Understanding Class Methods verses Instance Methods in Ruby

http://culttt.com/2015/06/10/understanding-class-methods-verses-instance-methods-in-ruby/

Over the last couple of weeks we've started to look at defining classes in Ruby.

First we looked at writing methods and some of the idioms you might encounter (Working with Ruby Methods).

Next we looked at defining basic classes and how we can use some of Ruby's nice little features to make setting up the boilerplate code much easier (Writing Ruby Classes).

Last week we looked at method accessibility and the difference between Public, Private and Protected methods (The difference between Public, Protected and Private methods in Ruby).

In this week's tutorial we're going to be looking at another important aspect of writing Ruby classes, Class methods verses Instance methods and the differences between them.

What are Class Methods?

I think in order to fully describe what is the difference between Class Methods and Instance Methods, it's probably going to be easier to just jump straight into some sample code.

In today's tutorial we're going to be building a shopping basket:

```
class Basket

end
```

So first up we'll look at Class Methods. A Class Method is a method that is defined on the class. For example:

```
class Basket
  def self.find(id)
    puts "finding basket with the id of #{id}"
  end
end
```

In this example I've defined a `find` method on the `Basket` class that accepts an `id`. In theory this would go to the database to find the particular basket and return it as a `Basket` object.

Notice how I've used `self` in the method definition? This is signifying that this method is a Class Method. Alternatively you could write it like this:

```
class Basket
  class << self
    def find(id)
      puts "finding basket with the id of #{id}"
    end
  end
end
```

To use this method, you invoke it from the class:

```
basket = Basket.find('abc')
```

When would you use a Class Method?

A Class Method is a piece of functionality that belongs to that class, but is not tied to any particular single instance.

In this case, we're simply using the `find` method to encapsulate the process of retrieving a basket object from the database.

You should use Class Methods when the functionality you are writing does not belong to an instance of that class.

What are Instance Methods

Instance Methods on the other hand, can only be called on a particular instance of the class.

```
class Basket
  def self.find(id)
    puts "finding basket with the id of #{id}"
  end

  def products
    []
  end
end
```

In this example I've added a `products` Instance Method. This Instance Method only makes sense when we actually have an instance of the `Basket` class:

```
basket = Basket.new
=> #<Basket:0x007fd2e446c7b0>

basket.products
=> []
```

If you try to call an Instance Method directly on the class, you will get an error:

```
Basket.products
NoMethodError: undefined method 'products' for Basket:Class
```

And similarly, if you try to call a Class Method on an object instance, you will also get an error:

```
basket.find('def')
NoMethodError: undefined method 'find' for #<Basket:0x007fd2e446c7b0>
```

When would you use Instance Methods?

You use Instance Methods when you need to act on a particular instance of the class.

This is often when the functionality concerns the identity of the instance such as calling properties on the object, or invoking behaviour.

For example, the `products` method only makes sense in the context of a Basket instance. You can't have a list of products without actually having a basket object to work with.

Conclusion

Over the last couple of weeks I've tried to emphasis good object-oriented design.

Class Methods verses Instance Methods seems to be one of those areas where object-design can go wrong.

You will often see Ruby classes that make use of Class Methods when they really should be using Instance Methods.

When the functionality you are implementing is tied to the identity of a particular object, you should be using instance methods.

Good object-oriented design is all about encapsulation and modelling behaviour as objects.

Class Methods are often used as a crutch to implement procedural code. This is missed because on the surface it looks like you are implementing the functionality on the given class.

However it is very important to be mindful of the functionality you are implementing and whether it should be in the context of a particular instance of the object or not.