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

Contents

Double Dispatch .......................................................... 2
Singleton - One Instance ................................................. 8

References

Ralph Johnson’s Object-Oriented Programming & Design

VisualWorks Source Code

Copyright ©, All rights reserved. 2003 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
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

primitive 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 disiplatching methods
• sumFromDouble:
• sumFromFloat:
• sumFromFraction:
• sumFromInteger:
• sumFromX:

Same with quotientFrom, productFrom, differenceFrom

Existing Types

Add double disiplatching methods
• sumFromX:
• quotientFromX:
• productFromX:
• differenceFromX:
Shared Responsibilities

Sometimes an operation depends on the class of several objects

Arithmetic – 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