Object Oriented Programming

...

Inheritance and Composition

Overview of topics

- Quickly cover OOP principles
  - What makes a language object-oriented?
- Real world examples of inheritance
- Inheritance in Ruby
  - “Is-a” relationships
- Composition in Ruby
  - “Has-a” relationships
  - Mixins (including Modules in classes and extending object instances)
- Method invocation in Ruby
  - Top-level vs class vs instance vs singleton
  - Method invocation precedence

What Makes a Computer Language Object-Oriented

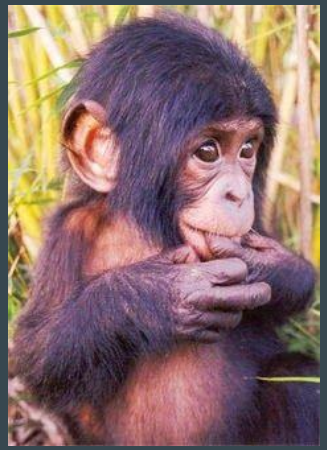
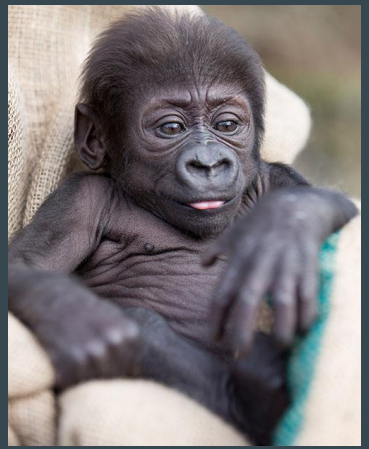
- An object-oriented language (generally) supports the following architectural characteristics
  - Classes
  - Objects (instances of classes)
  - Inheritance (classes as parents of other classes or the “is-a” relationship)
  - Composition (object instances containing other objects as data members or the “has-a” relationship)
  - Method calling (message sending) among objects
  - Encapsulation (abstraction of object data by allowing internal data access via public methods)
  - Polymorphism (the ability for different objects of different types to respond to a common interface definitions - most commonly seen in the context of method overriding in a class hierarchy chain)

What's in it for me?

- Unit testing
- Project management and division of labor
- Flexible design
- APIs

(Sort of) Real World Example

- Divide into groups!
- Use OOP skills to design a class hierarchy for the following situation:
  - The Salamanders' Wild and Crazy Great Ape Sanctuary!!
  - How would we model the sanctuary and the collection of great apes (which consists of some humans, some orangutans, some chimpanzees, and some gorillas. Just to make things exciting, there are some birds that fly around too to bother everyone). No code, just relationships.


Inheritance (“is-a”)

- Live code of ape relationships

Composition (“has-a”)

- Live code of sanctuary class

Modules (“Mixin”)

- Many languages support multiple inheritance
- Ruby does not
- However, methods that could apply to many types of class can be grouped in Modules
    - You’ve encountered modules - “Enumerable” methods are mixed into classes that support “enumerable” behavior
    - Including “Comparable” can be powerful
    - Modules are not classes
        - Just to confuse you, Ruby Classes are instances of the “Class” object, which inherits from the “Module” class (because everything in Ruby is an object)

Takeaway

- Focus on design
- What's the proper relationships between objects in the universe of your code?
- If you can't describe the general purpose of a class in one line (or an interface), it could be doing too much