Ruby Classes and Objects

Language Design – Classes

- Classes and objects (vs. prototypes)
- Instance variables/encapsulation
- Objects
  - Creation
  - Equality/comparison
  - Type
    - Type-safety
    - Type conversions
- Classes
  - Methods
  - Variables
- Inheritance – another lecture

Encapsulation

- Protect instance variables from outside forces
  - i.e., other classes
- Ruby is *strictly encapsulated*
  - Always private
  - *Must* include getters/setters
  - **Encapsulated:** other classes can't access data directly
- Reflection and open classes subvert this encapsulation

*covered later

Compare to Java

- Package by default (TODO: ?)
- Can declare public
- Best practice: Make instance variables private
- Why?
  - Provide data validity (can only update via setter)
  - Easy to change internal data representation (getters act as interface)

Based on the above, Java is *not* strictly encapsulated

Object Creation

- Every class inherits method `new`
  - Calls `allocate` to get space (cannot override)
  - Calls `initialize` to create instance variables
- Often convenient to provide default parameters for `initialize`

```
def initialize(x, y, z=nil)
    @x, @y, @z = x, y, z
end
```

Simple Class

```
class Cat
    def initialize(name)
        @name = name
    end
    def to_s
        "Cat: #{@name}"
    end
    def name
        @name
    end
    def name=(input)
        @name=input
    end
end
```

```
c = Cat.new("Fluffy")
puts c
puts c.name
c.name = "Bob"
```

- `new` calls `initialize`
- `to_s` like Java's `toString`
- Result of *last expression* in function is *returned*
- `@` indicates *instance variable*

Simple Class

You can see a slightly more complete `Cat` class here: Cat

Operator Overloading, etc.

Check out this Ruby class: [Bottle](demo_files