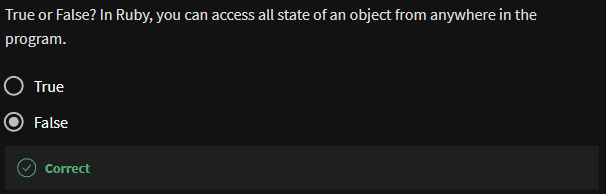
The rules of class-based OOP

In Ruby:

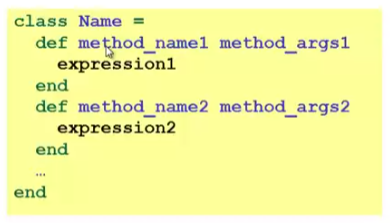
1. All values are references to *objects*
2. Objects communicate via *method* calls, also known as *messages*
3. Each object has its own (private) state
4. Every object is an instance of a class
5. An object’s class determines the object’s *behavior*
   1. How it handles method calls
   2. Class contains method definitions

Java/C#/etc. similar but do not follow (1) (e.g., numbers, null) and allow objects to have *non-private* state



* Rule no. 3 -> each object has its own private state

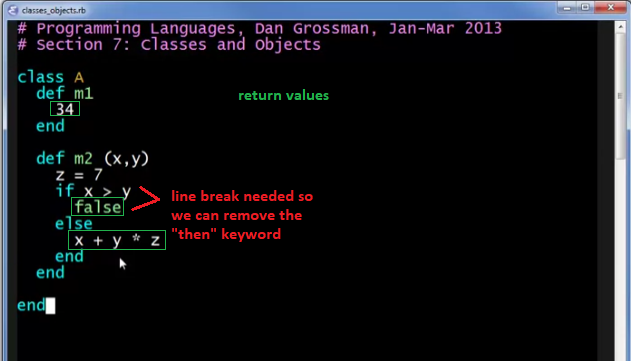
**Defining classes and methods**



* Define a new class called with methods as defined
* Method returns its last expression
  + Ruby also has explicit return statement
* Syntax note: **Line breaks often required** (else need more syntax), but indentation always only style

**Example**

Defining my class A



Loading the file



Making an object out of our class

<class name>.new



Assigning it to a variable

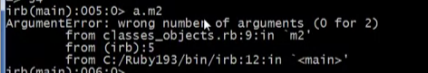
<variable name> = <class name>.new



Call the object’s method

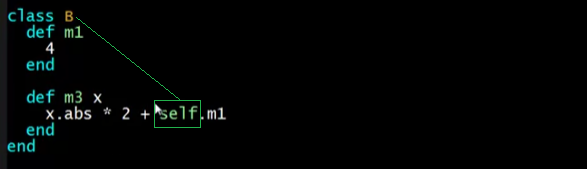
<variable name>.<method name>(…args if applicable)







Defining my new class B



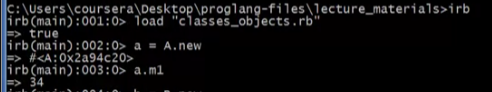
self = that means B.m1

Quitting the current session

quit



Load again



Assign to a new variable



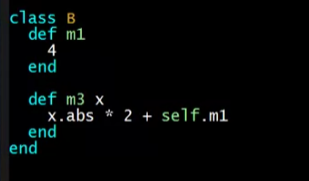
Accessing methods



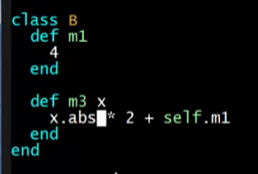


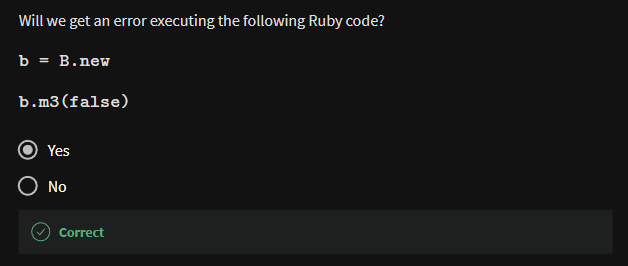


* Objects with class B do not have an M2 method





* 
* Numbers have built-in methods like “abs”

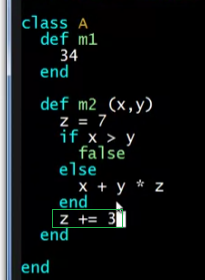


* Booleans have no “abs” method

**Creating and using an object**

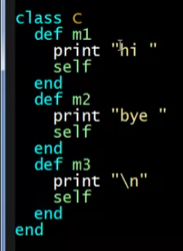
* **ClassName.new**
  + Creates a new object whole class is **ClassName**
* **e.m**
  + evaluates **e** to an object and then calls its **m** method
  + also known as “sends the m message”
  + can also write **e.m()**
* Methods can take arguments, called like **e.m(e1, …, en)**
  + Parentheses optional in some places, but recommended

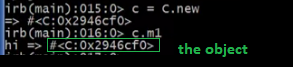
**Variables**

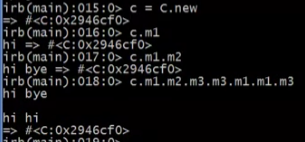
* Methods can use local variables
  + Syntax: starts with letter
  + Scope is method body
* No declaring them, just assign to them anywhere in method body (!)
* Variables are **mutable**, **x=e**
  + 
* Variables also allowed at “top-level” or in REPL
* Contents of variables are always references to objects because all values are objects

**Self**

* self is a special keyword/variable in Ruby
* Refers to “the current object”
  + The object whose method is executing
* So call another method on “same object” with **self.m(…)**
  + Syntactic sugar: can just write **m(…)**
    - no self.
* Also, can pass/return/store “the whole object” with just **self**







* + We can call the object from here
* Same as **this** in Java/C#/C++

Note:

* **New lines/separate lines** matter in Ruby
* **Indentation** does **not**