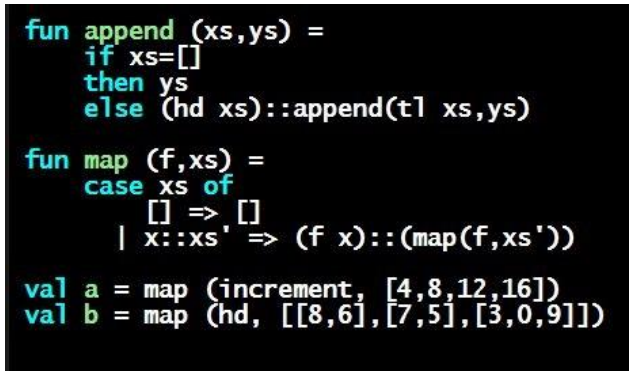

Programming Languages

Dan Grossman

2013

Optional: Multimethods

Being Fair

Belittling OOP style for requiring the manual trick of double dispatch is somewhat unfair…

What would work better:

- **`Int`**, **`MyString`**, and **`MyRational`** each define three methods all named **`add_values`**
  - One **`add_values`** takes an **`Int`**, one a **`MyString`**, one a **`MyRational`**
  - So 9 total methods named **`add_values`**
  - **`e1.eval.add_values e2.eval`** picks the right one of the 9 at run-time using the classes of the two arguments
- Such a semantics is called *multimethods* or *multiple dispatch*

Multimethods

General idea:

- Allow multiple methods with same name
- Indicate which ones take instances of which classes
- Use dynamic dispatch on arguments in addition to receiver to pick which method is called

If dynamic dispatch is essence of OOP, this is more OOP

- No need for awkward manual multiple-dispatch

Downside: Interaction with subclassing can produce situations where there is “no clear winner” for which method to call

Ruby: Why not?

Multimethods a bad fit (?) for Ruby because:

- Ruby places no restrictions on what is passed to a method
- Ruby never allows methods with the same name
  - Same name means overriding/replacing

Java/C#/C++: Why not?

- Yes, Java/C#/C++ allow multiple methods with the same name
- No, these language do *not* have multimethods
  - They have *static overloading*
  - Uses static types of arguments to choose the method
    - But of course run-time class of receiver [odd hybrid?]
  - No help in our example, so still code up double-dispatch manually
- Actually, C# 4.0 has a way to get effect of multimethods
- Many other languages have multimethods (e.g., Clojure)
  - They are not a new idea