

CS 535 Object-Oriented Programming & Design
Fall Semester, 2013
Doc 5 Control Messages & Classes
Sept 5 2013

Copyright ©, All rights reserved. 2013 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

References

Ralph Johnson's University of Illinois, Urbana-Champaign CS 497 lecture notes,
<http://st-www.cs.uiuc.edu/users/cs497/>

Smalltalk Best Practice Patterns, Beck

Smalltalk With Style, Klimas, Skublics, Thomas

if

(boolean expression) ifTrue: trueBlock

(boolean expression) ifFalse: falseBlock

(boolean expression) ifFalse: falseBlock ifTrue: trueBlock

(boolean expression) ifTrue: trueBlock ifFalse: falseBlock

```
a < 1 ifTrue: [Transcript show: 'hi mom' ]
```

```
difference := (x > y)
```

```
    ifTrue: [ x - y]
```

```
    ifFalse: [ y - x]
```

Boolean Expressions

	Symbol	Example
Or		a b
And	&	a & b
Exclusive or	xor:	a xor: (b > c)
Negation	not	(a < b) not

Lazy Logical Operations

	Message	Example
Or	or: aBlock	a or: [b > c]
And	and: aBlock	a and: [c b]

This is not C

This is a runtime error

```
5 ifTrue: [1 + 3]
```

A Style Issue

Both do the same thing

```
difference := (x > y)
  ifTrue: [ x - y]
  ifFalse: [ y - x]
```

```
(x > y)
  ifTrue: [difference := x - y]
  ifFalse: [difference := y - x]
```

isNil

Answers true if receiver is nil otherwise answers false

x isNil

ifTrue: [do something]

ifFalse: [do something else]

Shortcuts

ifNil:ifNotNil:

ifNotNil:ifNil:

ifNil:

ifNotNil:

x

ifNil: [do something]

ifNotNil: [do something else]

isNil Implementation

```
Object>>isNil  
^false
```

```
UndefinedObject>>isNil  
^true
```

nil is single instance of UndefinedObject

isNil - How it Works

```
| foo |  
foo := 5.  
foo isNil.  
foo := nil.  
foo isNil
```

```
Object>>isNil  
^false
```

```
UndefinedObject>>isNil  
^true
```

Replace cases with Polymorphic messages

```
if (x.class == Bar) then
  do the bar stuff
else
  do the foo stuff
```

```
Bar>>dolt
  do the bar stuff
```

```
Foo>>dolt
  do the foo stuff
```

```
x dolt
```

Blocks

A deferred sequence of actions – a function without a name

Can have 0 or more arguments

Executed when sent the message 'value'

Similar to

- Lisp's Lambda- Expression

- Erlang's funs

- Ruby's Blocks

- Python's lambda

- Anonymous functions

```
[:variable1 :variable2 ... :variableN |
```

```
| blockTemporary1 blockTemporary2 ... blockTemporaryK |
```

```
expression1.
```

```
expression2.
```

```
...]
```

Blocks and Return Values

Blocks return the value of the last executed statement in the block

```
| block x |
```

```
block := [:a :b |
```

```
  | c |
```

```
  c := a + b.
```

```
  c + 5].
```

```
x := block value: 1 value: 2.
```

x has the value 8

Blocks know their Environment

```
| a b |  
a := 1.  
b := 2.  
aBlock := [a + b].  
result := aBlock value
```

result is now 3

```
| a b |  
a := 1.  
b := 2.  
aBlock := [a + b].  
a := 5  
result := aBlock value
```

result is now 6

Blocks and Arguments

Using the value: keyword message up to 4 arguments can be sent to a block.

```
[2 + 3 + 4 + 5] value
```

```
[:x | x + 3 + 4 + 5 ] value: 2
```

```
[:x :y | x + y + 4 + 5] value: 2 value: 3
```

```
[:x :y :z | x + y + z + 5] value: 2 value: 3 value: 4
```

```
[:x :y :z :w | x + y + z + w] value: 2 value: 3 value: 4 value: 5
```

valueWithArguments: can be used with 1 or more arguments

```
[:a :b :c :d :e | a + b + c + d + e ] valueWithArguments: #( 1 2 3 4 5)
```

```
[:a :b | a + b ] valueWithArguments: #( 1 2 )
```

Where is the Value Message

```
difference := (x > y)
  ifTrue: [ x - y]
  ifFalse: [ y - x]
```

In the False class we have:

```
ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock
  ^falseAlternativeBlock value
```

In the True class we have:

```
ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock
  ^trueAlternativeBlock value
```

While Loop

```
aBlockTest while True
aBlockTest while True: aBlockBody
aBlockTest while False
aBlockTest while False: aBlockBody
```

The last expression in aBlockTest must evaluate to a boolean

```
| x y difference |
x := 8.
y := 6.
difference := 0.
[x > y] while True:
    [difference := difference + 1.
     y := y + 1].
^difference
```

```
| count |
count := 0.
[count := count + 1.
count < 100] while True.
Transcript
    clear;
    show: count printString
```


More Loops

Transcript

```
clear.
```

```
3 timesRepeat:
```

```
[Transcript
```

```
  cr;
```

```
  show: 'Testing!'].
```

```
1 to: 3 do:
```

```
[:n |
```

```
  Transcript
```

```
    cr;
```

```
    show: n printString;
```

```
    tab;
```

```
    show: n squared printString].
```

```
9 to: 1 by: -2 do:
```

```
[:n |
```

```
  Transcript
```

```
    cr;
```

```
    show: n printString].
```

Transcript

```
Testing!
```

```
Testing!
```

```
Testing!
```

```
1 1
```

```
2 4
```

```
3 9
```

```
9
```

```
7
```

```
5
```

```
3
```

```
1
```

Classes

Objects & Classes - Smalltalk Language Details

Items to cover

Defining classes

Packages

Namespaces

Class names

Methods

- Instance
- Class

Variables

- Instance variables
- Class instance variables
- Shared variables

Inheritance

self & super

The Rules

Everything in Smalltalk is an object

All actions are done by sending a message to an object

Every object is an instance of a class

All classes have a parent class

Object is the root class

How do you Define a Class?

Defining Point Class

```
Smalltalk.Core defineClass: #Point
  superclass: #{Core.ArithmeticValue}
  indexedType: #none
  private: false
  instanceVariableNames: 'x y '
  classInstanceVariableNames: "
  imports: "
  category: 'Graphics-Geometry'
```

Terms

Superclass

Package (parcel)

Namespace

Class Names & Namespaces

Classes are defined in a namespace

Classes in different namespaces can use the same name

Full name of a class includes namespace

Root.Smalltalk.Core.Point

Use import to use shorter names

Workspace windows import all namespaces

Methods

All methods return a value

All methods are public

Placed a method in the "private" category to tell others to treat it as private

Instance methods

Sent to instances of Classes

1 + 2

'this is a string' reverse

Class Methods

Sent to Classes

Commonly used to create instances of the class

Array new

Point x: 1 y: 3

Float pi

Convention

ClassName>>methodName

String>>reverse

Point class>>x:y:

Naming Conventions

Class Names

Use complete words, no abbreviations

First character of each word is capitalized

SmallInteger

LimitedWriteStream

LinkedMessageSet

Simple Superclass Name

Simple words

One word preferred, two at maximum

Convey class purpose in the design

Number

Collection

Magnitude

Model

Qualified Subclass Name

Unique simple name that conveys class purpose

When name is commonly used

Array

Number

String

Prepend an adjective to superclass name

Subclass is conceptually a variation on the superclass

OrderedCollection

LargeInteger

CompositeCommand

Class Names and Implementation

Avoid names that imply anything about the implementation of a class

"A proper name that is stored as a String"

ProperName

~~ProperNameString~~

"A database for Problem Reports that uses a Dictionary"

ProblemReportDatabase

~~ProblemReportDictionary~~

"Not implemented with a Set, it is a specialized Set"

SortedSet

Method Names

Always begins with a lowercase first letter

Don't abbreviate method names

Use uppercase letters for each word after the first

Method Naming Guidelines

Choose method names so that statements containing the method read like a sentence

FileDescriptor seekTo: work from: self position

Use imperative verbs and phrases for methods which perform an action

Dog

sit;

lieDown;

playDead.

aFace lookSuprised

~~aFace surprised~~

Method Naming Guidelines

Use a phrase beginning with a verb (is, has) when a method returns a boolean

isString

aPerson isHungry

~~aPerson hungry~~

Use common nouns for methods which answer a specific object

anAuctionBlock nextItem

~~anAuctionBlock item~~

"which item"

Method Naming Guidelines

Methods that get/set a variable should use the same name as the variable

books
^books

~~getBooks~~
^books

books: aCollection
books := aCollection

~~setBooks: aCollection~~
books := aCollection

Inheritance

Inheritance

Smalltalk supports only single inheritance

Each class has single parent class

A class inherits (or has) all

- Methods defined in its parent class

- Methods defined in its grandparent class

- etc.

- Methods defined in any ancestor class

- Variables defined in any ancestor class

Terms

Parent Class
Superclass

Child class
Subclass

Object

Is the ancestor of all classes

Has no parent class

Contains important methods for all classes & objects

Inheritance and Name Clashes

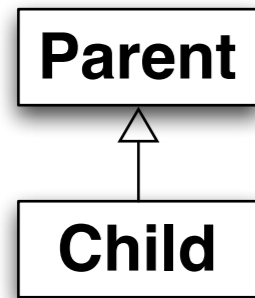
Subclass can implement methods with same name as parent

This is called overriding the method

When message is sent to instance of the subclass, the subclass method is used

Subclass can not overload/override variable names

Example



Parent>>foo
^'foo'

Child>>foo
^'bar'

	Result
aParent aChild	
aParent := Parent new.	
aChild := Child new.	
aParent foo.	'foo'
aChild foo.	'bar'

Types of Variables

Temporary (Local) Variable

Named Instance Variable

Class Instance Variable

Shared Variable

Indexed Instance Variable

Temporary (Local) Variable

```
| a b sum |  
a := 5.  
b := 10.  
sum := a + b.
```

```
Point>>grid: aPoint  
"Answer a new Point to the nearest rounded grid modules  
specified by aPoint."  
| newX newY |  
aPoint x = 0  
  ifTrue: [newX := 0]  
  ifFalse: [newX := x roundTo: aPoint x].  
aPoint y = 0  
  ifTrue: [newY := 0]  
  ifFalse: [newY := y roundTo: aPoint y].  
^newX @ newY
```

Usage Convention

Do not use the same temporary variable name within a scope for more than one purpose

```
| aRecord |  
aRecord := self indexRecord.  
aRecord lock: 12.  
aRecord := aRecord at: 12.  
self update: (aRecord at: 1) with: self newData.  
aRecord unlock: 12.
```

Named Instance Variable

Each object has its own copy of a named instance variable

Like

- Protected C++ data member

- Protected Java field

Accessible by

- Instance methods of the class

- Instance methods of subclasses of the class

Not accessible by

- Methods in non-subclasses

- Class methods

Example

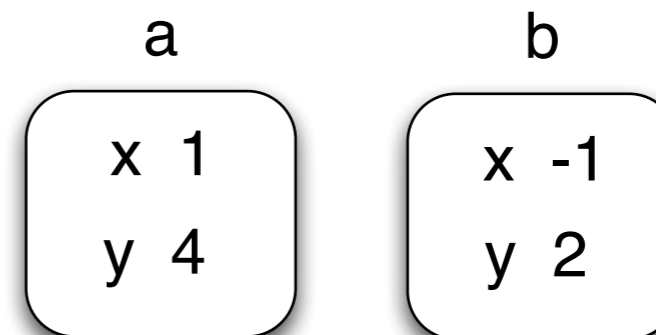
```
Smalltalk defineClass: #ClassPoint
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'x y '
  classInstanceVariableNames: "
  imports: "
  category: "
```

```
ClassPoint >>y: aNumber
  y := aNumber
```

```
ClassPoint >>x: aNumber
  x := aNumber
```


Example

```
| a b |  
a := ClassPoint new.  
a  
  x: 1;  
  y: 4.  
b := ClassPoint new.  
b  
  x: -1;  
  y: 2.
```



Adding Removing Instance Variables

Method 1 Edit Class Definition

```
Smalltalk defineClass: #ClassPoint
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'x y z w '
  classInstanceVariableNames: "
  imports: "
  category: "
```

Adding/Removing Instance Variables

Method 2: Use Browser's Class menu

