Classes in Ruby

Plus, a brief introduction to
Object Oriented Programming

Our Goals

- Define what a class is
- Talk about why you would actually use them
- Explain the difference between instance variables and normal variables
- Be able to create new instances, and call methods upon them
- Explain what Object Oriented Programming is very briefly?

So, Object Oriented Programming?

- It's a very, very old principle
- It's a focus on data and structure rather than logic
- It is an attempt to replicate real life
- Ruby is definitely an object-oriented language!

Everything is an object!

```ruby
# Everything in Ruby inherits from a class / hash

{}.class # => Hash
[].class # => Array
"".class # => String

{}.class.ancestors
[].class.ancestors
"".class.ancestors

{}.class.superclass
```

A lot of talk about classes...

What are they?

- Very similar to factories, constructors and OLOO in JavaScript
- I like to think of them as blueprints
- They help to reduce duplicated code
- They make it easier to debug
- They help with organisation of your code

What do they look like?

```
class Person
end

class Animal
end

class Vehicle
end

class Instrument
end
```

We can add methods

```ruby
class Person
    def speak
        puts "I am now speaking"
    end

    def laugh
        puts "out loud"
    end
end
```

We always create instances

```
class Person
    def speak
    end

    def laugh
    end
end

# Create an instance
person = Person.new

# Call methods on the instance
person.speak
person.laugh
```

Types of Variables

Local Variables

- Defined in a method, and not available outside of that method
- Always start with lowercase letter or an underscore

Instance Variables

- Available everywhere on an instance of a class or an object (all methods)
- Prefixed with an @

Types of Variables

Class Variables

- Available throughout all instances of a class
- They belong to a particular class, and are often characteristics
- Prefixed with @@

Global Variables

- Available between all classes
- Prefixed with a $

Types of Variables

Constants

- Can never be changed
- All uppercase, the words are seperated by underscores if necessary

Types of Methods

Instance Methods

Methods on an instance of a class

Factory Methods

Methods on a class itself

Predicate Methods

Methods that return true or false (2.even? for example)

Storing Information

```ruby
# We want to store information on a person!
# We use getters and setters to do this

class Person
    def name=( name )
        @name = name
    end

    def name
        @name
    end
end

jane = Person.new
jane.name=( "Jane" )
jane.name # => "Jane"
```

There is a bit of duplication now...

```ruby
class Person
    def name=( name )
        @name = name
    end

    def age=( age )
        @age = age
    end
end

jane = Person.new
jane.name = "Jane"
jane.age = 42
```

Let's make that better!

```ruby
class Person
    attr_accessor :name, :age
end

jane = Person.new
jane.name = "Jane"
jane.age = 42
```

Attr

```
class Person
    attr_accessor :name, :age
end

class Person
    attr_reader :name, :age
end

class Person
    attr_writer :name, :age
end
```

Attr

```ruby
class Person
    attr_accessor :name, :age
end

person = Person.new

person.name = "Name"
person.age = 42

# We could have to write this stuff a lot...
```

Initialize

```
class Person
    attr_accessor :name, :age

    def initialize( name, age )
        @name = name
        @age = age
    end
end

person = Person.new "Person", 42
```

Instance Methods

```ruby
class Person
    def method_one
    end
    def method_two
    end
end

person = Person.new "Person", 42

# See all of the instance methods
person.class.instance_methods
# See all of the instance methods that you've created
person.class.instance_methods( false )
```

Inheritance

```
class Vehicle
    def generic_vehicle_method
    end
end

class Boat < Vehicle
    def specific_boat_method
    end
end

b = Boat.new
b.specific_boat_method
b.generic_vehicle_method
```

The Composed Method Technique

Divide your classes up into lots of small methods!

1. Each method should do a single thing, focusing on solving a single aspect of a problem
2. Each methods needs to operate at a single conceptual level - don't mix high level logic with nitty-gritty details
3. Each method needs to have a name that reflects its purpose

Have a crack at these exercises