# CSE 341 AA: Section 9

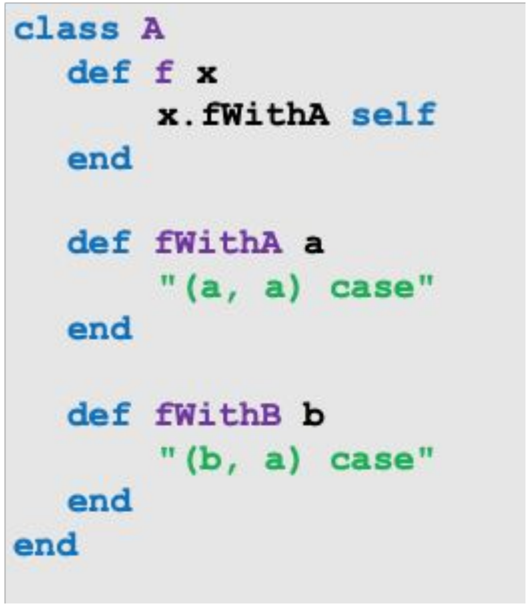
Porter Jones
pbjones@cs.washington.edu
Office Hours: Thursdays 5:30 - 7:30pm

## Double Dispatch

- Ruby and Java both use *single dispatch*, which uses the runtime class of self to lookup the method when a call is made

- *Double dispatch* uses the runtime classes of both self and a single method parameter to lookup the method when a call is made
    - Ruby and Java don't have double dispatch, but it's possible to emulate it by using the single dispatch twice (example on next slide)
    - You get to do this on HW 7!

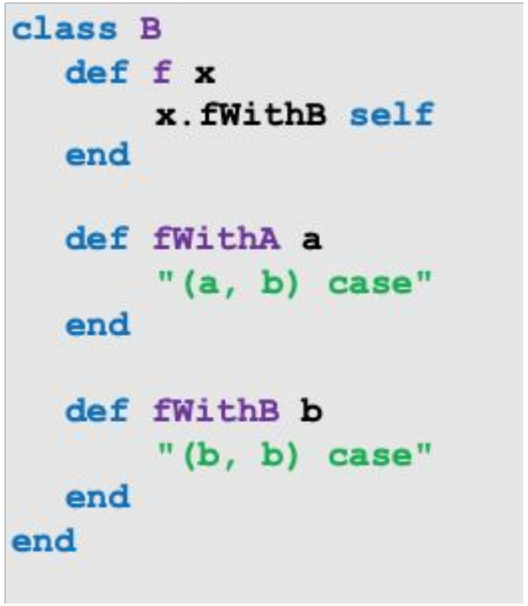
# Double dispatch emulation



```
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 in Ruby

- A mixin is a collection of methods
  - You can’t create instances of a mixin
  - Typically languages with mixins let a class have one superclass and any number of mixins

- Including a mixin makes its methods part of the class
  - Order of includes matters for extending/overriding
  - Mixins can access methods/instance variables defined in a class

## Mixins Example

```ruby
class Pt
  attr_accessor :x, :y
  include Comparable # Defines <, >, ==, !=, >=, <= in terms of <=>

  def distance
    Math.sqrt (@x ** 2 + @y ** 2)
  end

  def <=> other
    self.distance <=> other.distance
  end
end
```

## Mixins: 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

## 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

## Visitor Example

```
class Add
  attr_reader :e1, :e2
  def initialize(e1,e2)
    @e1 = e1
    @e2 = e2
  end
  def accept(visitor, arg=nil)
    visitor.visitAdd(self, arg)
  end
end
```

```
class TypeChecker
  def visitAdd(add, arg)
    t1 = add.e1.accept(self)
    t2 = add.e2.accept(self)
    # Make sure t1 and t2 are ints
    # return the type int
  end
end
```