

CS 535 Object-Oriented Programming & Design
Fall Semester, 2011
Doc 6 Smalltalk Classes
Sept 7 2011

Copyright ©, All rights reserved. 2011 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

Reading

Smalltalk by Example, Alex Sharp,

Chapter 2 Methods

Chapter 8 Control Structures

Chapter 4 Variables

Chapter 5 Instance Creation

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

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

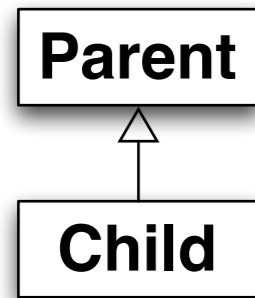
Subclass can implement methods with same name as parent

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

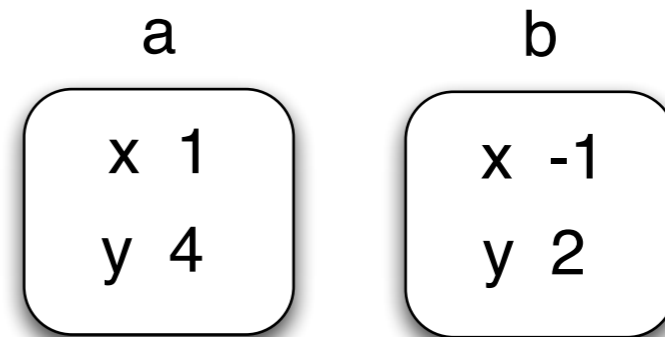
```
Smalltalk defineClass: #ClassPoint
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'x y '
  classInstanceVariableNames: "
  imports: "
  category: "
```

```
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.
```



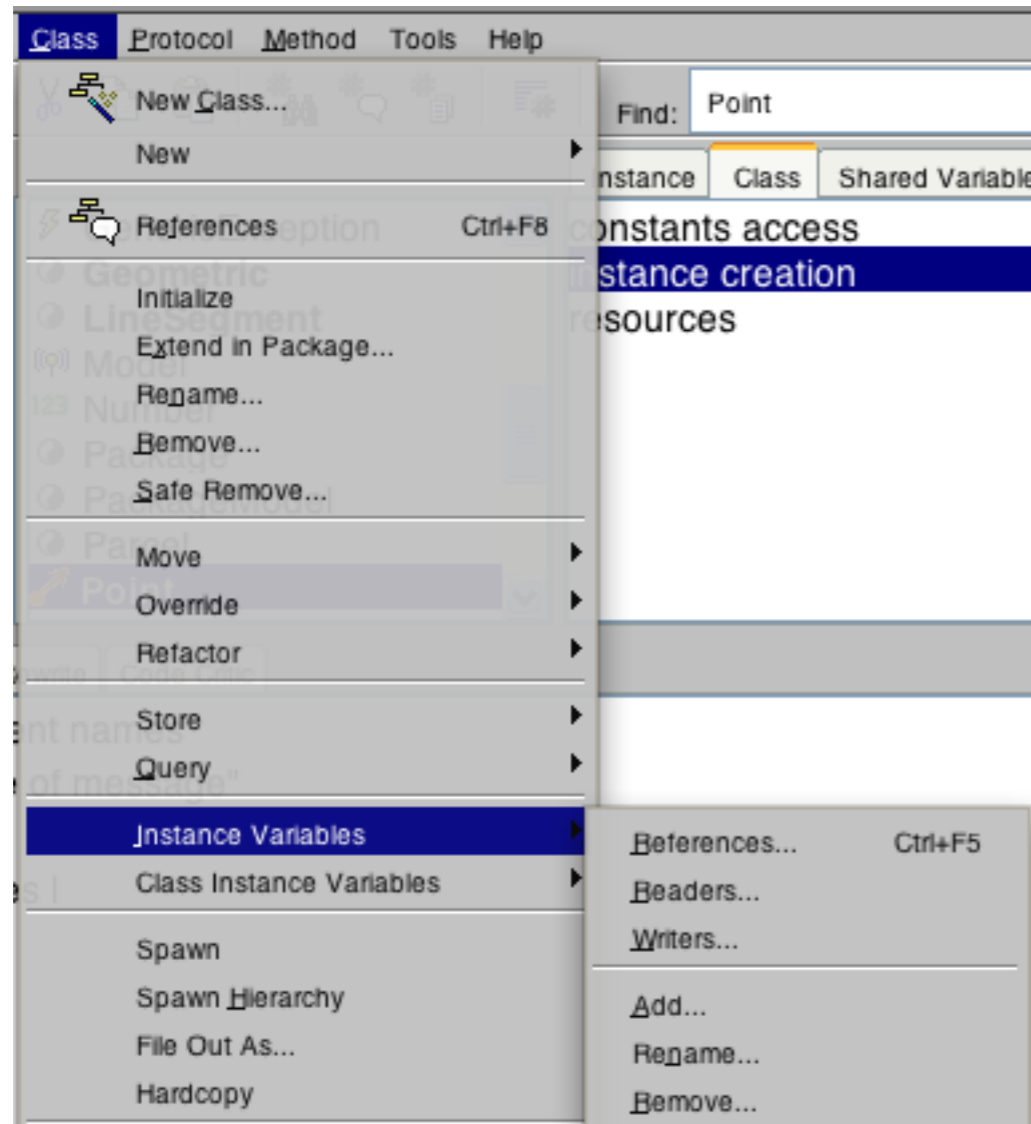
Adding Removing Instance Variables

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



self & super

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

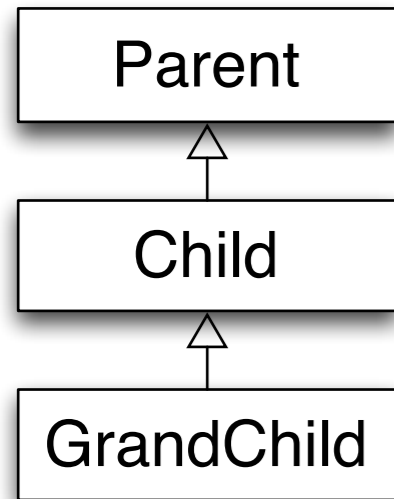
super

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

self and super Example



```
Parent>>name
^'Parent'
```

```
Child>>name
^'Child'
```

```
Child>>selfName
^self name
```

```
Child>>superName
^super name
```

```
GrandChild>>name
^'GrandChild'
```

Code	Output
grandchild	
grandchild := Grandchild new.	
Transcript	
show: grandchild name;	Grandchild
cr;	
show: grandchild selfName;	Grandchild
cr;	
show: grandchild superName;	Parent
cr;	

How does this work

grandchild selfName

Receiver is grandchild object

Code in selfName method is ^self name

To find the method "self name" start search in Grandchild class

grandchild superName

Receiver is grandchild object

Code in superName method is ^super name

superName is implemented in Child class

To find the method "super name" start search in the superclass of Child

Why Super

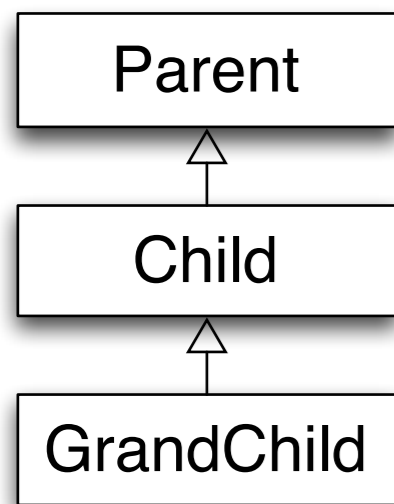
Super is used when:

The child class overrides a method
Needs to call overridden method

Common Pattern

```
ClassPointSubclass>>initialize  
  super initialize.  
  z := 0.
```

Why doesn't super refer to parent class of the receiver?



```
Parent>>name  
^'Parent'
```

```
Child>>name  
^super name , 'Child'
```

trouble
trouble := Grandchild new.
Transcript
show: grandchild name;

Class Methods

```
ClassPoint class>>origin  
  ^self x: 0 y: 0
```

```
ClassPoint class>>x: xNumber y: yNumber  
  ^(self new)  
    x: xNumber;  
    y: yNumber;  
    yourself
```

```
ClassPoint class>>new  
  ^super new initialize
```

```
center := ClassPoint origin.  
center x  
"Returns o"
```

new & initialize

ClassPoint>>initialize

x := 0.

y := 0.

ClassPoint class>>new

^super new initialize

ClassPoint new



SomeParentClass new initialize



aClassPointObject initialize

SomeParentClass new returns a ClassPoint object

Initialization and Inheritance

```
Smalltalk.Core defineClass: #Parent  
  superclass: #{Core.Object}  
  instanceVariableNames: 'foo '
```

Class Method

```
new  
  ^super new initialize
```

Instance Methods

```
initialize  
  foo :=6.
```

```
foo  
  ^foo
```

Initialization of Subclass

How to initialize bar?

```
Smalltalk.Core defineClass: #Child  
  superclass: #{Core.Parent}  
  instanceVariableNames: 'bar '
```

Bad Idea 1 – Use Same pattern

```
Child class>>new  
  ^super new initialize
```

```
Child>>initialize  
  bar := 2.
```

```
Child>>bar  
  ^bar
```

Why bad?

Does not work!

```
| test |  
test := Child new.  
test foo "returns nil"
```

initialize is called twice

Child class>>new is not needed
Child class inherits an identical method

Bad Idea 2 – Subclass initializes Parent Variable

Child>>initialize

bar := 2.

foo := 6.

Why Bad?

Child class now involved in private affairs of the Parent

Changes to the Parent instance variables require changing Child

Solution

```
Parent class>>new  
  ^super new initialize
```

```
Parent>>initialize  
  foo :=6.
```

```
Parent>>foo  
  ^foo
```

```
Child>>initialize  
  super initialize  
  bar := 2.
```

```
Child>>bar  
  ^bar
```

Class Methods that Create Instances

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: ...

Disabling new

Point new

Does not work

Point x: 1 y: 12

This works

Point class>>new

^self shouldNotImplement

Implementers wanted users to specify initial value of a point

Class Instance Variables

A class has one instance of a class instance variable

Each subclass has a different instance

Accessible by

- Class methods of the class

- Class methods of subclasses

Example

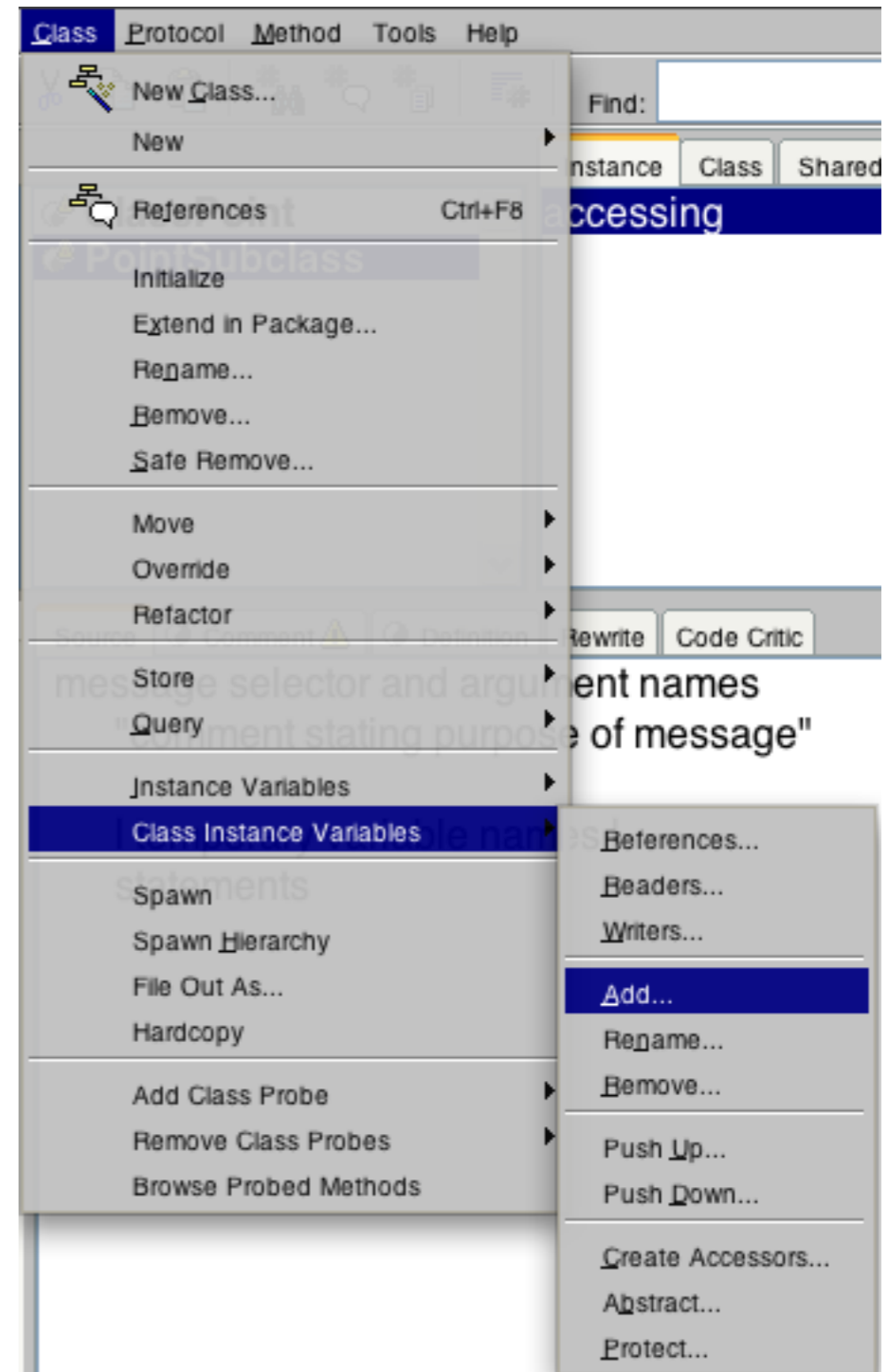
```
Smalltalk.Core defineClass: #ClassInstanceVariableExample
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: 'test '
  imports: "
  category: 'As yet unclassified'
```

Adding/Removing Class Instance Variables

Method 1

Edit the class definition directly

Method 2



Example

```
Smalltalk.Core defineClass: #Parent  
  superclass: #{Core.Object}  
  classInstanceVariableNames: 'test '
```

```
Parent class>>test
```

```
test isNil ifTrue:[ test := 0].  
test := test + 1.  
^test
```

```
Smalltalk.Core defineClass: #Child  
  superclass: #{Core.Parent}  
  classInstanceVariableNames: "
```

Transcript	
print: Parent test;	1
cr;	
print: Parent test;	2
cr;	
print: Child test;	1
flush	

Lazy Initialization

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

More on Blocks

Integer>>foo

| x block |

x := 10.

block := [self + x].

^block

In workspace

| x fooBlock result |

x := 5.

fooBlock := 3 foo.

result := fooBlock value

what is the value in result?

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