CS 683 Emerging Technologies: Embracing Change
Spring Semester, 2001
Doc 6 Classes
Contents

References

CS 497 Object-Oriented Programming & Design, Lecture notes, Ralph Johnson, Department of Computer Science, UIUC, http://st-www.cs.uiuc.edu/users/cs497/lectures.html

Smalltalk Best Practice Patterns, Kent Beck, Prentice Hall, 1997

Reading

Squeak: Object-Oriented Design with Multimedia Applications, Guzdial, Chapter 3

Classes

Object are instances of a class

A Class defines the behavior of its objects

A Smalltalk class has
- Methods
- Variables

Smalltalk supports only single inheritance

Types of Methods

Instance methods
- Sent to instances of Classes

Class methods
- Sent to Classes
- Similar to static methods in Java/C++

All methods are public

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

All methods return a value

Types of Variables

Named Instance Variable
- Like protected C++ data member (protected Java field)
- Accessible by
  - Instances (objects) of the class
  - Instances of subclasses

Class Variable
- Like protected static C++ data member
- Accessible by
  - The class and subclasses
  - Instances of the class and instance of subclasses

Class Instance Variable
- No C++ equivalent
- Accessible only by the class

Pool Variable
- No C++ equivalent
- Accessible by class and instances
- Can be shared by other class
- A bit complex, and very rare

Indexed Instance Variable
- Used for Arrays
- Most programmers never have add these to their classes

Pseudo-Variables

self
Similar to "this" in C++/Java

super
Similar "super" in C++/Java

Called pseudo-variables because:

They do change value
You can not assign values to them

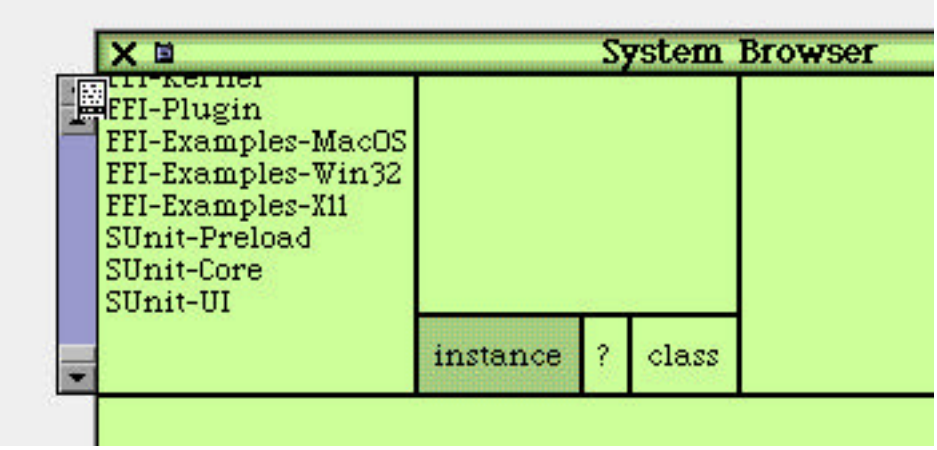
Creating Classes

One uses the Squeak browser to create new classes and modify existing classes. Squeak does not keep separate files for each class. The browser with the image file and the changes file handles storing the code. The browser acts like an IDE, compiler, source control management system and searchable database for Smalltalk code. After a while one actually gets so used to interacting with code via the browser that one stops printing code on paper.

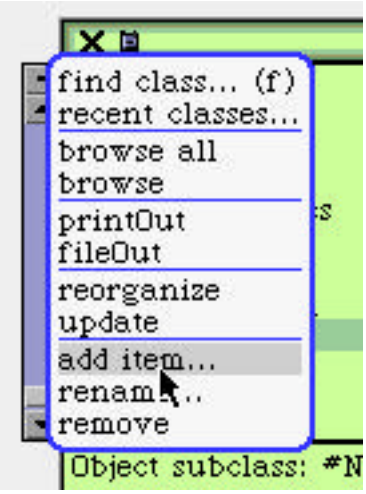
Normally one creates a new class category to keep your classes separate from the rest of the classes in Squeak. So first we need create a new category. Given the amount of code in Squeak, it is very useful to select meaningful names for your categories and classes.

Creating New Class Category

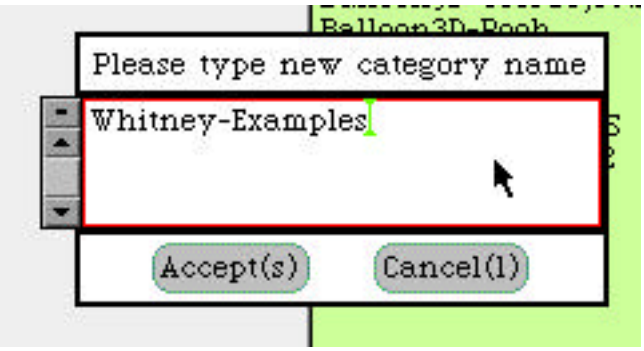
Get the context menu in the class category pane of the browser. You can do this as done below by clicking on the top of the scrollbar of the class category pane. You can also do this buy clicking the yellow button in the class category pane. The yellow button maps to option-click on a Mac, right-click on a 2-button PC mouse, middle-click on a 3-button PC or Unix mouse.



In the menu select "add item"



A dialog window will appear. Type in the name of the category in the dialog. You should follow the convention of using hyphenated names. The first part of the name is a major category, the second part the name being a subcategory. Don't worry about getting write the first time. You can change the name later if/when you think of a better name.



When you click on the Accept button the new category will be added to the class category pane. If an existing category was selected when you started, the new category is added above the selected category. Otherwise the category is added at the end of the pane. When you are done, in the bottom pane of the browser you will see a template for creating a new class like:

```
Object subclass: #NameOfClass
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Adding a Class

To create a class use the System browser to

- Create/Select the class category for the class
- Edit the class template in the bottom pane of the browser
- Accept (or save) the changes to the template

  Use the accept item in the context menu of the text pane or Alt-s on PC or command-s on Mac saves the changes

Here is the class definition for a class with
  Name Counter
  Parent class of Object
  One instance variable named count

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

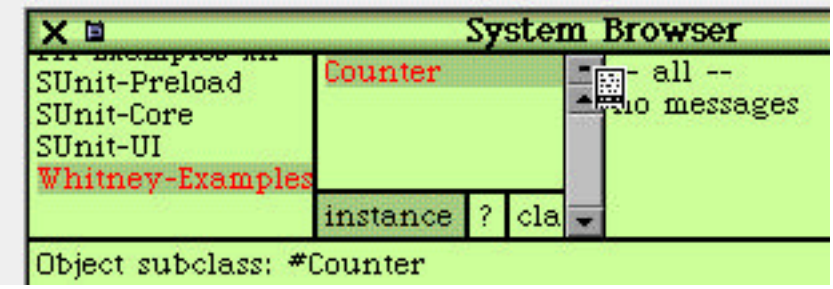
Here is a class definition with multiple variables

```
Object subclass: #BankAccount
    instanceVariableNames: ' name balance history id '
    classVariableNames: ' InterestRate'
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

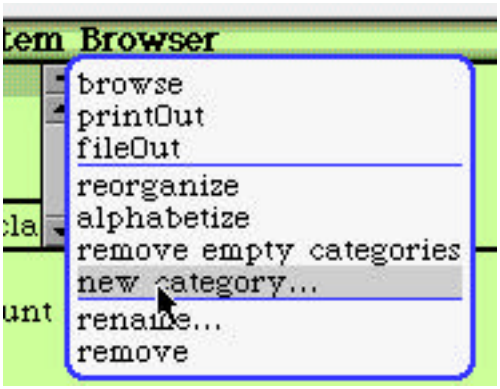
Adding a Method Category

Methods of classes are organized by categories. Like categories of classes the names of the method categories are important. The categories are here to help programmers handle and understand code. Pick meaningful names for categories. When possible use commonly used categories. Explore the system to find out the common categories.

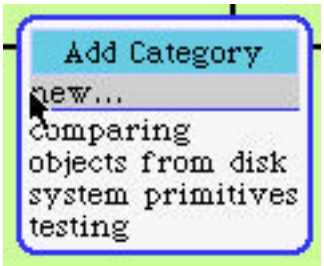
To add a new method category first get the context menu in the method category pane of the browser



In the menu select "new catetory…"

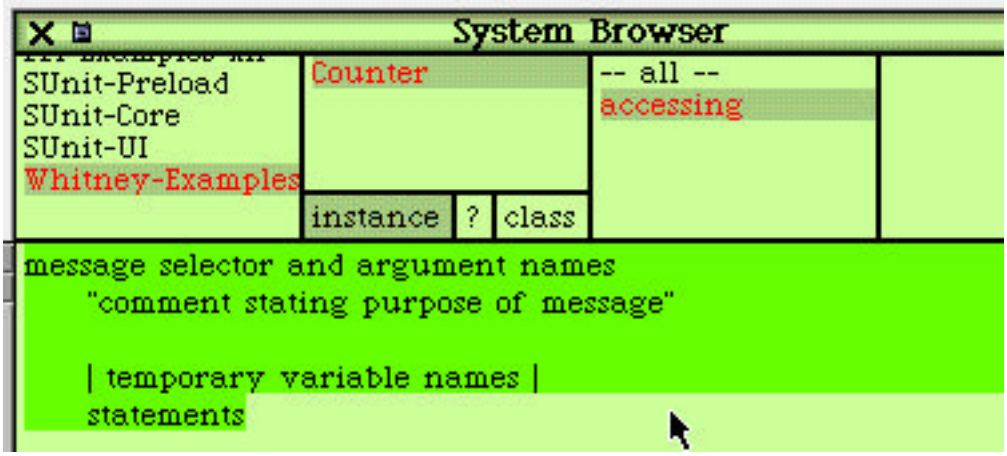


You get another menu. Select "new…" unless the name you wish to use is already listed.



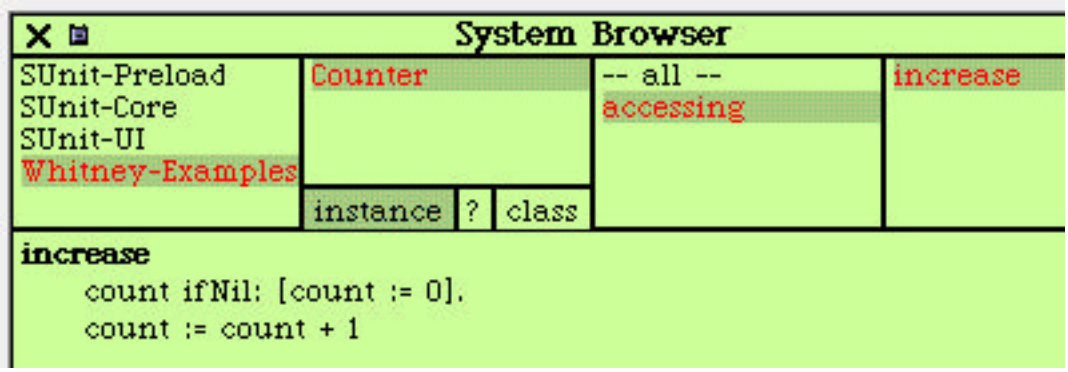
This gives you a dialog. Enter the name of your category.

After you accept the new category, it is added to the category pane. There will be a method template in the bottom pane of the browser.



Adding a Method

When you select a method category, the bottom pane of the browser shows the outline for a method. Replace the outline with the text for the method and then accept the changes.



Sample Class
Counter

Here is a simple class that increase or decrease a count

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Instance Methods

Category: **access**

```
count
    ↑count

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

increase
    count ifNil: [count := 0].
    count := count + 1
```

The above lists three different unary methods. The contents of the methods are all indented. So name of the method is left justified and indicates the start of a new method.

Sample use of the Counter Class

```
| test |
test := Counter new.
test
    increase;
    increase.
Transcript
    open;
    show: test count;
    cr;
    show: test
```

Output in Transcript

```
2
a Counter
```

Comment

new is the default message to send to a class to create an object. The method is defined in a parent class. It is not a constructor, just a class method.

Some Problems with Counter Class

- Instance variable count is not initialized

  Every time count used it is checked to see if it is nil

- Printing a Counter object does not display anything useful

We need the use of super to improve our Counter class. So we will cover inheritance, super and self.

Inheritance

A class can be extended or subclassed

The class that is extended is a superclass

Term	Equivalent Terms
superclass	parent class, base class
subclass	child class, derived class

The child class inherits the methods and variables of its parent class

Example

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''

| child |
child := Counter new.     "new - inherited class method"
child printString         "printString -inherited instance method
```

Inheritance and Name Clashes

Subclass can implement methods with same name as parent

Some people call this overloading the method

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

Subclass can not declare instance or Class variables with the same name as variables defined in superclasses

Self, Super
The Rules

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, Super Example

Three classes to study self, super

```
Object subclass: #Parent
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Instance Methods

```
name
    ↑'Parent'
```

Self, Super Example - Continued

```
Parent subclass: #Child
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Instance Methods

```
name
    ↑'Child'

returnSelf
    ↑self

returnSuper
    ↑super

selfName
    ↑self name

superName
    ↑super name


Child subclass: #Grandchild
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Instance Methods

```
name
    ↑'Grandchild'
```

Self, Super Example - Continued

Test Program

| grandchild |	Output In Transcript
grandchild := Grandchild new.	
Transcript	
show: grandchild name;	Grandchild
cr;	
show: grandchild selfName;	Grandchild
cr;	
show: grandchild superName;	Parent
cr;	
show: grandchild returnSelf;	a Grandchild
cr;	
show: grandchild returnSuper	a Grandchild

Self, Super Example - Continued
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 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.

Recursion

Smalltalk supports recursion

Here is the factorial method from the Integer class:

```
factorial
    "Answer the factorial of the receiver."

    self = 0 ifTrue: [↑ 1].
    self > 0 ifTrue: [↑ self * (self - 1) factorial].
    self error: 'Not valid for negative integers'
```

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

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-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
```

Comments

To add a class method to a class

- Click on the class pane/button in the browser and
- Follow the same steps for adding an instance method

Example - Instance Creation with Parameters

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-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"

	| a |
	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[2]

"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 same the same methods
- Systems with similar classes

Example

```
new
    ↑self count: 0

count: anInteger
    ↑super new setCount: anInteger
```

Not

```
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
```

Class Variables

An example showing the use of class variables

```
Object subclass: #BankAccount
    instanceVariableNames: 'name balance '
    classVariableNames: 'InterestRate '
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Instance Methods

Category: accessing

```
monthlyInterest
    ↑balance * (InterestRate / 12)
```

Category: initialize

```
setName: aString balance: aFloat
    name := aString.
    balance := aFloat
```

Class Methods

Category: initialize-release

```
initialize
    "BankAccount initialize"

    InterestRate := 0.065
```

Category: instance creation

```
name: aString balance: aFloat
    ↑self new
        setName: aString
        balance: aFloat
```

Class Variables - Comments

Class variables start with an uppercase letter

Both instance and class methods can access class variables

Initializing Class Variables

Use a class method called "initialize"

System calls this method when loading classes

Provide a comment that executes the method

printString

Similar to toString in Java

Standard method that returns text description of an object

Used by Transcript, print it and other tools to display objects

Object implements printString

- printString calls printOn: to produce the text description
- Implement printOn: in to replace default printString behavior

printString Example

```
Object subclass: #Counter
    instanceVariableNames: 'count '    "parts left out"
```

Instance Methods

Category: initialize

```
initialize
    count := 0
```

Category: **access**

```
count
    ↑count

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

increase
    count ifNil: [count := 0].
    count := count + 1
```

Category: printing

```
printOn: aStream
    aStream
        nextPutAll: 'Counter(';
        nextPutAll: count printString;
        nextPutAll: ')'
```

Class Methods

Category: instance creation

```
new
    ↑super new initialize
```

PrintString Example - Continued

The expression:

```
Counter new
```

When executed with "print it" displays:

```
Counter(0)
```

Implementing printOn:

The argument to printOn: is a WriteStream on a string

Important methods on a WriteStream

nextPut: aCharacter
  Add a character to the stream

nextPutAll: aString
  Add a string to the stream

cr
tab
crtab
space
  Add given whitespace character(s) to the stream

One must add only characters & strings to the argument of printOn:

Abstract Classes

Abstract class
Defines a common interface for subclasses

Instances are not allowed

Abstract method
Methods declared but not implemented in abstract class

Abstract methods in Smalltalk
Indicated by method body of "self subclassResponsibility"

Raise an exception if executed

```
Object subclass: #SampleAbstractClass
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'

methodDeclaredButNotImplemented: anArgument
    self subclassResponsibility
```

Interfaces

Squeak image does not support interfaces as it stands

SmallInterfaces

Class library for Squeak that supports interfaces

Useful in
- Indicating protocol required by parameters
- Design process

Available at:
http://wiki.cs.uiuc.edu/VisualWorks/SmallInterfaces

shouldNotImplement

Sometimes a class inherits an inappropriate method

Example

SortedCollection inherits at:put:

Elements in SortedCollection are in sorted order

Placing an element at a specified location is inappropriate

To indicate that a method should not be use in a class:

Place "self shouldNotImplement" in the method body

```
SomeOtherClass subclass: #SomeClass
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'

methodDeclaredInParentButDoesNotMakeSenseHere: anArgument
    self shouldNotImplement
```

Some Practical Matters
Printing Source Code

printOut
- Menu item in menu of system browser's top four panes
- Produces html file of the selected contents
- Use a Web browser to print the file

Dandelion
- Produces java doc like pages for Squeak code
- Rational Rose support
  - Converts Squeak classes to UML via Petal files
- Written in Squeak
- Available at:
  http://www.mars.dti.ne.jp/~umejava/smalltalk/stClasses/dandelion/index.html

Exchanging Source Code

Squeak stores all your source code in one file

How to exchange source code with other people?

Use either:
- Fileouts
- Change sets

Filing Source code Out & In

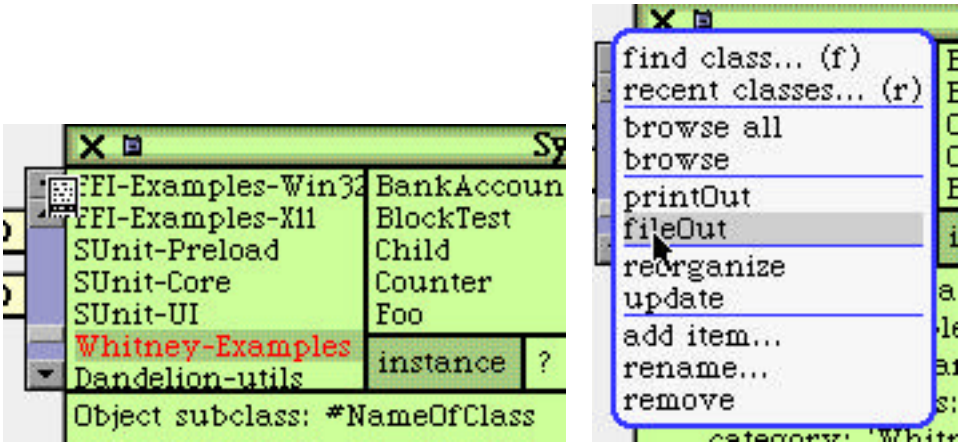
Source code can be filed out to a separate file

The file ends in .st

The file can be filed into another image

To File out Code

Select the fileout menu item in the context menu in any of the top four panes in the System Browser.

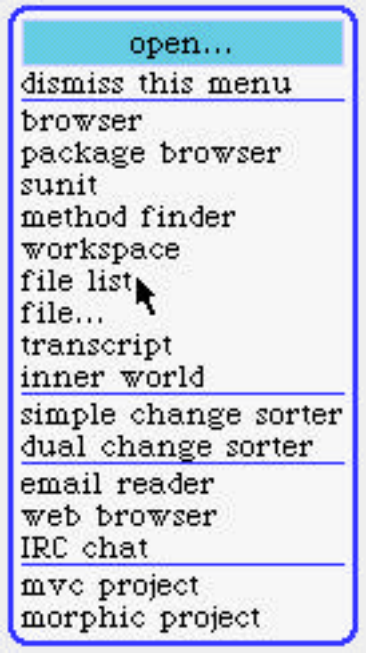


This will produce a file ending in .st in the same directory as image.

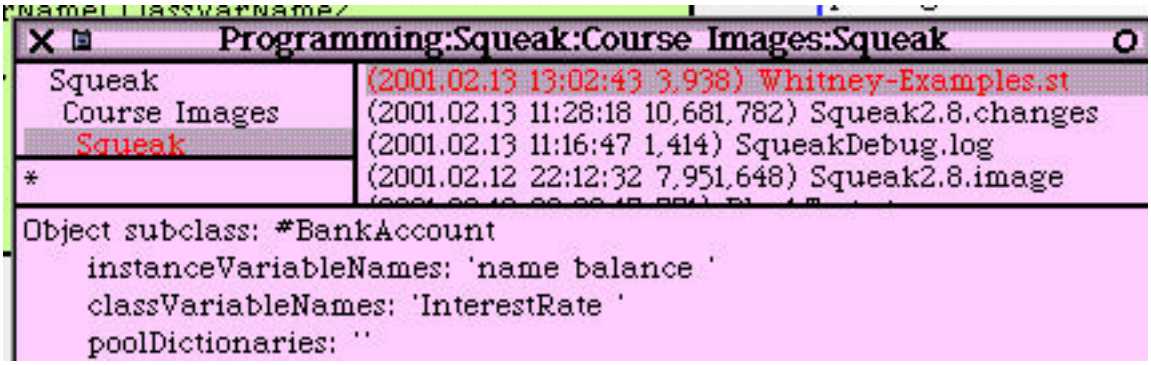
To File in Code

First open the Transcript window. Any errors that occur while filing in code are written to the Transcript. This happens only if the Transcript is open.

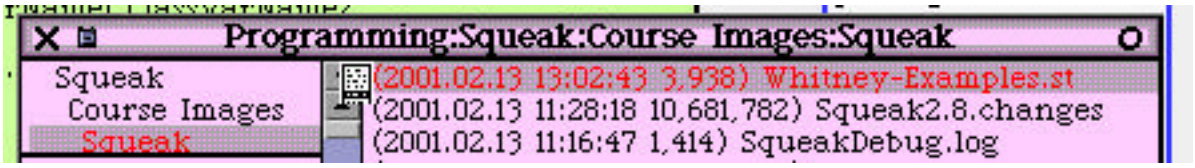
Open the file list tool. This is done by selecting the item "file list" in the open menu.



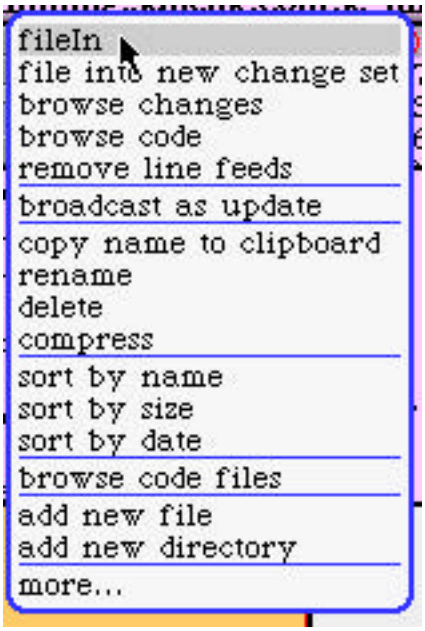
The upper left pane of the file list tool navigates among volumes and directories. The upper right pane lists contents of directories. Select a .st file in the upper right pane.



Get the context menu in the upper right pane



Select the menu item "fileIn" or "file into new change set".



Change Sets

Change set
- All the changes made while in a project
- Contains changes made in any class
- More useful than fileouts

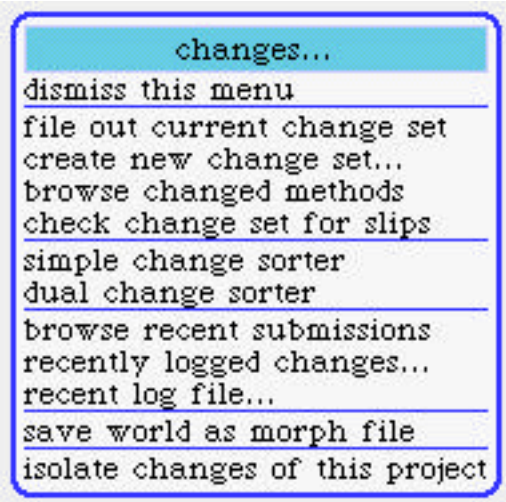
The file set initial has the same name as the project.

The default names for projects (unnamed) is not useful

Give your projects and change sets meaningful names

To File out a Change Set

Select the "changes…" item in the world menu. The changes menu looks like:



The "file out current change set" item does just that. It produces a file ending in .cs. The simple change sorter and the dual change sorter allow you to view and edit the change set.

To File in a Change Set

Follow the same directions for filing in a .st file

Classes Creation via Message

```
Object subclass: #Counter
    instanceVariableNames: 'count '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Whitney-Examples'
```

Give yourself bonus points if you noticed that the template to create a new class is actually a keyword message sent to the parent class. The Smalltalk class system is implemented in Smalltalk. As a result one can implement multiple inheritance, private methods, etc in Smalltalk. This is done by changing classes in the system, not by changing the compiler.

Exercises

1. Explore the existing code library to find 8 commonly used categories for methods.

2. Enter the Counter class into your system. Execute the following program:

```
| a |
a := Counter new.
a
        increment;
        increment.
a count
```

3. Which class defines the default version of the class method new?

4. Add a method **asGrade** to the Integer class. This method should return the following value:

Integer value	Value Returned
90 or higher	$A
80-89	$B
70-79	$C
60-69	$D
59 or less	$F

5. Add a method **gpa** to SequenceableCollection. When sent to a collection containing the characters $A, $B, $C, $D, $F the message gpa computes the grade point average on a 4.0 scale. In the 4.0 scale an A is worth 4 points, B is worth 3 points, etc. The method assumes that all the grades are for courses with the same number of units.

6. Create a change set containing the source code for problems 4 and 5.

7. Extend 5 to include the grades A-, B+, B-, C+, C-, D+, D-.