Dynamic Aspects of Class and Object Definitions

Alex Korban
AUTHOR, DEVELOPER
@alexkorban korban.net

Overview

Executable class bodies

Meaning of “self”

Differences between classes and objects

Define simple classes with Struct

Monkey patching and refinements

Method calls as messages

Method aliasing and singleton methods

Executable Class Bodies

Expressions

Class definitions are expressions

Executable content

The contents of a definition are executed

Scope and context

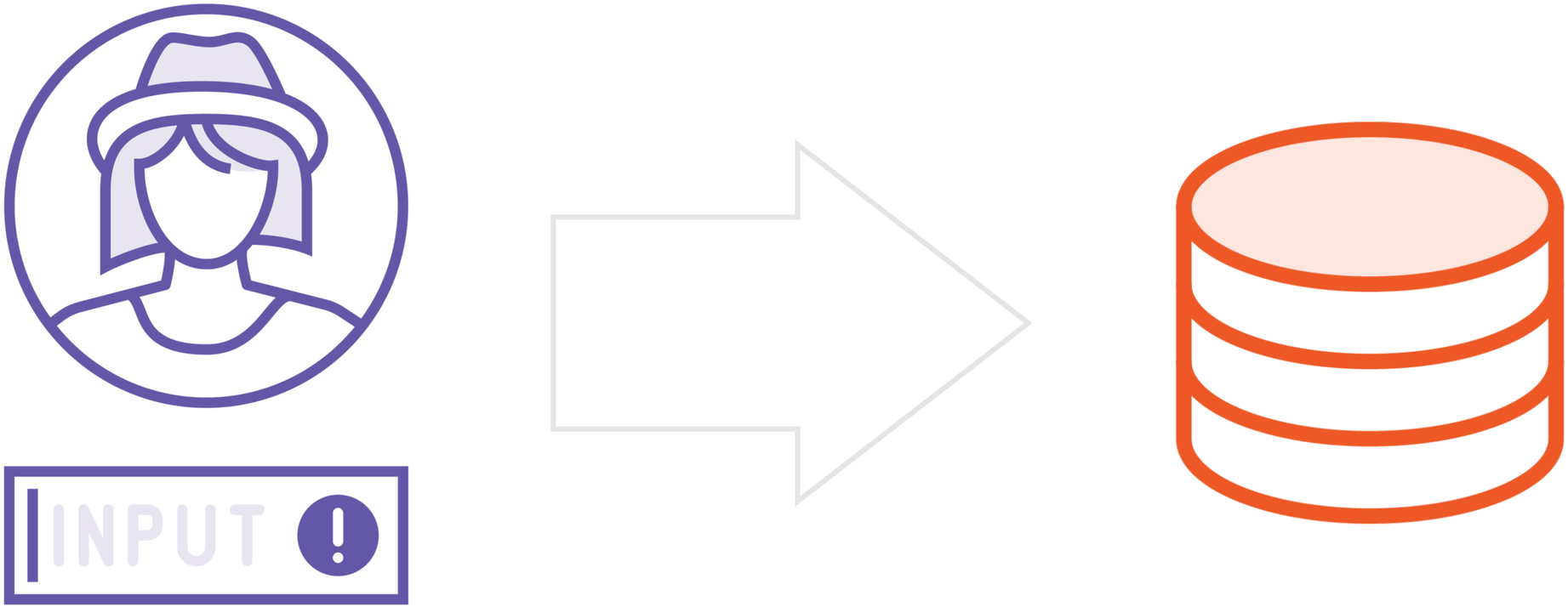
A definition is a new scope and a new context

```
attr_accessor :title, :author
```

attr_accessor

A method rather than a keyword

Validating User Input



Declarative Validation Rules

```
class BookForm
  attr_accessor :title, :author, :pub_year, :isbn

  validates :title, blank: false
  validates :author, blank: false
  validates :pub_year, type: :int
  validates :isbn, blank: false, format: :isbn
end
```

Declarative Validation Rules

```
class BookForm
  def self.validates(attr, rules)
    @validations ||= Hash.new
    @validations[attr] = rules
  end

  def is_valid?
    # apply validation rules to the attributes
  end

  def self.validations
    @validations
  end
end
```

self

Refers to the current execution context

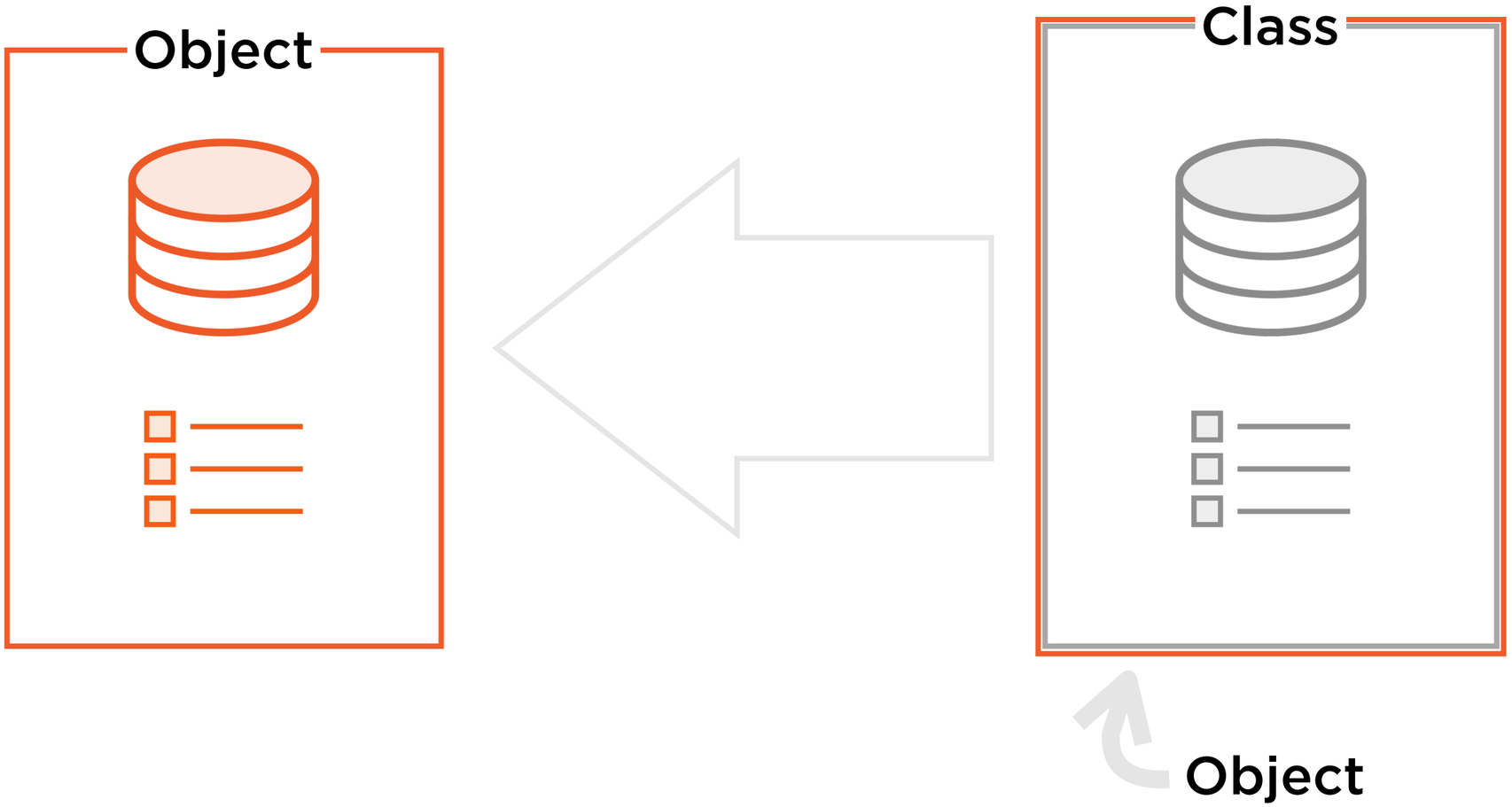
Can be used at any point in a program

Inside a class definition, refers to the class being defined

```ruby
class Book
  attr_accessor :title

  def save
    DB.save(:book, title: title)
  end
end

#...

book.save
```

- *self* refers to the class being defined inside a class definition
- *self* points to the receiver inside the method
- *self* refers to *book* inside *save*

The Difference between Objects and Classes



The Difference between Objects and Classes

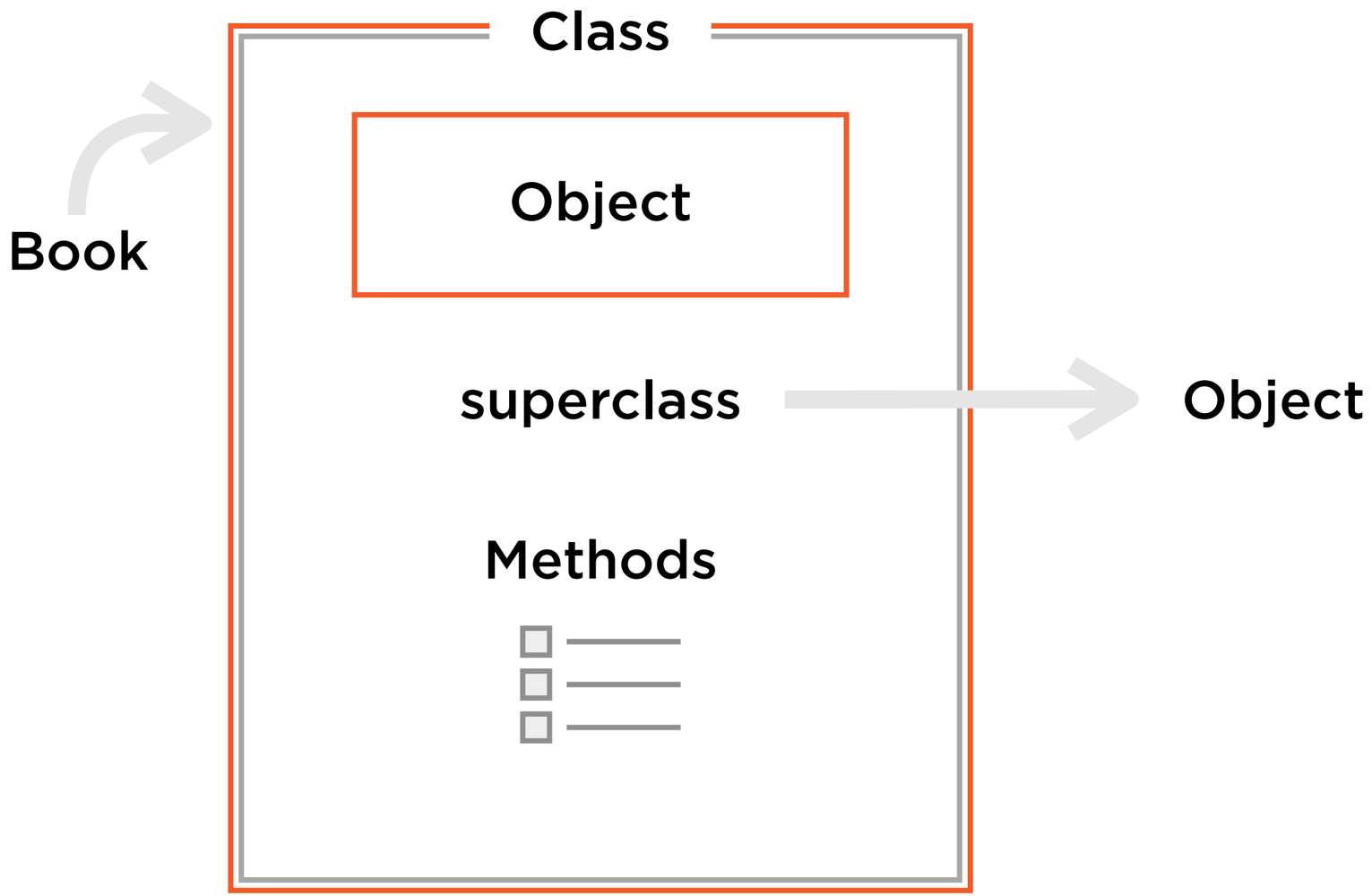
Classes as objects

Classes can be manipulated like objects

Metaprogramming

This facilitates metaprogramming, which is used quite widely

The Difference between Objects and Classes



Struct

Review

Comment

Struct

Comment

review_id

user_id

created_at

text

Class

A class forms an execution context and can contain regular executable code

```ruby
class Book
  def title=(s)
    @title = s
  end
end
class Book
  def title
    @title
  end
end
```

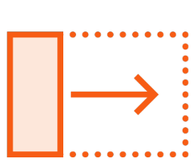
book.title

book.title

Open Classes

Add or override functionality by re-opening the class definition

Monkey Patching



Adding or modifying functionality at runtime

Overwrite an existing method

```
"code for newbies".titleize  #=> "Code for Newbies"
```

Title Case Conversion

Used in several different areas, can’t be in a class like Book or Review

Title Case Conversion

```
class String
  SHORT_WORDS =
    %w{a an and as at but by en for if in of on or the to via vs vs.}

  def titleize
    split.map {|word|
      if SHORT_WORDS.include?(word) then word else word.capitalize end
    }.join(" ")
  end
end
```

Monkey Patching

Useful tool for fixing bugs in third-party code

Use only after careful consideration

Can introduce unexpected behaviour

Modifying the standard library should be rare

Makes code more brittle across upgrades

Refinements

Monkey patching

Unintended consequences because of global effects

Refinements

Alternative to monkey patching since Ruby 2.0

Initial limitations

Limited use until recently

Ruby 2.7

A mature alternative to monkey patching

Limited scope

More restrained and safe

Refinements

```
# refine_string.rb
module RefineString
  refine String do
    SHORT_WORDS =
      %w{a an and as at but by en for if in of on or the to via vs vs.}

    def titleize
      split.map {|word|
        if SHORT_WORDS.include?(word)
          word
        else
          word.capitalize
        end
      }.join(" ")
    end
  end
end
```

```
require_relative "refine_string"

using RefineString

"code for newbies".titleize
```

Activating Refinements

Activation of refinements is governed by lexical scope

```ruby
require_relative "refine_string"

class Book
  using RefineString

  def title=(s)
    @title = s.titleize
  end
end
```

Activating Refinements

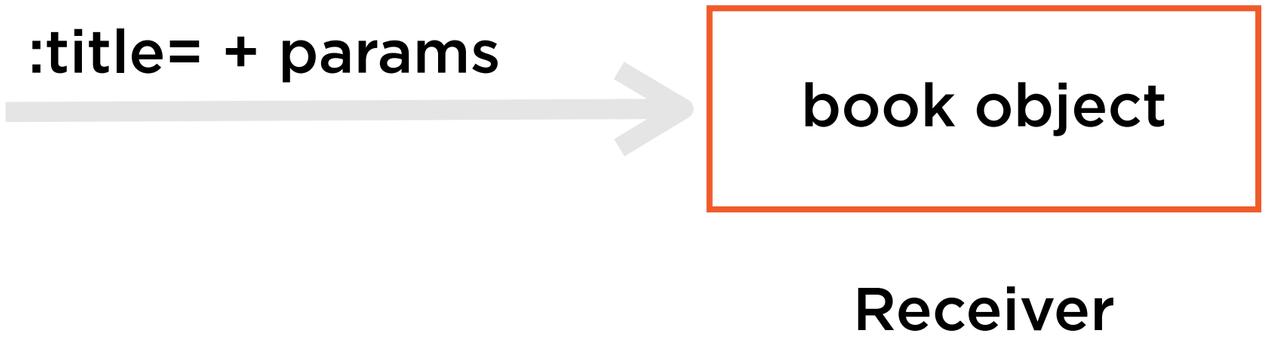
A refinement can be scoped to a class expression

```ruby
"abc".respond_to? :titleize  #=> false
```

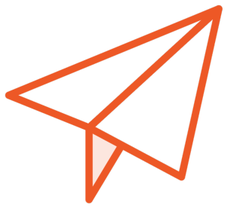
Activating Refinements

***respond_to?* doesn't account for refinements**

Methods Are Messages



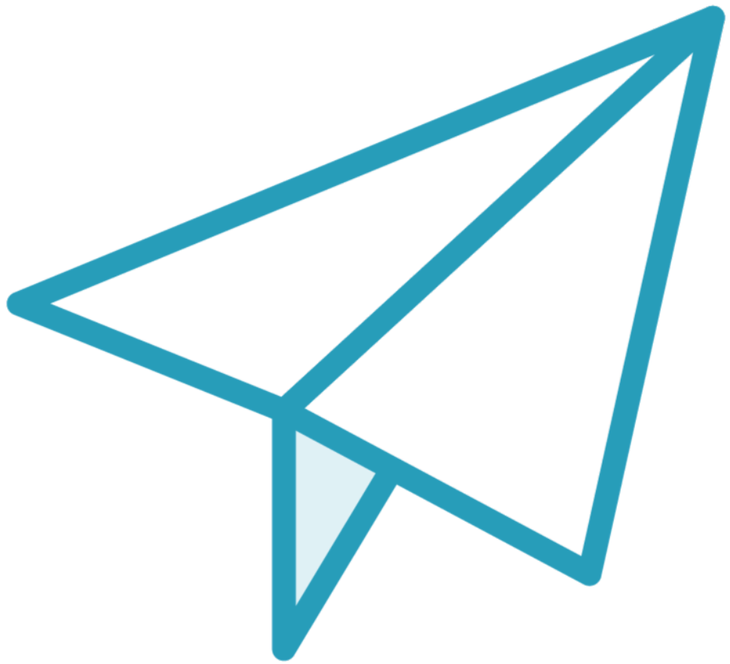
Methods Are Messages



Object **has** ***__send__*** **for sending method call messages**

```
book.public_send :title=, "Code"
```

Methods Are Messages



Methods can have restricted visibility

__*send*__ can be useful for calling non-public methods

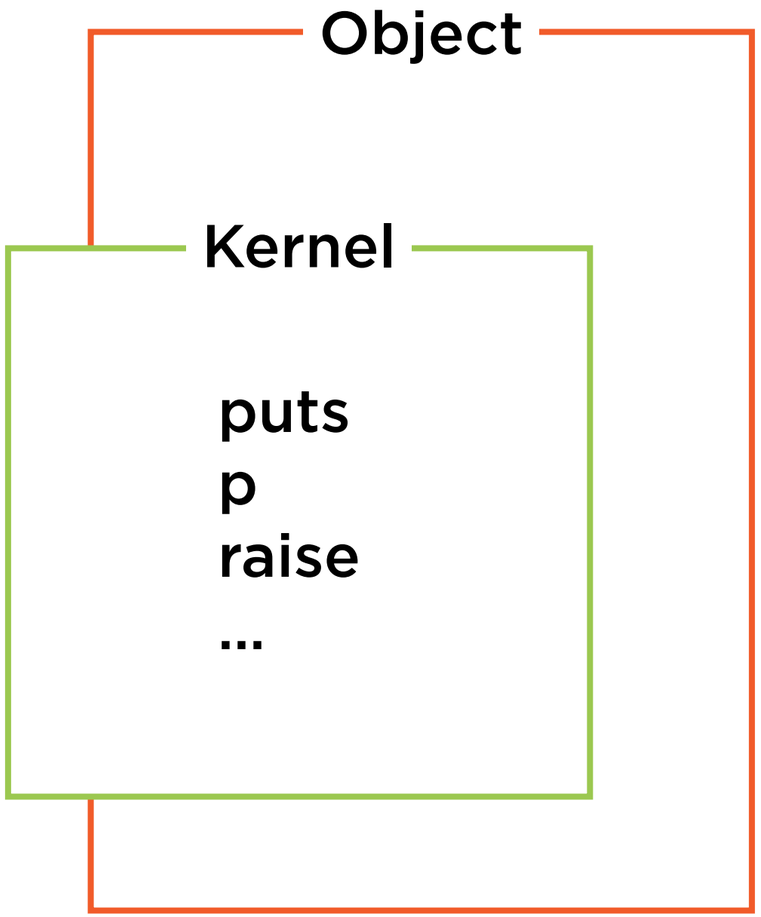
__*send*__ has an alias, *send*

```
def titleize
  split.map {|word|
    if SHORT_WORDS.include?(word) then word else word.capitalize end
  }.join(" ")
end
```

Method Calls without an Explicit Receiver

***self* is the implicit receiver**

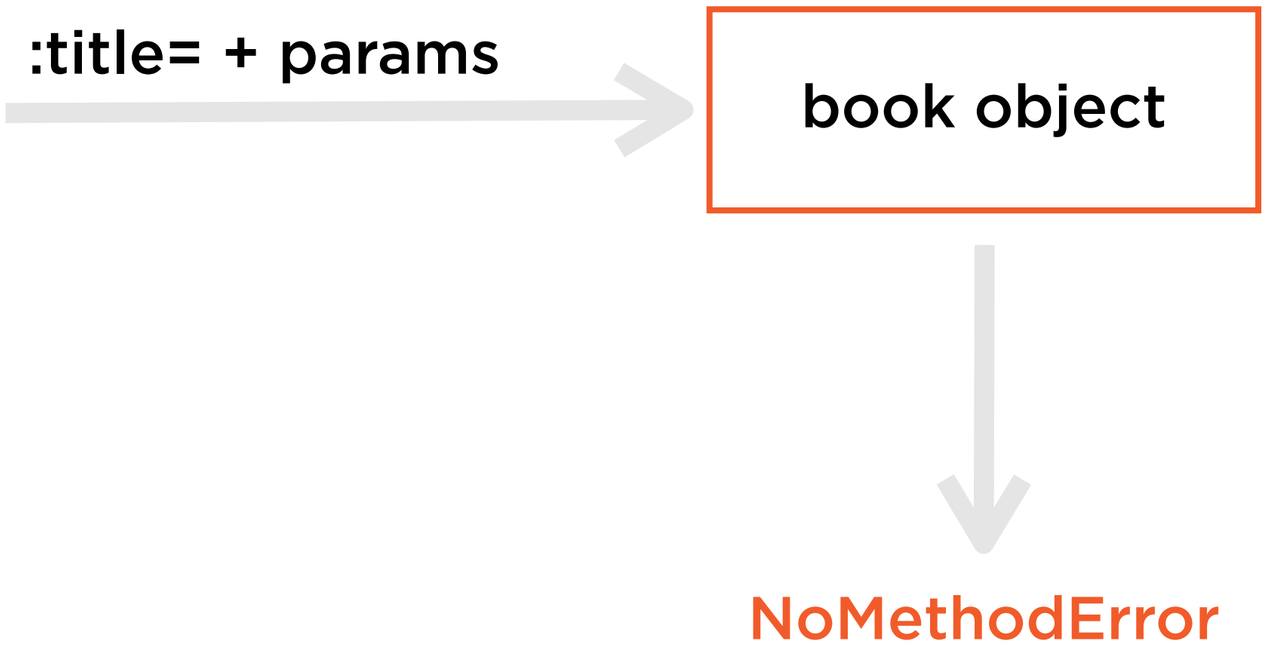
Method Receiver



method_missing

```ruby
class Query
  # Replace the query's dataset with dataset returned by the method call
  def method_missing(method, *args, &block)
    @dataset = @dataset.__send__(method, *args, &block)
    if !@dataset.is_a?(Dataset)
      raise(SQL::Error, "#{method.inspect} did not return a Dataset")
    end
    self
  end
end
```

method_missing



method_missing

method_missing

```ruby
class Query
  # Replace the query's dataset with dataset returned by the method call
  def method_missing(method, *args, &block)
    @dataset = @dataset.__send__(method, *args, &block)
    if !@dataset.is_a?(Dataset)
      raise(SQL::Error, "#{method.inspect} did not return a Dataset")
    end
    self
  end
end
```

Time the execution of methods

Log times without altering methods

```
alias_method :orig_exit, :exit
```

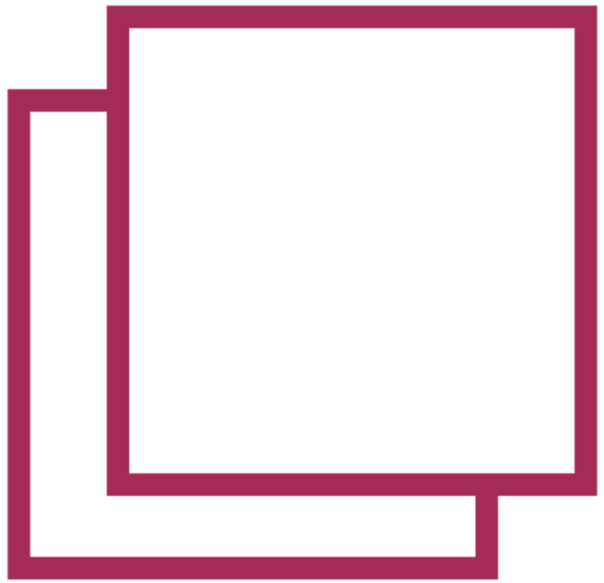
Method Aliasing

Method Aliasing

```
class Collection
  def find_by_author(author)
    puts "in find_by_author"
  end

  # pass a block for custom sorting
  def custom_sort
    puts "in custom_sort"
    yield
  end

  log_time :find_by_author
  log_time :custom_sort
end
```

Method Aliasing

```
def self.log_time(method)
  alias_method "_original_#{method}".to_sym, method

  define_method(method) {|*args, &block|
    start_time = Time.now

    puts "Calling #{method} with args #{args.inspect} #{'and a block' if block}"
    result = __send__ “_original_#{method}".to_sym, *args, &block

    end_time = Time.now - start_time
    puts "Call to #{method} with args #{args.inspect} took #{end_time}s"
    result
  }
end
```



Wrapping Functions

Can also be achieved with *Module.prepend*

Account Management

Avoid calling account management methods accidentally

Confirmation steps at the level of UI

Tests at the level of code

What if methods weren’t present until needed?

Singleton Method

A method that is defined on a specific object rather than a class

Singleton Methods

```ruby
class User
  attr_reader :id
  attr_reader :name
  attr_reader :billing_details

  def initialize(id:, name:)
    @id = id
    @name = name
  end
end

current_user = User.new(id: 123)
```

```ruby
def current_user.update_billing(details)
  @billing_details = details
end

current_user.update_billing(card: card_no)
```

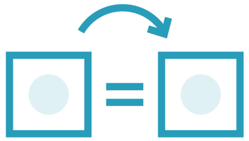
Singleton Methods

```
module AccountMgmt
  def cancel_account!
    puts "Account cancelled for #{name}"
  end

  def update_billing(details)
    @billing_details = details
  end
end
```

Class Methods Are Singleton Methods

```
def current_user.update_billing(details)
  ...

def self.find(title)
  ...
```

self refers to the class being defined



Classes are objects

A class method is a singleton method in the class object

```
def Book.find
  puts "Finding a book"
end
```

Add a Class Method Outside a Class Expression

Summary

Dynamic features of classes and objects

Classes are objects

Class definitions are expressions

Methods are messages

Alias methods

Struct can be used to define simple classes

Open classes

Monkey patching and refinements