up for #{name}"
  end

end
```

```ruby
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

```
private method 'bump_karma' called
for #<User:0x10ad1f6b8>
```

private methods cannot be called with explicit receiver

RUBY BITS

VISIBILITY

```
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  protected

  def bump_karma
    puts "karma up for #{name}"
  end

end
```

```
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

"karma up for joe"
"karma up for leo"

hidden from outside but accessible from other instances of same class

INHERITANCE

```
class Image
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, {@size}"
  end
end

class Video
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, {@size}"
  end
end
```

duplicated functionality!

INHERITANCE

```ruby
class Attachment
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, #{@size}"
  end
end

class Image < Attachment
end

class Video < Attachment
end
```

much DRYer!

```ruby
class Video < Attachment
  attr_accessor :duration
end
```

if a method only makes sense for one subclass, put it there.

RUBY BITS

SUPER

```
class User
  def initialize(name)
    @name = name
  end
end
```

doesn't call User#initialize

```
class Follower < User
  def initialize(name, following)
    @following = following
  end
  def relationship
    "#{@name} follows #{@following}"
  end
end
```

```
follower = Follower.new("Oprah", "aplusk")
follower.relationship
```

```
" follows aplusk"
```

no @name!

CLASSES

RUBY BITS

SUPER

```
class User
  def initialize(name)
    @name = name
  end
end
```

Calls User#initialize

```
class Follower < User
  def initialize(name, following)
    @following = following
    super(name)
  end
  def relationship
    "#{@name} follows #{@following}"
  end
end
```

```
follower = Follower.new("Oprah", "aplusk")
follower.relationship
```

▶ "Oprah follows aplusk"

SUPER

```ruby
class Grandparent
  def my_method
    "Grandparent: my_method called"
  end
end
```

```ruby
class Parent < Grandparent
end
```

not here

```ruby
class Child < Parent
  def my_method
    string = super
    "#{string}\nChild: my_method called"
  end
end
```

```ruby
child = Child.new
puts child.my_method
```

```
Grandparent: my_method called
Child: my_method called
```

RUBY BITS

SUPER

```ruby
class Grandparent
  def my_method(argument)
    "Grandparent: '#{argument}'"
  end
end
```

```ruby
class Child < Parent
  def my_method(argument)
    string = super
    "#{string}\nChild: '#{argument}'"
  end
end
```

same as super(argument)

```ruby
child = Child.new
puts child.my_method('w00t!')
```

```
Grandparent: 'w00t!'
Child: 'w00t!'
```

OVERRIDING METHODS

```ruby
class Attachment
  def preview
    case @type
    when :jpg, :png, :gif
      thumbnail
    when :mp3
      player
    end
  end
end
```

typical case

the oddball

This is slow

RUBY BITS

OVERRIDING METHODS

```
class Attachment
  def preview
    thumbnail
  end
end
```

the default

```
class Audio < Attachment
  def preview
    player
  end
end
```

special handling

new subclass!

HIDE INSTANCE VARIABLES

```
class User
  def tweet_header
    [@first_name, @last_name].join(' ')
  end
  def profile
    [@first_name, @last_name].join(' ') + @description
  end
end
```

a lot of repetition when working with these...

RUBY BITS

HIDE INSTANCE VARIABLES

```ruby
class User
  def display_name
    [@first_name, @last_name].join(' ')
  end
  def tweet_header
    display_name
  end
  def profile
    display_name + @description
  end
end
```

You can use an accessor method, even within the same class

RUBY BITS

HIDE INSTANCE VARIABLES

```
class User
  def display_name
    title = case @gender
      when :female
        married? ? "Mrs." : "Miss"
      when :male
        "Mr."
    end
    [title, @first_name, @last_name].join(' ')
  end
end
```

if you need to change the logic
later, you can do it in just one place!

RUBY BITS

PRESS START