Today’s Agenda

• Double Dispatch Again
• Mixins
• The Visitor Pattern

Dispatch Overview

Dispatch is the runtime procedure for looking up which function to call based on the parameters given:

• Ruby (and Java) use Single Dispatch on the implicit self parameter
  • Uses runtime class of self to lookup the method when a call is made
  • This is what you learned in CSE 143
• Double Dispatch uses the runtime classes of both self and a single method parameter
  • Ruby/Java do not have this, but we can emulate it
  • You can dispatch on any number of the parameters and the general term for this is Multiple Dispatch or Multimethods

Emulating Double Dispatch

• To emulate double dispatch in Ruby (on HW7) just use the built-in single dispatch procedure twice!
  • Have the principal method immediately call another method on its first parameter, passing self as an argument
  • The second call will implicitly know the class of the self parameter
  • It will also know the class of the first parameter of the principal method, because of Single Dispatch
  • There are other ways to emulate double dispatch
    • Found as an idiom in SML by using case expressions

Double Dispatch Example

```ruby
class A
  def f x
    x.fWithA self
  end

  def fWithA a
    "(a, a) case"
  end

  def fWithB b
    "(b, a) case"
  end
end

class B
  def f x
    x.fWithB self
  end

  def fWithA a
    "(a, b) case"
  end

  def fWithB b
    "(b, b) case"
  end
end
```

Mixins

• A mixin is (just) a collection of methods
  • Less than a class: no instances of it
• Languages with mixins (e.g., Ruby modules) typically let a class have one superclass but include any number of mixins
• Semantics: Including a mixin makes its methods part of the class
  • Extending or overriding in the order mixins are included in the class definition
  • More powerful than helper methods because mixin methods can access methods (and instance variables) on self not defined in the mixin
Mixin Example

```ruby
module Doubler
  def double
    self + self # assume included in classes w/ +
  end
end

class String
  include Doubler
end

class AnotherPt
  attr_accessor :x, :y
  include Doubler
  def + other
    ans = AnotherPt.new
    ans.x = self.x + other.x
    ans.y = self.y + other.y
    ans
  end
end
```

Method Lookup Rules

Mixins change our lookup rules slightly:
- When looking for receiver `obj`'s method `m`, look in `obj`'s class, then mixins that class includes (later includes shadow), then `obj`'s `superclass`, then the `superclass'` mixins, etc.
- As for instance variables, the mixin methods are included in the same object
  - So usually bad style for mixin methods to use instance variables since names can clash

The Two Big Ones

The two most popular/useful mixins in Ruby:
- Comparable: Defines `<`, `>`, `==`, `!=`, `>=`, `<=` in terms of `<=>`
  - [http://ruby-doc.org/core-2.2.3/Comparable.html](http://ruby-doc.org/core-2.2.3/Comparable.html)
- Enumerable: Defines many iterators (e.g., `map`, `find`) in terms of `each`
  - [http://ruby-doc.org/core-2.2.3/Enumerable.html](http://ruby-doc.org/core-2.2.3/Enumerable.html)
- Great examples of using mixins:
  - Classes including them get a bunch of methods for just a little work
  - Classes do not “spend” their “one superclass” for this
  - Does not bring on the complexity of multiple inheritance

The Visitor Pattern

- A template for handling a functional composition in OOP
  - OOP wants to group code by classes
  - We want code grouped by functions
    - This makes it easier to add operations at a later time.
- Relies on Double Dispatch!!!
  - Dispatch based on (VisitorType, ValueType) pairs.
- Often used to compute over AST’s (abstract syntax trees)
  - Heavily used in compilers