What is OOP?

- Objects are complex entities (which we sometimes call "data structures") with qualities and abilities.
- In an object-oriented programming language, we work with complex objects rather than simple "primitives" like numbers and letters.

Object Orientation

- (Nearly) Everything is an Object
- Objects "communicate" by sending and receiving messages
- Objects have their own memory
- Every object is an instance of a class

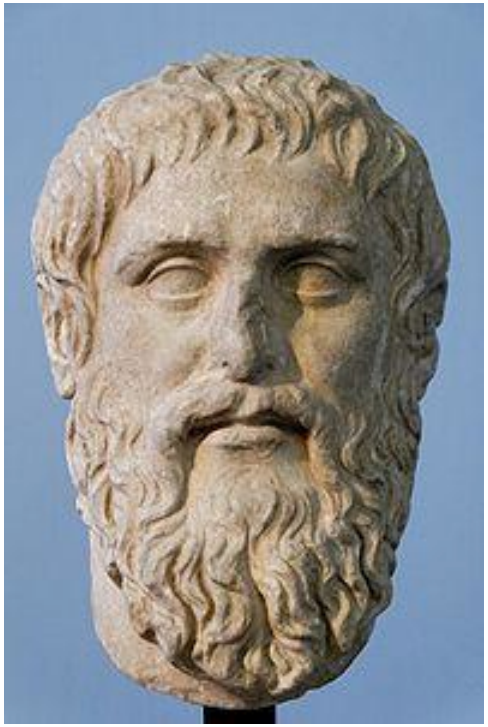
Alan Kay

Object-oriented Programming
Graphical User Interface
3D Graphics
ARPANET (what became the Internet)

Classes and instances

Classes are **archetypes**

Instances are particular **objects**



Class

Describes the generic characteristics of a single ***type*** of an object

What things are of this type ***are***

- Dog
- Vehicle
- Baby

Classes and Instances

Classes

- Template for an object
- Describes state
- Describes behavior
- Used to create many instances

Instances

- Discreet instantiation of a class
- Shares behavior with other instances

Object

Take this cat...

It has qualities (***attributes***)
white
long-haired
4 years old

And can do things (***methods***)
walk
eat
meow
nap

Methods

- Defines a behavioral characteristic
- What the things of the class's type ***do***.
  - Chase
  - Drive
  - Talk
- The "verbs"

Methods

- Defined within a ***class***
- Store instructions to execute on ***attributes***
- keywords `def` and `end` start and end a ***method***
- Every method evaluates to something
  - `return` keyword not required
  - last statement

use # at the beginning of a line to write a **comment**
(Ruby will ignore everything on the line after the #)
Exception?
`"#{variable}"`

Variable

- Defines ***attribute*** characteristics
- What things of the class' type ***have***
  - Breed
  - Model Year
  - Favorite Ice Cream

Instance

- A specific incarnation of a class
  - Rin Tin Tin
  - garbage truck
  - the neighbor's kid

Coffee Class

```
class Coffee

end

c = Coffee.new
puts c

#<Coffee:0x007ffb1d0b6290>
 => nil
```

Coffee Class

```
class Coffee
    def initialize
        puts "Coffee is created"
    end
end

c = Coffee.new
Coffee is created
 => #<Coffee:0x007ffb1d09ba08>
```

Coffee Class

```
class Coffee
  def initialize
    @temperature = 0
    @flavor = 'sweet, smoky, Sumatran’
  end
end

myCoffee = Coffee.new
=> #<Coffee:0x007ffb1d025cb8
@temperature=0, @flavor="sweet, smoky,
Sumatran">
```

Constructor Overloading

```
class Coffee
  def initialize(temp = 0, flavor = 'bland')
    @temperature = temp
    @flavor = flavor
  end
end

waynes_coffee = Coffee.new(80, 'spicy')
 => #<Coffee:0x007ffb1b8219c8 @temperature=80,
@flavor="spicy">
brandons_coffee = Coffee.new(90)
 => #<Coffee:0x007ffb1d0f0030 @temperature=90,
@flavor="bland">
```

Variable Scope

local variable
`temperature`

instance variable
`@temperature`

class variable
`@@temperature`

global variable
`$temperature`

constant
`TEMPERATURE`

Manipulating Values

Use the "!" operator

```
def temp!(temp)
      @temperature = temp
end

myCoffee.temp!(120)
puts myCoffee.temp

yourCoffee = Coffee.new
puts yourCoffee.temp
```

Existential Operator

```
def hot?(temp)
  if temp > 160
    return true
  end

  false
end
```

Method Chaining

Do a series of tasks in order (left-to-right)
`task.try.tryAgain.success?`

First, `task.try` executes, then `result.tryAgain`

```
task.try.tryAgain.success?
result.tryAgain.success?
result.success?
```

Inheritance

- A relation between two classes
  - Cats are mammals, all mammals are animals
- Classes lower in the hierarchy 'inherit' features
  - If all mammals can breathe, then all cats can breath
- Only ***one*** level of inheritance!!!

```
class < parent
```

Inheritance

```ruby
class Drink

  def initialize
    @container = 'can'
    @material = 'aluminum'
  end

  def get_container
    @container
  end

  def get_material
    @material
  end

end
```

```ruby
class Coffee < Drink

  def initialize
    @container = 'mug'
    @material = 'ceramic'
    @flavor = 'sumatran'
  end

  def get_flavor
    @flavor
  end

end
```

```ruby
myCoffee = Coffee.new
puts "The #{myCoffee.get_material}
#{myCoffee.get_container} has #{myCoffee.get_flavor} coffee
```

Inheritance

```
myCoffee = Coffee.new
puts "The #{myCoffee.get_material}
#{myCoffee.get_container} has #{myCoffee.get_flavor} coffee
in it."
```

Modules

- Group **methods**, **classes**, and **constants**
- **Namespace** to prevent name clashes
- Implement ***mixin*** facility
- Declared with `module` keyword

```
module Hilt
    ...
end
```

Mixins

A trick to eliminate "multiple inheritance"

Example

Documentation

- Explain what the code is intended to do
- Reminders to yourself on what it does
- If you can't explain it easily, rewrite the code

Development Cycle...

- Works: http://gist.github.com/649456
- Better: http://gist.github.com/649460
- Not Embarrasing: http://gist.github.com/649480

You can run the default documentor on the third one:

```
rdoc sound3.rb
```

http://people.virginia.edu/~wsg4w/rdoc/Virgo.html

another style:

http://people.virginia.edu/~wsg4w/yard/Virgo.html

Questions?

Afternoon (and beyond) Hacks

- TryRuby
- Personal Chef (start at 11. Objects, Attributes, and Methods)
- Encryptor Lab
- Event Manager Lab
- RSpec and BDD
- EventReporter
- RSpec
- Learn Ruby the Hard Way
- Ruby Koans

I'm Stuck

- IRC (#hilt on freenode)
- Ask someone around you
- Google
- Take a break
- Raise your hand