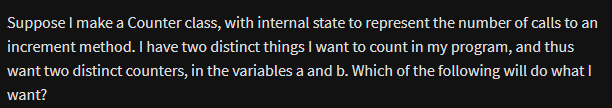
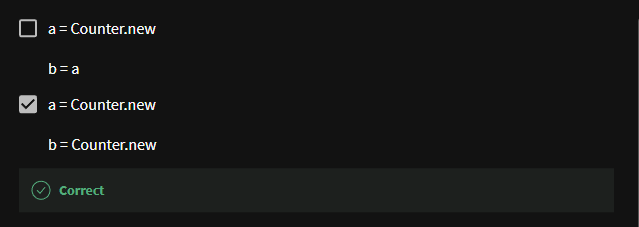
**Objects have state**

* An object’s state persists
  + Can grow and change from time object is created
* State only directly accessible from object’s methods
  + Can read, write, extend the state
  + Effects persist for next method call
* State consists of **instance variables**
  + Aka **fields**
  + Syntax: starts with an @
    - Eg. @foo
  + “Spring into being” with assignment
    - So, misspellings silently add new state (!)
    - Using one not in state not an error / accessing an undefined variable
      * Produces **nil** object

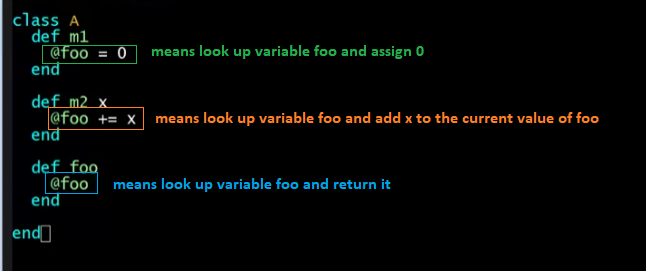
**Aliasing**

* Creating an object returns a reference to a **new object** NOT an alias
  + **Different** state from EVERY other object
* Variable assignment (e.g., x=y) creates an **alias**
  + Aliasing means **same object** means **same state**

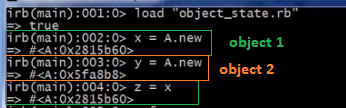




**Seeing all in action**

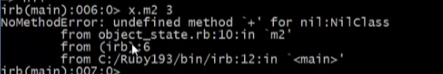


Aliasing





* foo not yet instantiated



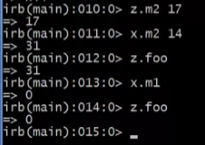
* you cannot call + on an undefined variable



* instantiate the variable foo, assign something to it



* calling z.foo will get you 0 since x.foo is already 0 (alias -> object 1)
* calling y.foo is nil because you haven’t instantiated the foo variable here in this object (object 2)



Adding more instantiation in our class

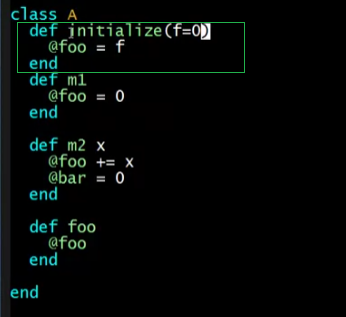
**initialize** Method

* called when you create objects
* much better style
* syntax:

def initialize (parameter = default arg)

// your variable instantiations

End



When you have this **initialize** method

* it will be called before the object returns
* it will be called when you instantiate a new object

no argument for new:



* + used the default value of foo

with argument for new:



* + assigned the argument to the value of foo

**Initialization**

* A method name **initialize** is special
  + called on a new object before new returns
  + arguments to new are passed on to initialize
  + excellent for creating object invariants
  + like constructors in Java/C#/etc.
* Usually, good style to create instance variables in initialize
  + just a convention
  + unlike OOP languages that make “what fields an object has” a (fixed) part of the class definition
    - in Ruby, different instances of same class can have different instance variables

**Class variables**

* There is also **state shared by the entire class**
* Shared by (and only accessible to) all instances of the class
* Called *class variables*
  + Syntax: starts with an @@
    - E.g., @@foo
* Less common, but sometimes useful
  + And helps explain via contrast that each object has its own instance variables

**Class constants and methods**

* *Class constants*
  + Syntax: start with capital letter
    - E.g., Foo
  + Should not be mutated
  + Visible outside class c as c::Foo (unlike class variables)
    - Accessing: <class name>::<constant name>
* *Class methods* (cf. Java/C# static methods)
  + Syntax (in some class C):

def self.method\_name (args)

…

End

* + Use (of class method in class C):
    - C.method\_name(args)
  + Part of the class, not a particular instance of it

Example:

