Ruby Modules

CSCI400

7 September 2017

Color Key

- Clickable URL link
- Write down an answer to this for class participation
- Just a comment – don't confuse with yellow

Example Code Files

Download Ruby module examples

Open Classes

Open Classes

Client can *append* to class definition

- Add methods and variables
- *Can overwrite existing methods*

Open Classes and Encapsulation

Open Classes Example

```ruby
# ruby_open_classes-1.rb
class Pikachu
  attr_accessor :name, :level
  def initialize(name, level)
    @name, @level = name, level
  end
end
```

Open Classes Example

```ruby
# ruby_open_classes-1.rb
class Pikachu
  attr_accessor :name, :level
  def initialize(name, level)
    @name, @level = name, level
  end
end
```

```ruby
# ruby_open_classes-2.rb
require_relative "ruby_open_classes-1.rb"
class Pikachu
  def attack
    puts "#{name} used thundershock!"
  end
end
```

Liskov Substitution Principle

- *Subtypes must be substitutable* for base class
  - Goal: Don't break when calling parent method on child
- Open classes
  - Objects must behave similarly after changes to class

LSP Violation Example 1

```
# run in irb
puts (1/2)
require "mathn"
puts (1/2)
```

What class was modified in this example?

LSP Violation Example 2

```ruby
# cat.rb
class Cat
  attr_accessor :age
  def old?
    @age > 10
  end
end
```

LSP Violation Example 2

```ruby
# cat.rb
class Cat
  attr_accessor :age
  def old?
    @age > 10
  end
end
```

```ruby
# main.rb
require_relative "cat.rb"
class Cat
  def old
    puts "Cats never grow old..."
  end
end
```

Modules

Modules

Named group of methods, constants, and class variables

- *All* classes are modules
- *Not* all modules are classes
  - Can't be *instantiated* or *subclassed*
- Purpose
  - *Namespaces* and *mixins*

Purpose 1: Namespace

Skim this

How do we specify a namespace in:

1. Java?
2. C++?

For exam: Care more about purpose, not syntax

Namespace in Ruby

- Class methods have implicit namespace
- What about *non-object* functions?
  - Java: Create a class with static method
    - e.g. `Math.abs`
  - Ruby
    - Global methods → *naming conflicts*
    - Solution: Module (provides namespace)

Example from Text

```
module Base64
  def self.encode(data)
    # ...
  end
  def self.decode(text)
    # ...
  end
end # end of module
```

```
text = Base64.encode(data)
data = Base64.decode(text)
```

See: ruby_modules-1.rb

Purpose 2: Mixins

Skim this

And this

- What's one purpose/goal of a mixin?
- What is the *diamond problem* and how do mixins avoid it?
- Think: How is this similar/different from Java interfaces?

Above questions are fair exam material

Multiple Inheritance: Problem

Child with *two parents* (e.g. C++)

- Purpose: Child has is-a relationship with both
- Issue: Parents implement same method (*diamond problem*)
- Solution: Limit classes to single parent

Alternative to Multiple Inheritance

- Class incorporates methods from multiple entities. . .
  - . . . *without* entire class hierarchy
- Examples
  - Java: interfaces
  - Ruby: *Modules as mixins*

Module Mixins

Mixins in Ruby

- Module with instance methods
  - Can be *mixed into* other classes
  - Eliminates need for multiple inheritance
- Examples
  - `Enumerable`
    - Defines iterators that make use of `each`
  - `Comparable`
    - Defines `<`, `>`, `==`, etc. in terms of `<=>`

Mixin Example 1 (1/2)

```ruby
module FlyingCreature
  def fly
    puts "Flying! Speed: #{@speed}"
  end
end
```

See: ruby_modules-2.rb

Mixin Example 1 (1/2)

```ruby
module FlyingCreature
  def fly
    puts "Flying! Speed: #{@speed}"
  end
end

class Bird
  include FlyingCreature
  def initialize
    @speed = 5
  end
end
```

See: ruby_modules-2.rb

Mixin Example 1 (2/2)

```
tweety = Bird.new
tweety.fly
puts "tweety is a flying creature? "\
     "#{tweety.is_a? FlyingCreature}"
```

See: ruby_modules-2.rb

Mixin Example 2 (1/2)

```ruby
module FlyingCreature
  def fly
    puts "Flying! Speed: #{@speed}"
  end
end
```

See: ruby_modules-2.rb

Mixin Example 2 (1/2)

```ruby
module FlyingCreature
  def fly
    puts "Flying! Speed: #{@speed}"
  end
end

class Mammal
  # ...
end
class Bat < Mammal
  include FlyingCreature
  # ...
end
```

See: ruby_modules-2.rb

Mixin Example 2 (2/2)

```
bat = Bat.new
bat.fly
puts "bat is a flying creature? "\
     "#{bat.is_a? FlyingCreature}"

batman = Mammal.new
puts "batman is a flying creature? "\
     "#{batman.is_a? FlyingCreature}"
```

See: ruby_modules-2.rb

Mixins: Best Practice

Skim

- When should you *not* use a class?
- Do we create instances of modules?
- How do we use module instance methods?

Other

Modules vs. Interfaces

On your own: Ruby Modules vs. Java Interfaces

Not exam material

Singleton Method Example

```
class Person
  def initialize(name)
    @name = name
  end
end
```

```
shugo = Person.new("Shugo")
matz = Person.new("Matz")

def matz.design_ruby; puts "I designed Ruby!"

matz.design_ruby
shugo.design_ruby # error
```