Why you need to avoid class variables

Have you ever wanted to have some data that is stored on the class itself instead of every individual object of this class?

This could be something like a on / off flag or some kind of counter.

In Ruby, we have two ways to do this & that's what you are going to learn in this lesson.

Class Variables

A class variable looks like this: `@@count`.

But the problem is that class variables are shared with subclasses.

Example:

```ruby
class Animal
  @@count = 15

  def self.count
    @@count
  end
end

class Tiger < Animal
  @@count = 1
end

p Animal.count # 1
p Tiger.count  # 1
```

As you can see in this example, defining `@@count` in `Tiger` changes the value of `@@count` in `Animal`, because they are the same variable.

This is the main reason to avoid class variables. In the next section you will learn what you should do instead.

Class Instance Variables

Since classes are objects too, they can have their own instance variables.

Example:

```ruby
class Animal
  @count = 10
end
```

> Notice how this is a regular instance variable defined outside of any methods. That's what makes it a 'class instance variable'.

To access this variable you have to define a getter / accessor method.

Like this:

```ruby
class Animal
  @count = 10

  def self.count
    @count
  end
end

p Animal.count
```

This `@count` variable is unique for `Animal`, so this solves the problem with regular class variables.

Now if you create a subclass of `Animal` you will find that `@count` is `nil`.

```
class Cow
  @count = 10

  def self.count
    @count
  end
end

class BigCow < Cow
end

p Cow.count    # 10
p BigCow.count # nil
```

Conclusion

I hope that helps! The takeaway is that you should use class instance variable to avoid any problems.