Object-oriented Ruby Fundamentals

CLASSES AND OBJECTS

Alex Korban
AUTHOR, DEVELOPER

@alexkorban korban.net

Overview

Create classes and instantiate objects

Store data in objects and define methods

Initialize objects

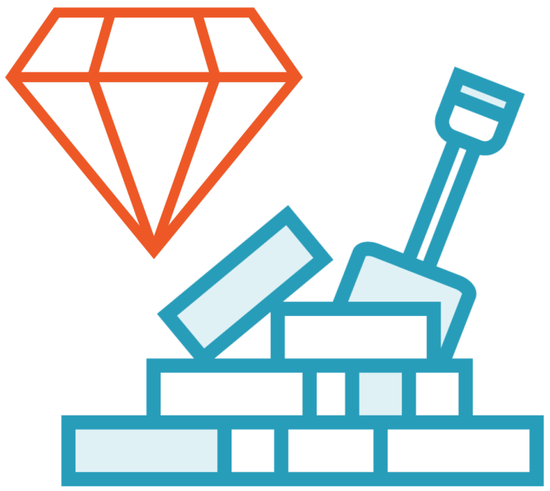
Scope data and methods to classes

Write custom operators

Order objects and test for equality

Web apps for libraries and publishers

Libra



Proof of concept

Ruby experiment

Book tracker

Reading stats

Book reviews

Follow others

Ruby

Books

Authors

Collections

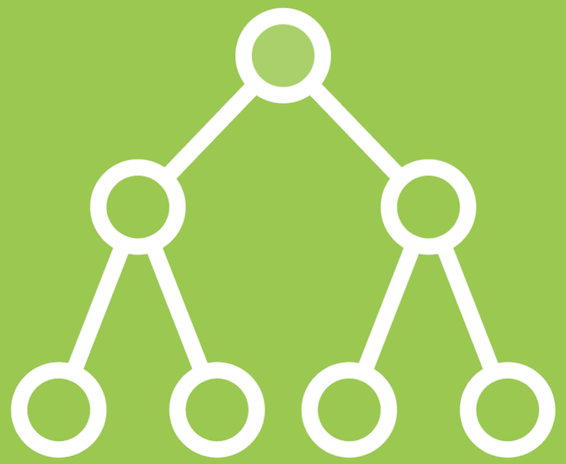
Prerequisites

Some familiarity with Ruby

Recommended courses:

- Ruby: The Big Picture
- Ruby: Getting Started

Ruby 2.7 installed



Everything Is an Object

There are relatively few primitives, and many things in Ruby are expressed in terms of objects and methods.

Objects

An object is a collection of data and methods

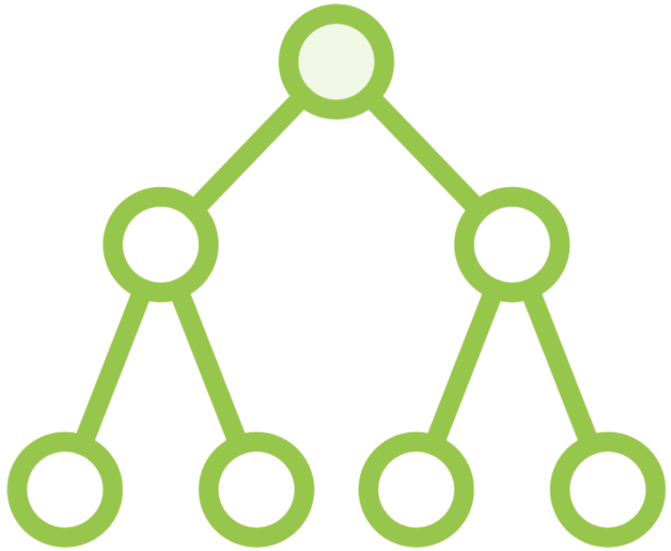
Methods return the object's data
or manipulate it

A String object has methods to get and set
its state

Current data values constitute object state

Objects

```
object.method(arguments)
```

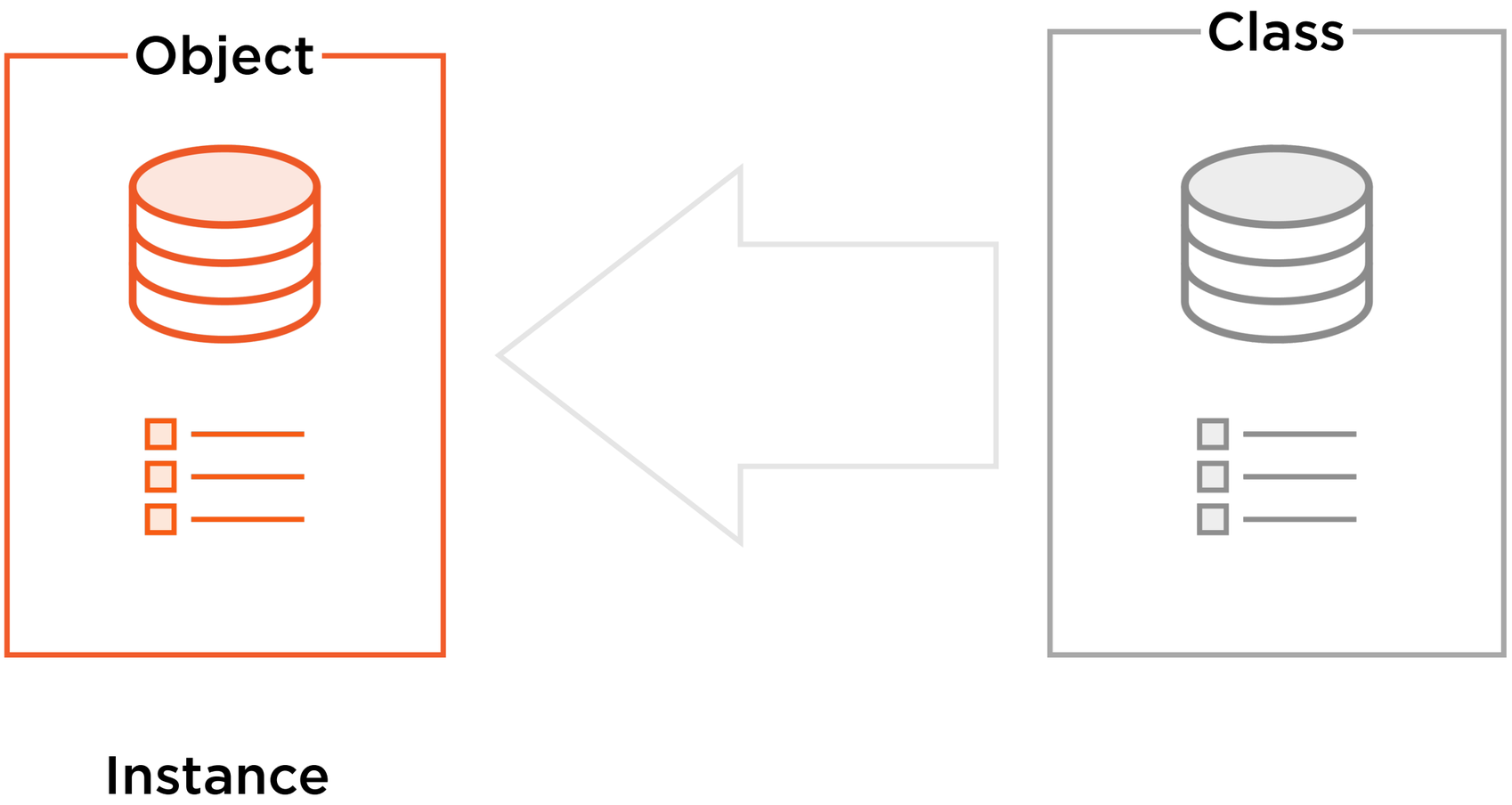


Even classes are objects

Dynamic typing + duck typing

- Variable can refer to objects of different types
- Type doesn’t determine semantics
- Polymorphism without inheritance
- More generic code

Creating and Instantiating Classes



Key Classes in Libra Project

Book

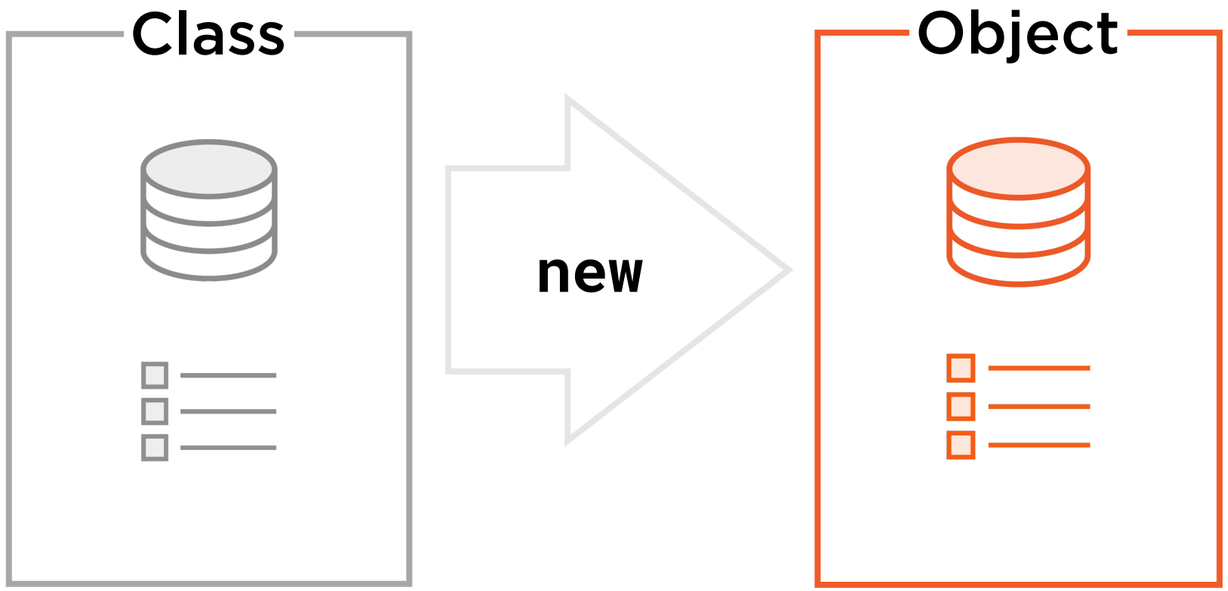
User

Collection

new

Constructor

Class method



Instance Methods and Variables

Title

Author

Instance Methods and Variables

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

  def title
    @title
  end
end
```

```ruby
b = Book.new
b.title = "Code"
b.author = "Ruby Red"
b.pub_year = 2020
```

Initialization

initialize

```
b = Book.new("Code", "Ruby Red", 2020)

b = Book.new(pub_year: 2020, title: "Code", author: "Ruby Red")
```

Initialization

Destroying Objects

Garbage collector will do it for you

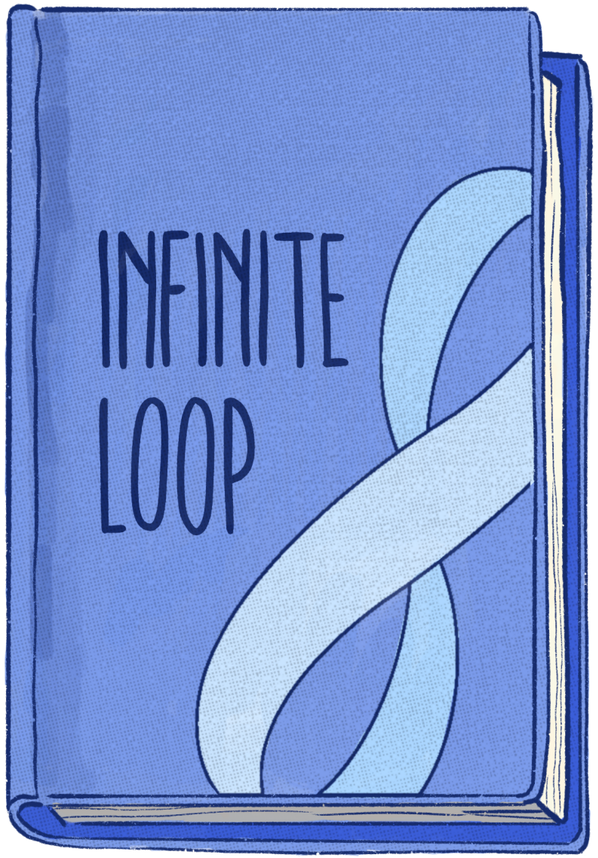
Mark and sweep

Incremental since Ruby 2.2

Manual heap compaction since Ruby 2.7

- Call GC.compact

Accessors



Title

Author

Publisher

Publication date

Page count

ISBN

Rating

Reviews

Format

Edition

```ruby
class Book
  attr_accessor :title
  attr_accessor :author
  attr_accessor :pub_year
end
```

attr_accessor

Provides read and write accessors for an instance variable

Accessors

```ruby
class Book
  attr_accessor :title
end
```

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

  def title
    @title
  end
end
```

Accessors

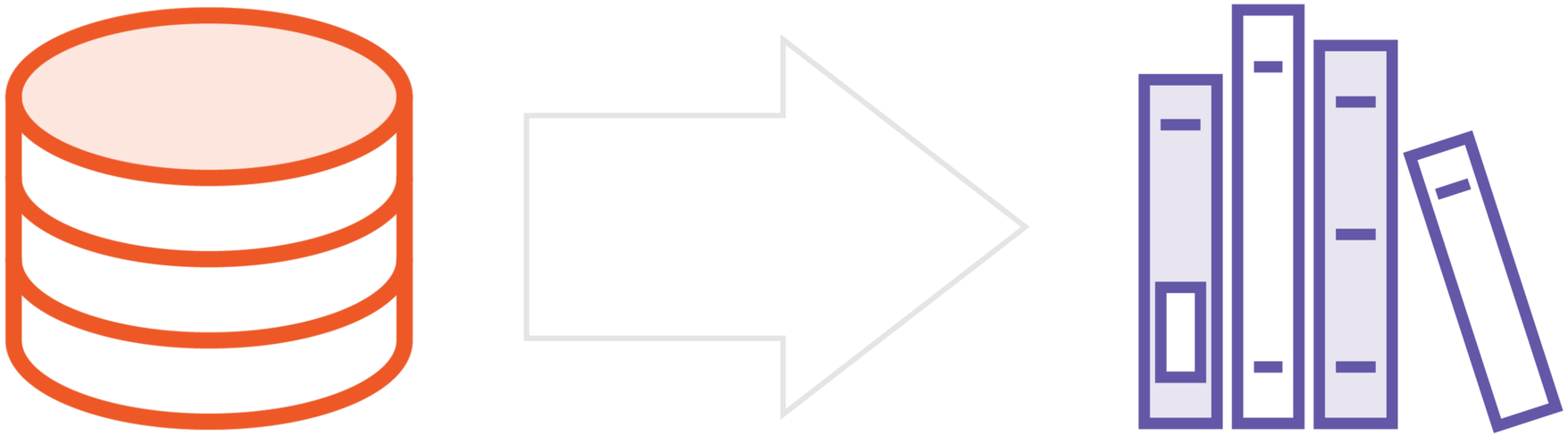
Read-only access

attr_reader

Write-only access

attr_writer

Read-only Attributes



Read-only Attributes

```ruby
class Book
  def initialize(title:, author:, pub_year:)
    @title = title
    @author = author
    @pub_year = pub_year
  end

  attr_reader :title, :author, :pub_year
end
```

attr_writer

Create only the write accessor

Attribute value goes into a calculation

Result accessible via another method

Accessors

```ruby
class Book
  attr_reader :title
  attr_writer :title
end
```

```ruby
class Book
  attr_accessor :title
end
```

Virtual Attribute

If an accessor method doesn't directly get or set an instance variable, it defines a virtual attribute.

Virtual Attributes

```ruby
class Book
  def author
    "#{@author.last_name}, #{@author.first_name}"
  end

  def author=(last_first_name)
    last_name, first_name = last_first_name.split(/\s*,\s*/)
    @author = Author.find(first_name, last_name)
  end
end
```

Virtual Attributes

Uniformity

Regular and virtual attributes have the same interface

Consistency

Replace direct instance variable access with more complex code

Class Methods

Search for a book

Class Methods

Class method

Called on a class instead of an object

new

new is an example of a class method

Object state

Independent of object state

Class Methods

```
Book.find
```

```
book.find
```

```
self.
```

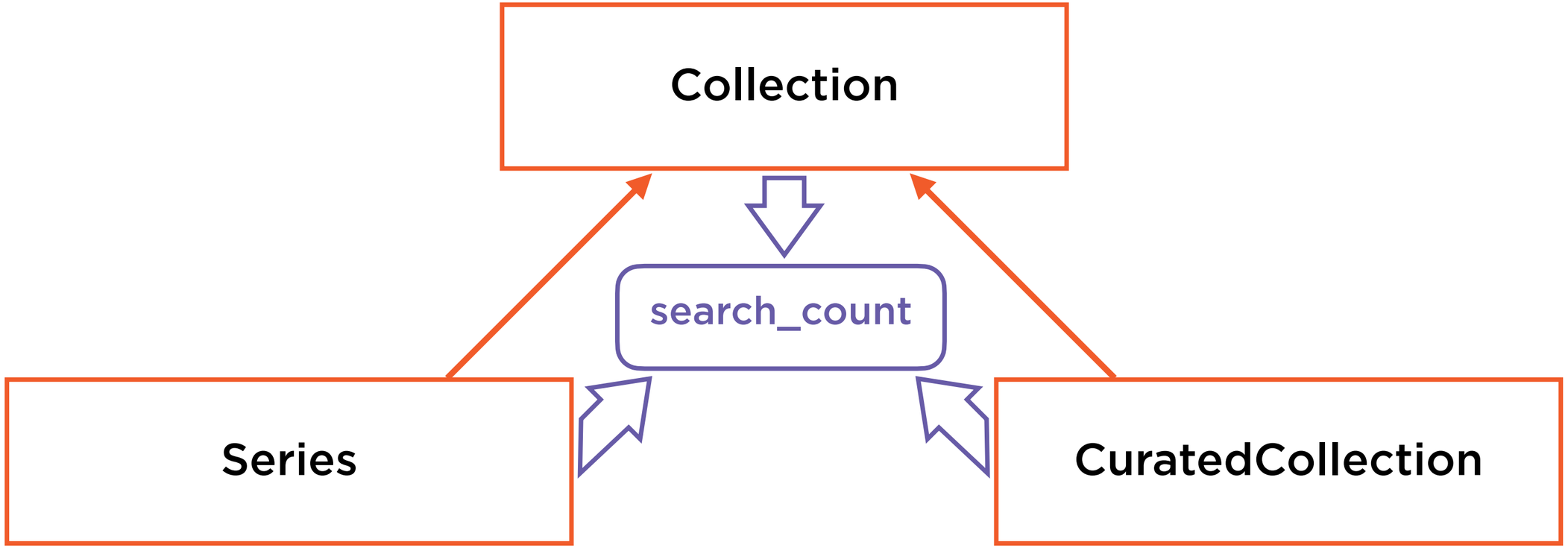
Class Variables

Class variables maintain class-level state which can be referred to by class methods.

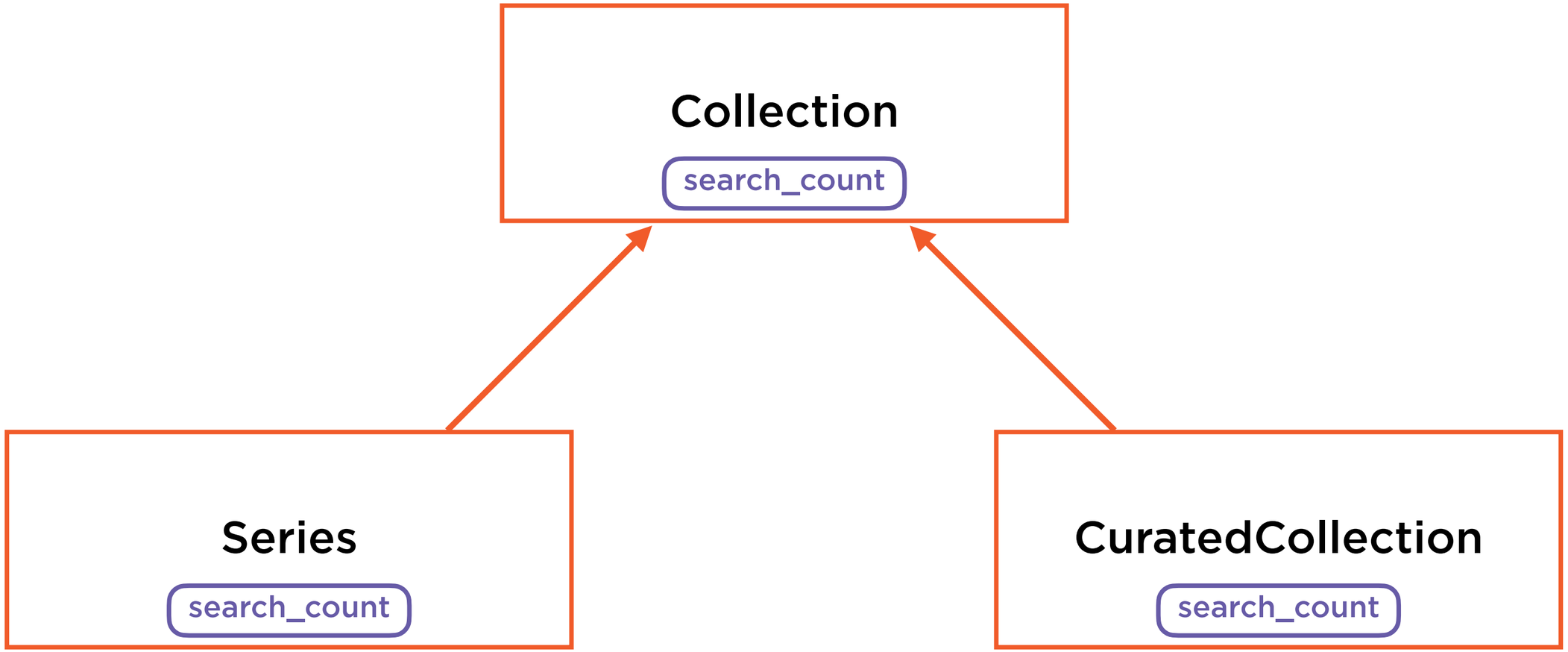
Class Variables

```ruby
class Book
  @@search_count = 0

  def self.find(title)
    @@search_count += 1
    Book.new(title: "Code", author: "Ruby Red", pub_year: 2020)
  end

  def self.search_count
    @@search_count
  end
end
```

Class Variables



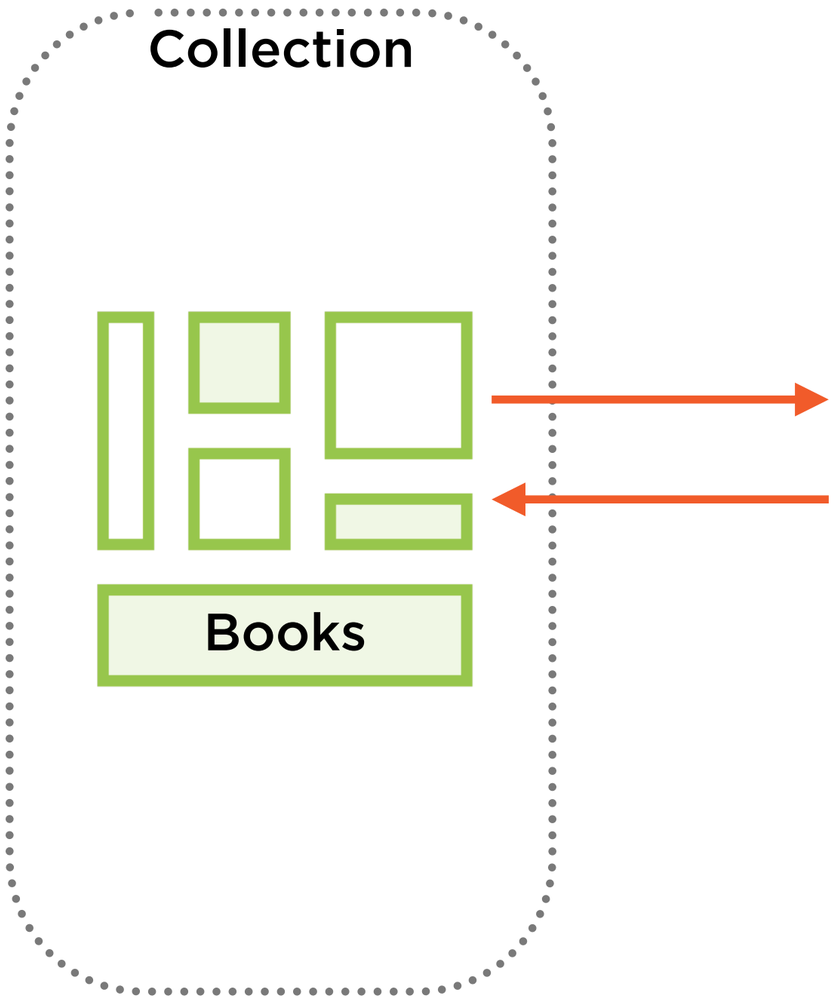
Class Instance Variables



Operators

Not implemented as methods	Implemented as methods
&&	[]
||	Comparison operators
and	Mathematical operators
or	Bitwise operators
not	
? :	
Assignment operators	

Collection



Custom Operators

Mimic widespread semantics

Match up with standard library behaviour

Emulate outside semantics such as mathematical operators

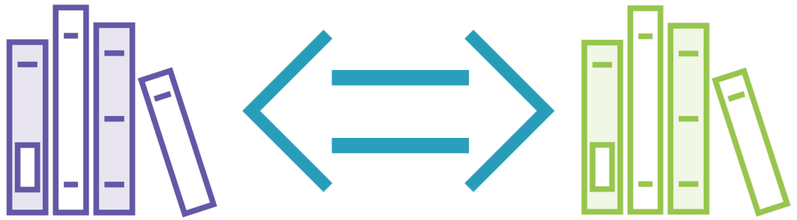
Object Comparison and Ordering

A

...

Z

Enumerable



<=>

Default implementation	obj1 == obj2 → 0 otherwise → nil
Needed implementation	obj1 == obj2 → 0 obj1 < obj2 → -1 obj1 > obj2 → 1

Object Equality

Object Equality

Equality comparison methods

```
==
eql?
equal?
```

Equivalent expressions

```
obj1 == obj2
obj1.eql? obj2
obj1.equal? obj2
```

equal?

The purpose of this method is specifically to check for object identity. You should not override it in your classes and subclasses.

Class Equality

```
def collection_html(collection)
  case collection
  when Series
    # return series-specific HTML
  when Collection
    # return generic collection HTML
  end
end
```

Class Equality

kind_of? / is_a?

`series.kind_of?(Series)` → **true**

`series.kind_of?(Collection)` → **true**

instance_of?

`series.instance_of?(Series)` → **true**

`series.instance_of?(Collection)` → **false**

Summary

Modelling with classes

Creating classes and instantiating objects

Adding instance variables and methods

Defining accessors and virtual attributes

Initializing objects

Creating class variables and methods

Writing operators

Understanding object and class equality