CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 13 Double Dispatch
Contents

References

Ralph Johnson's Object-Oriented Programming & Design lecture notes, Polymorphism (Day 5) http://st-www.cs.uiuc.edu/users/cs497/lectures.html

VisualWorks Source Code

Double Dispatch
Multiplication Motivation

Integer * Integer
  Primitive integer * integer operation

Integer * Float
  Convert integer to float
  Use primitive float * float operation

Float * Integer
  Convert integer to float
  Use primitive float * float operation

Integer * Matrix
  Multiple each element of Matrix by integer

Matrix * Integer
  Multiple each element of Matrix by integer

Integer * Fraction
  Multiple Fraction numerator by integer

Actual operation depends on the type of both arguments

How to implement?

Double Dispatch

```
Integer>>+ aNumber
   “We do not know what aNumber is”
   “Send message to aNumber telling it we are an integer”

   ^aNumber sumFromInteger: self

Float>>sumFromInteger: anInteger
   “Now we know the type of the receiver and sender
   Do the correct thing for this pair”

   ^anInteger asFloat + self

Float+ aNumber
   "Answer a Float that is the result of adding the receiver to the
argument.
   The primitive fails if it cannot coerce the argument to a Float"

   <primitive: 41>
   ^aNumber sumFromFloat: self


Double>>sumFromInteger: anInteger

   ^anInteger asDouble + self
```

Example

3 + 2.5

```
Integer>>+ aNumber

   ^aNumber sumFromInteger: self
```

Then

2.5 sumFromInteger: 3

```
Float>>sumFromInteger: anInteger

   ^anInteger asFloat + self
```

3.0 + 2.5

```
Float+ aNumber

   <primitive: 41>
   ^aNumber sumFromFloat: self
```

primitve 41 adds 3.0 and 2.5 and returns 5.5

Triple Dispatching?

Why does Float>>sumFromInteger: send another message?

It has all the information it needs

It could perform the operation there, but it is easier to just call +

Adding a New Type Of Arithmetic Value

New type X

Add primary methods: + - / *

Add double displatching methods
- sumFromDouble:
- sumFromFloat:
- sumFromFraction:
- sumFromInteger:
- sumFromX:

Same with quotentFrom, productFrom, differenceFrom

Existing Types

Add double displatching methods
- sumFromX:
- quotentFromX:
- productFromX:
- differenceFromX:

Shared Responsibilities

Sometimes an operation depends on the class of several objects

Arthmetic – depends on the types of both arguments

Displaying object - depends on type of object and windowing system

Singleton - One Instance

Sometimes need to insure only one instance of a Class

```
Smalltalk defineClass: #SingletonExample
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: 'uniqueInstance '
   imports: ''
   category: 'CS535'


SingletonExample class methodsFor: 'instance creation'

current
   uniqueInstance ifNil: [uniqueInstance := self basicNew].
   ^uniqueInstance

new
   "Force all creation access through current to insure only one
instance"

   self shouldNotImplement
```