
RUBY
BITS
DYNAMIC CLASSES & METHODS
PRESS START

STRUCT

```ruby
class Tweet
  attr_accessor :user, :status

  def initialize(user, status)
    @user, @status = user, status
  end
end
```

Struct gives same functionality

```ruby
Tweet = Struct.new(:user, :status)
```

```ruby
tweet = Tweet.new('Gregg', 'compiling!')
tweet.user # => Gregg
tweet.status # => compiling!
```

STRUCT EXTRA METHODS

```ruby
class Tweet
  attr_accessor :user, :status

  def initialize(user, status)
    @user, @status = user, status
  end

  def to_s
    "#{user}: #{status}"
  end
end
```

```ruby
Tweet = Struct.new(:user, :status) do
  def to_s
    "#{user}: #{status}"
  end
end
```

ALIAS_METHOD

```ruby
class Timeline
  def initialize(tweets = [])
    @tweets = tweets
  end

  def tweets
    @tweets
  end

  def contents
    @tweets
  end
end
```

Same implementation but different names

RUBY BITS

ALIAS_METHOD

```ruby
class Timeline
  def initialize(tweets = [])
    @tweets = tweets
  end

  def tweets
    @tweets
  end

  alias_method :contents, :tweets
end
```

Makes 'contents' a new copy of the tweets method

RUBY BITS

ALIAS_METHOD

```ruby
class Timeline
  def initialize(tweets = [])
    @tweets = tweets
  end

  attr_reader :tweets

  alias_method :contents, :tweets
end
```

RUBY BITS

ALIAS_METHOD

```
class Timeline
  attr_accessor :tweets

  def print
    puts tweets.join("\n")
  end
end
```

previous version of the method can still be used

To add authentication outside the class

```
class Timeline
  alias_method :old_print, :print

  def print
    authenticate!
    old_print
  end

  def authenticate!
    # do some authentication here
  end
end
```

SUPER IS CLEANER

```ruby
class Timeline
  attr_accessor :tweets

  def print
    puts tweets.join("\n")
  end
end
```

```ruby
class AuthenticatedTimeline < Timeline
  def print
    authenticate!
    super
  end

  def authenticate!
    # do some authentication here
  end
end
```

DEFINE_METHOD

```ruby
class Tweet
  def draft
    @status = :draft
  end

  def posted
    @status = :posted
  end

  def deleted
    @status = :deleted
  end
end
```

```ruby
class Tweet
  states = [:draft, :posted, :deleted]
  states.each do |status|
    define_method status do
      @status = status
    end
  end
end
```

methods are created dynamically

SEND

```ruby
class Timeline
  def initialize(tweets)
    @tweets = tweets
  end

  def contents
    @tweets
  end

  private

  def direct_messages
  end
end
```

```ruby
tweets = ['Compiling!', 'Bundling...']
timeline = Timeline.new(tweets)
```

```ruby
timeline.contents
```

same

```ruby
timeline.send(:contents)
```

same

```ruby
timeline.send("contents")
```

```ruby
timeline.send(:direct_messages)
```

send can run private or protected methods

SEND

```ruby
class Timeline
  def initialize(tweets)
    @tweets = tweets
  end

  def contents
    @tweets
  end

  private

  def direct_messages
  end
end
```

```ruby
tweets = ['Compiling!', 'Bundling...']
timeline = Timeline.new(tweets)
```

```ruby
timeline.send(:direct_messages)
```

can run private or protected methods

```ruby
timeline.public_send("direct_messages")
```

```
private method `direct_messages' called for
#<Timeline:0x007fd273904eb0> (NoMethodError)
```

prevents running private or protected methods

RUBY BITS

THE METHOD METHOD

```ruby
class Timeline
  def initialize(tweets)
    @tweets = tweets
  end

  def contents
    @tweets
  end

  def show_tweet(index)
    puts @tweets[index]
  end
end
```

```ruby
tweets = ['Compiling!', 'Bundling...']
timeline = Timeline.new(tweets)
```

```ruby
content_method = timeline.method(:contents)
```

```
=> #<Method: Timeline#contents>
```

```ruby
content_method.call
```

```
=> ["Compiling!", "Bundling..."]
```

```ruby
show_method = timeline.method(:show_tweet)
```

```
=> #<Method: Timeline#show_tweet>
```

```ruby
show_method.call(0)
```

```
Compiling!
```

RUBY BITS

THE METHOD METHOD

```ruby
class Timeline
  def initialize(tweets)
    @tweets = tweets
  end

  def contents
    @tweets
  end

  def show_tweet(index)
    puts @tweets[index]
  end
end
```

```ruby
tweets = ['Compiling!', 'Bundling...']
timeline = Timeline.new(tweets)
```

```ruby
show_method = timeline.method(:show_tweet)
```

```
=> #<Method: Timeline#show_tweet>
```

```ruby
(0..1).each(&show_method)
```

same

```
Compiling!
Bundling...
```

turns the Method object into a Proc object

```ruby
show_method.call(0)
show_method.call(1)
```