CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 4 Classes Part 2

Contents

Objects & Classes - Smalltalk Language Details .................................................. 3
Class Names ........................................................................................................ 4
  Simple Superclass Name .................................................................................. 5
  Qualified Subclass Name .............................................................................. 6
  Class Names & Namespaces ........................................................................ 7
Methods ........................................................................................................... 9
  Instance methods ......................................................................................... 10
  Class methods ............................................................................................... 11
Variables ......................................................................................................... 12
  Named Instance Variable ............................................................................ 13
  Class Instance Variable ............................................................................. 18
  Shared Variables .......................................................................................... 21
  Indexed Instance Variable .......................................................................... 30
Inheritance ....................................................................................................... 33
Special or PseudoVariables ............................................................................ 36
Implicit Return Values .................................................................................... 43
Initializing Instance Variables ........................................................................ 44

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

Reading

Application Developer’s Guide is the file AppDevGuide.pdf in the docs directory
of the VisualWorks directory. On Rohan this file is
/opt/smalltalk/vw7nc/doc/AppDevGuide.pdf

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.
VisualWorks Menu Shortcut

When you install VisualWorks on Windows it creates a short cut on the start menu

This short cut only opens the image called visual.im in the image directory

If you save an image under a different name you will not be able to use that image using the start menu short cut

I never use the start menu short cut for VisualWorks

I always start VisualWorks by double-clicking on the image I wish to run

The first time I have to bind the image file to visual.exe
Objects & Classes - Smalltalk Language Details

Items to cover:

Class names

Methods
• Instance
• Class

Variables
• Instance variables
• Class instance variables
• Shared variables

Inheritance

self & super
Class Names

Smalltalk class names:

• Use complete words, no abbreviations

  Names are read 100 to 1000 times more often than typed

  Abbreviations waste more time (reading) than they save

• First character of each word is capitalized

  SmallInteger, LimitedWriteStream, LinkedMessageSet
Simple Superclass Name

Superclass names

• 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
  
  If name is in common use
    
    Array, Number, String
  
  If the purpose is more important than class hierarchy

• Prepend an adjective to superclass name
  
  If the class hierarchy is important
    
    Subclass is conceptually a variation on the superclass

OrderedCollection, LargeInteger, CompositeCommand
Class Names & Namespaces

Classes are defined in a namespace

Allows classes in different namespaces to use the same name

Full name of a class includes namespace

   Root.Smalltalk.Core.Point is full name of Point class

The import mechanism allows one to use shorter names

Workspace windows import all namespaces
Point Name Example

Each of the following is legal code in a workspace

Root.Smalltalk.Core.Point
   x: 1
   y: 1.

Smalltalk.Core.Point
   x: 1
   y: 1.

Core.Point
   x: 1
   y: 1.

Point
   x: 1
   y: 1.
Methods

All methods are public

Methods considered private are placed in the method category called "private"

All methods return a value
Instance methods

Sent to instances of Classes

Examples
  1 + 2

'this is a string' reverse
Class methods

Sent to Classes

Can not be sent directly to instance of the class

Commonly used to create instances of the class

Examples

Point new

Point x: 1 y: 3

Float pi
Variables

Types of Variables

- Named Instance Variable
- Class Instance Variable
- Shared Variable
- Temporary Variable
- Indexed Instance Variable
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.Core defineClass: #Point
   superclass: #{Core.ArithmeticValue}
   indexedType: #none
   private: false
   instanceVariableNames: 'x y '  
   classInstanceVariableNames: ""
   imports: ""
   category: 'Graphics-Geometry'

x & y are instance variables

| a b |

a := Point
   x: 1
   y: 4.

b := Point
   x: -1
   y: 2.

We now have two point objects. Each point object has a local copy of x and y. Values in the local copies are different.
Accessing Instance variables in an Instance method

Point instance method example

Point>>dotProduct: aPoint

∧(x * aPoint x) + (y * aPoint y)
Adding an Instance Variables to a Class

To add an instance variable to a class,

Add the variable name in the string argument of instanceVariableNames:

Example

To add a z to the point class, just add z to the string

Smalltalk.Core defineClass: #Point
  superclass: #{Core.ArithmeticValue}
  indexedType: #none
  private: false
  instanceVariableNames: 'x y z '
  classInstanceVariableNames: "
  imports: "
  category: 'Graphics-Geometry'
Removing an Instance Variables to a Class

Remove the variable name from the class definition

Make sure no methods still use the variable
Class Instance Variable

A class has one instance

Each subclass has a different instance

Accessible by
  Class methods
  Class methods of subclasses
Example
Class Definitions

Smalltalk.Core defineClass: #ClassInstanceVariableExample
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: 'test'
  imports: "
  category: 'As yet unclassified'

ClassInstanceVariableExample class>>test
  test isNil ifTrue:[ test := 0].
  test := test + 1.
  ^test

Smalltalk.Core defineClass: #ClassInstanceVariableChild
  superclass: #{Core.ClassInstanceVariableExample}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: "
  category: 'As yet unclassified'
The Test

ClassInstanceVariableExample test.

Transcript
  print: ClassInstanceVariableExample test;
cr;
  print: ClassInstanceVariableChild test;
cr;
  flush

Output in Transcript

2
1
Shared Variables

Only one copy exists of each shared variable

Multiple classes and object can access the same shared variable
Adding a Shared Variable to a Class

In a browser select an existing class.

Click on the "shared variables" radio button in the browser.

Now use the "Protocol" menu to add a new category for the shared variables.

You get the template

```
SharedVariableClass defineSharedVariable: #NameOfBinding
  private: false
  constant: false
  category: 'accessing'
  initializer: 'Array new: 5'
```

Change `NameOfBinding` to the name of the variable.

Keep the "#" before the name.

Change the other values as appropriate.
Private and Public Shared Variables

A shared variable can be accessed from any class

Private shared variable access

In the class just use short name of the variable

Outside the class use the fully qualified name

Public shared variable access

In the class & subclasses use short name

Outside the class

Use fully qualified name or

Use import to shorten the name

Private does not seem to be the correct term
Example

Smalltalk.Core defineClass: #SharedVariableExample
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: "
   classInstanceVariableNames: "
   imports: "
   category: 'Course-Examples'

Core.SharedVariableExample defineSharedVariable: #PublicVariable
   private: false
   constant: false
   category: 'data'
   initializer: '3'

Core.SharedVariableExample defineSharedVariable: #PrivateVariable
   private: true
   constant: false
   category: 'data'
   initializer: '5'

SharedVariableExample Instance Methods

privateVariable
   ^PrivateVariable

privateVariable: anObject
   PrivateVariable := anObject

publicVariable
   ^PublicVariable

publicVariable: anObject
   PublicVariable := anObject
**Subclass Accessing Shared Variables**

Smalltalk.Core defineClass: #SharedVariableChild
  superclass: #{Core.SharedVariableExample}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: "
  category: 'Course-Examples'

Core.SharedVariableChild methodsFor: 'accessing'

childPublic
  ^PublicVariable

childPrivate
  "Subclass can not use short name on private"
  ^Core.SharedVariableExample.PrivateVariable
Other Class Accessing Shared Variables

Smalltalk.Core defineClass: #SharedVariableUser
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: ""
  classInstanceVariableNames: ""
  imports: 'Core.SharedVariableExample.*'
  category: 'Course-Examples'

Core.SharedVariableUser methodsFor: 'accessing'

privateVariable
  "Can not use short name on private variables"
  ^Core.SharedVariableExample.PrivateVariable

publicVariable
  ^PublicVariable
Initializing Shared Variables

Core.SharedVariableExample defineSharedVariable: #PublicVariable
  private: false
  constant: false
  category: 'data'
  initializer: '3'

Shared variables have an initializer

The initializer by:

- "Initialize Variable" item in the browser "Method" menu
- Sending a message

  #{Core.SharedVariableExample.PublicVariable} initialize
Constant Verses Non-Constant Shared Variables

A constant shared variable's value:

- Is set by an initializer
- Can not be changed by assigning a new value

Non-constant shared variable's value can changed

Core.SharedVariableExample.PublicVariable := 42.
Some Existing Uses of Shared Variables

Transcript

A shared variable

An object that writes to a window

Character Constants

Where are tab, space, cr, If defined?

Solution one - Class methods of Character

Character cr
Character space
Character If

Works, but is verbose

Solution Two - Shared Variables

Graphics.TextConstants defines shared variables

Tab, CR, LF, Space

To use in a class import "Graphics.TextConstants.*"
Indexed Instance Variable

Provides slots in objects for array like indexing

Used for Arrays

I have never added indexed instance variables

I have always used existing collection classes
Adding Indexed Instance Variables to a Class

Create a class using Class:Add Class:Indexed menu item in the browser.

You get the template somewhat like:

Core defineClass: #NameOfClass
    superclass: #{NameOfSuperclass}
    indexedType: #objects
    private: false
    instanceVariableNames: 'instVarName1 instVarName2'
    classInstanceVariableNames: "
    imports: "
    category: 'Course-Examples'
For this example I edited the template to be:

```plaintext
Core defineClass: #IndexedExample
    superclass: #{Core.Object}
    indexedType: #objects
    private: false
    instanceVariableNames: "
    classInstanceVariableNames: "
    imports: "
    category: 'Course-Examples'
```

Once this is saved the following works:

```plaintext
| a |

a := IndexedExample new: 10.
a
    at: 1 put: 2.
^a
```
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

Some Terminology

Parent Class
Superclass
Base Class
  Mean same thing

Child class
Subclass
Derived class
  Mean the same thing
Class "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 overloading the method

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

Subclass can not overload variable names
Special or PseudoVariables

self
Refers to the receiver of the message (current object)

Methods referenced through self are found by:

Searching the class hierarchy starting with the class of receiver

super
Refers to the receiver of the message (current object)

Methods referenced through super are found by:

Searching the class hierarchy starting the superclass of the class containing the method that references super

Called pseudo-variables because:

They do change value
You can not assign values to them
Self, Super Example

Three classes to study self & super: Parent, Child, GrandChild

Parent

Smalltalk.Core defineClass: #Parent
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: "
  category: 'Course-Examples'

Instance Methods

name
  ^'Parent'
Child

Smalltalk.Core defineClass: #Child
  superclass: #{Core.Parent}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: "
  category: 'Course-Examples'

Instance Methods

name
  ^'Child'

selfName
  ^self name

superName
  ^super name

GrandChild

Smalltalk.Core defineClass: #GrandChild
  superclass: #{Core.Child}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: "
  category: 'Course-Examples'

GrandChild Instance Methods

name
  ^'GrandChild'
Self, Super Example - Continued

Test Program

<table>
<thead>
<tr>
<th>grandchild</th>
<th>Output In Transcript</th>
</tr>
</thead>
<tbody>
<tr>
<td>grandchild := Grandchild new.</td>
<td></td>
</tr>
<tr>
<td>Transcript</td>
<td></td>
</tr>
<tr>
<td>show: grandchild name;</td>
<td>Grandchild</td>
</tr>
<tr>
<td>cr;</td>
<td></td>
</tr>
<tr>
<td>show: grandchild selfName;</td>
<td>Grandchild</td>
</tr>
<tr>
<td>cr;</td>
<td></td>
</tr>
<tr>
<td>show: grandchild superName;</td>
<td>Parent</td>
</tr>
<tr>
<td>cr.</td>
<td></td>
</tr>
</tbody>
</table>
Self, Super Example - Continued

How does this Work?

**grandchild selfName**

receiver is grandchild object

Code in selfName method is \(^\text{self name}\)

To find the method self name start search in Grandchild class

**grandchild superName**

receiver is grandchild object

Code in superName method is \(^\text{super name}\)

superName is implemented in Child class

To find the method self name start search in the superclass of Child
Why Super

Super is used when:

The child class extends the behavior of the inherited method

That is:

• Child class inherits a method, call it foo
• Child class implements a with the same name
• Child class needs to access the inherited method

In this case super is needed access the inherited method
**Why doesn't super refer to parent class of the receiver?**

Object subclass #Parent

name
`^'Parent'

Parent subclass #Child

name
`^super name , 'Child'

Child subclass #Grandchild

"No methods in Grandchild"

**Sample Program**

```
| trouble |
trouble := Grandchild new.
trouble name.
```

Assume that super did refer to the parent class of the receiver. Sending the message "name" to trouble would call the code "super name , 'Child". The super would refer to the parent class of the receiver. Since the receiver is a Grandchild object, "super name" would refer to the "name" method in the Parent class. Hence the method will call itself with no way to end.
Implicit Return Values

If a method does not explicitly return a value, self is returned

Hence a method like:

decrease
count ifNil: [count := 0].
count := count - 1

Is really:

decrease
count ifNil: [count := 0].
count := count - 1.
^self

Style Issue - When to explicitly return?

Only explicitly return a value from a method when the intent of the method is to return a value. An explicit return indicates to other programmers that the intent of the method is to compute some return value. The intent of the decrease method is to change the state of the receiver. Hence it should not have a value explicitly returned.
Initializing Instance Variables

If the instance variables always start at same value:

- Create in instance method to initialize them
- Implement class method "new" to initialize the object

Smalltalk.Core defineClass: #Counter
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'count'
  classInstanceVariableNames: ''
  imports: ''
  category: 'Course-Examples'

Instance Methods

Category: initialize

initialize
  count := 0

Category: access

count
  ^count

decrease
  count := count - 1

increase
  count := count + 1

Class Methods

Category: instance creation

new
  ^super new initialize
Example - Instance Creation with Parameters

Smalltalk.CS535 defineClass: #Counter
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'count '
  classInstanceVariableNames: "
  imports: "
  category: 'Course-Examples'

Instance Methods

Category: initialize

setCount: anInteger
  count := anInteger

Category: access

count
  ^count

decrease
  count := count - 1

increase
  count := count + 1
Example - Instance Creation with Parameters Continued

Class Methods

Category: instance creation

new
  ^self count: 0

count: anInteger
  ^super new setCount: anInteger

Category: examples

example
  "Counter example"  "or one can use self example"

<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
</table>
a := Counter new.
a
  increase;
  increase.
  ^a count
Class Methods that Create Instances
Some Guidelines\(^1\)

Smalltalk does not have constructors like C++/Java

Use class methods to create instances

Place these class methods in "instance creation" category

### Initial State of Instances

Create objects in some well-formed state

Class creation methods should:

- Have parameters for initial values of instance variables or
- Set default values for instance variables

Provide an instance method that:

- Sets the initial values of instance variables
- Place method in "initialize" or "initialize - release" category
- Use the name `setVariable1: value variable2: …`

\(^1\) See Beck 1997, Constructor Method and Constructor Parameter Method patterns, pp. 21-24 and Johnson’s class notes on Smalltalk Coding Standards
Beck's First Rule of Good Style

"In a program written with good style, everything is said once and only once"

Some violations of the rule:

• Methods with the same logic
• Classes with the same methods
• Systems with similar classes

Example

```ruby
new
  ^self count: 0
count: anInteger
  ^super new setCount: anInteger
```

Not

```ruby
new
  ^super new setCount: 0
count: anInteger
  ^super new setCount: anInteger
```

If the logic of creating a new instance changes, the first version only has one place to change.

---

2 See Beck 1997, page 6
Providing Examples in Class Methods

A common Smalltalk practice is to provide

• Class method(s) implementing example use of the class

• Comment in the method to execute the example

Place such example methods in "example(s)" category

Category: examples

example

"Counter example"

| a |

a := Counter new.
a
    increase;
    increase.
^a count