Subclasses, inheritance, and overriding

* The essence of OOP

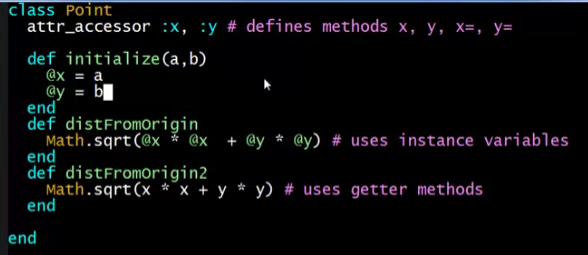
Subclassing

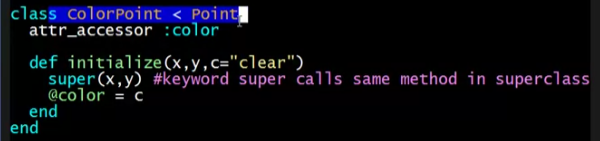
* A class definition has a superclass (Object if not specified)



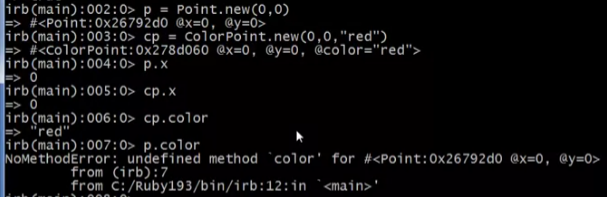
* The superclass affects the class definition
  + Subclass *inherits* all method definitions from superclass
  + But subclass can *override* method definitions as desired
* Unlike Java/C#/C++
  + No such thing as “inheriting fields” since all objects create instance variables by assigning to them
  + Subclassing has nothing to do with a (non-existent) type system: can still (try to) call any method on any object

Example





Using in REPL



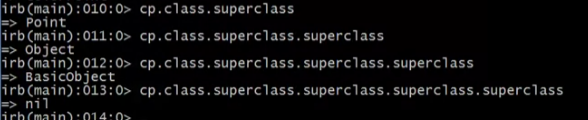
Checking the class of an object

* .class



Checking the superclass of a class

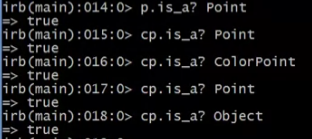
* .superclass



* + nil.superclass => error

Checking if an object is a class/subclass of the given class

* *obj*.is\_a? *class*

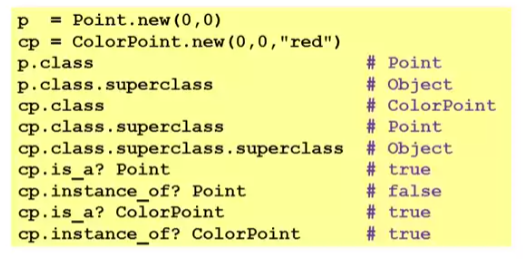


Checking if an object is an INSTANCE of the class

* should be a **direct** instantiation of the class
  + not just a subclass of the class
* *obj*.instance\_of? *class*



An object has a class



* Using these methods is usually non-OOP style
  + Disallows other things that “act like a duck”
  + Nonetheless semantics is that an instance of ColorPoint “is a” Point but is not an “instance of” Point
  + [ Java note: **instanceof** is like Ruby’s **is\_a?** not Ruby’s **instance\_of** ]