



# Getting Squeak

- <http://www.squeak.org>
- **Read instructions for working in labs linked from course web pages.**
  - Save your Squeak image in your home directory
  - To run Squeak directly in your network home directory, you will have to map a network drive (Squeak doesn't like Windows share paths.)
- Environment takes some getting used to: start as soon as you can!

# Squeak and mice

- Squeak needs 3 buttons; many Mac/Windows machines have fewer
- Name buttons by color

Windows mappings:

usual meaning:



Red left-click

“move/select”



Yellow right-click

“context menu”



Blue ALT + left-click

“window/Morphic menu”

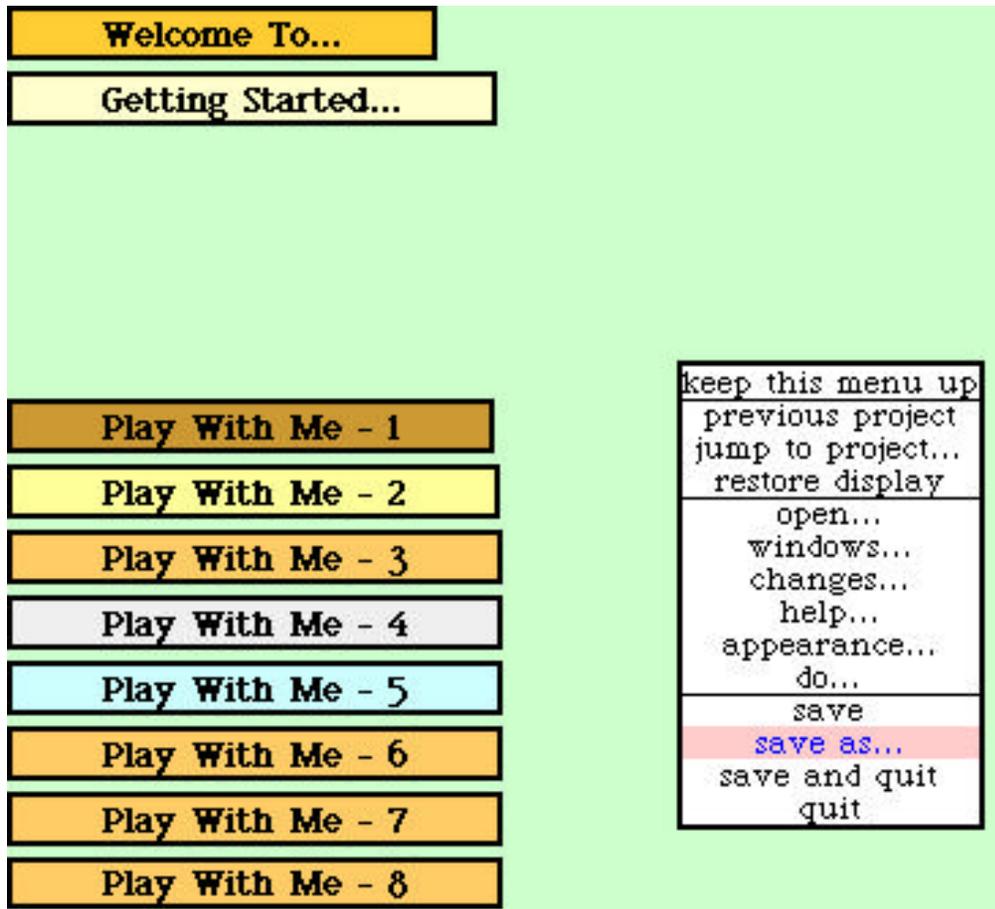
- See Guzdial p. 42 (or Squeak page linked from course web) for other platforms

# Squeak quick start: overview

Goal: to get you up & coding quickly.

1. Create your own image
2. Create your own project
3. Bring up a workspace, browser, transcript
4. Evaluate some code in workspace
5. Define a class in browser

# 1. Create your own image



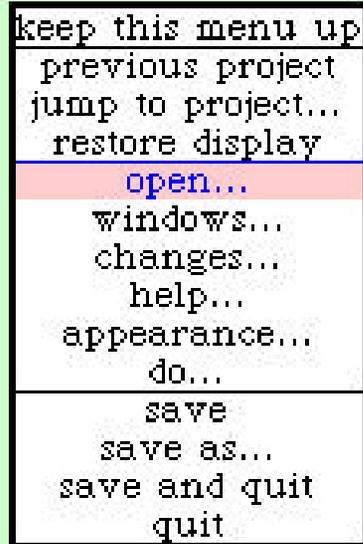
1. Red-click on desktop
2. Select “save as...”
3. Choose a filename.



# Why create your own image?

- Actually, you don't have to---Squeak allows you to undo any changes you make.
- But: I like having my own filename.
  - Keeps me from confusing it with other people's images, the standard image, etc.
- Delete old image to save space
  - (can always re-download it)
- Drag and drop your image onto Squeak executable to run it.
- Remember to keep both **.image** and **.changes**

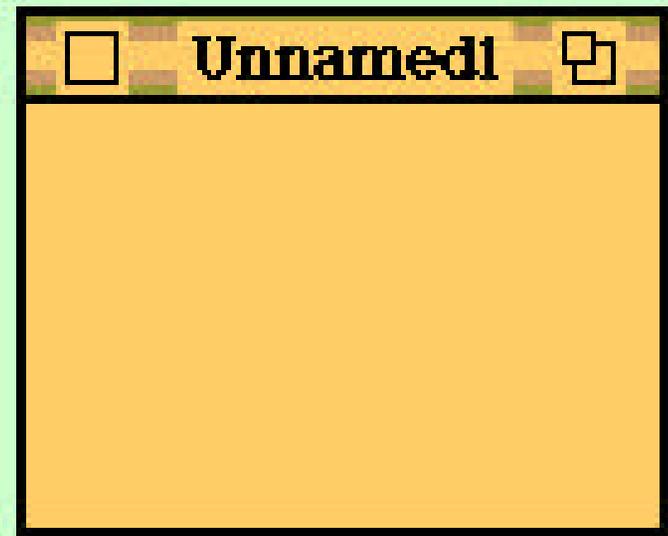
## 2: Create your own project



keep this menu up  
previous project  
jump to project...  
restore display  
**open...**  
windows...  
changes...  
help...  
appearance...  
do...  
save  
save as...  
save and quit  
quit



keep this menu up  
browser  
method finder  
workspace  
file list  
file...  
transcript  
morphic world  
simple change sorter  
dual change sorter  
mvc project  
**morphic project**  
construction project

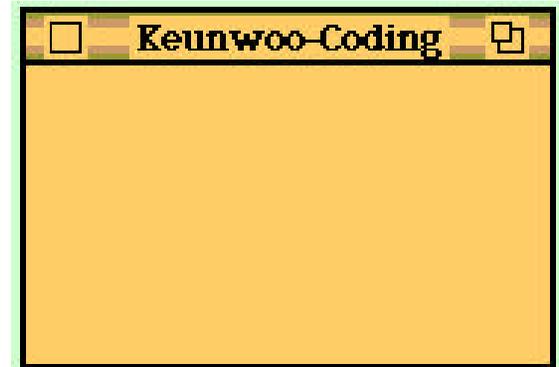
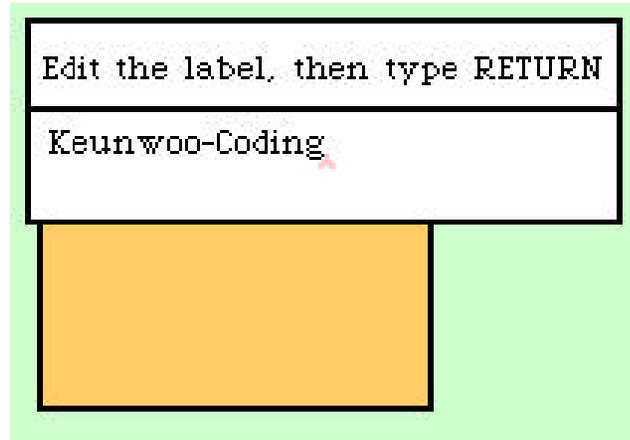


□ **Unnamed1** □

Choose “open” -> “morphic project”.

A little window will be created; this is an iconic view of your new project (initially empty)

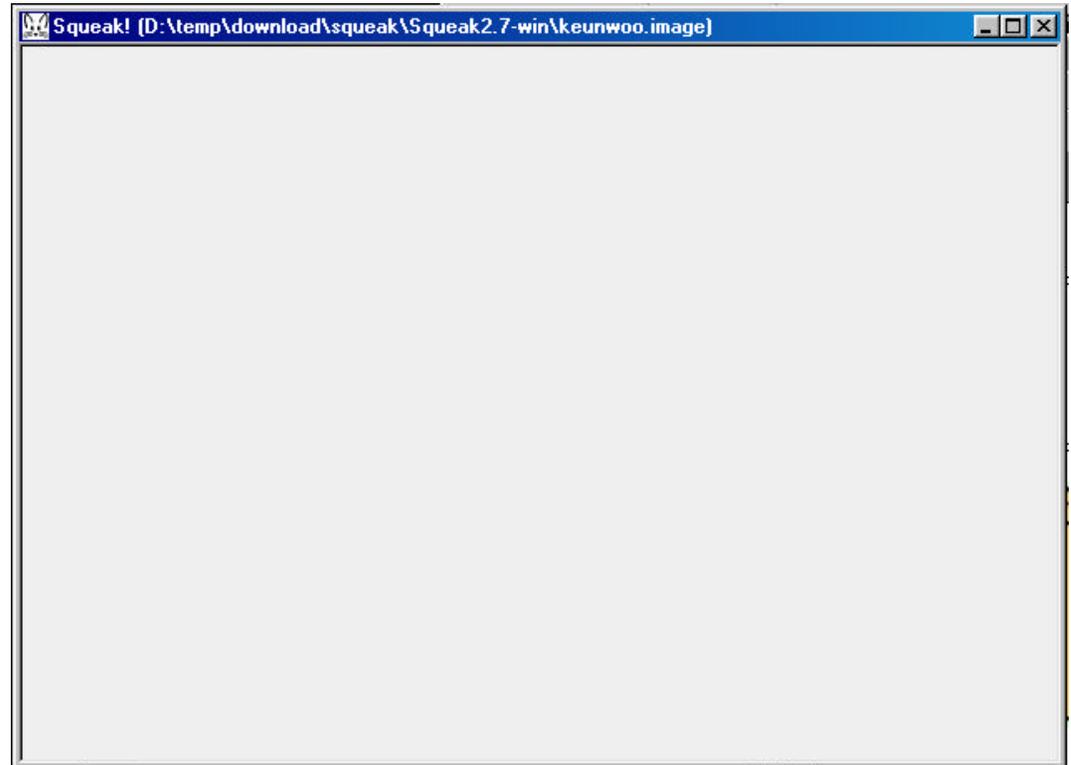
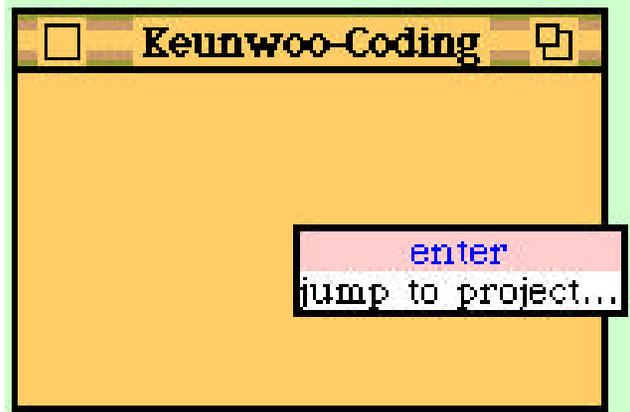
# Naming your project



To rename any window, click on its title bar and type in a new name.

# Entering your project

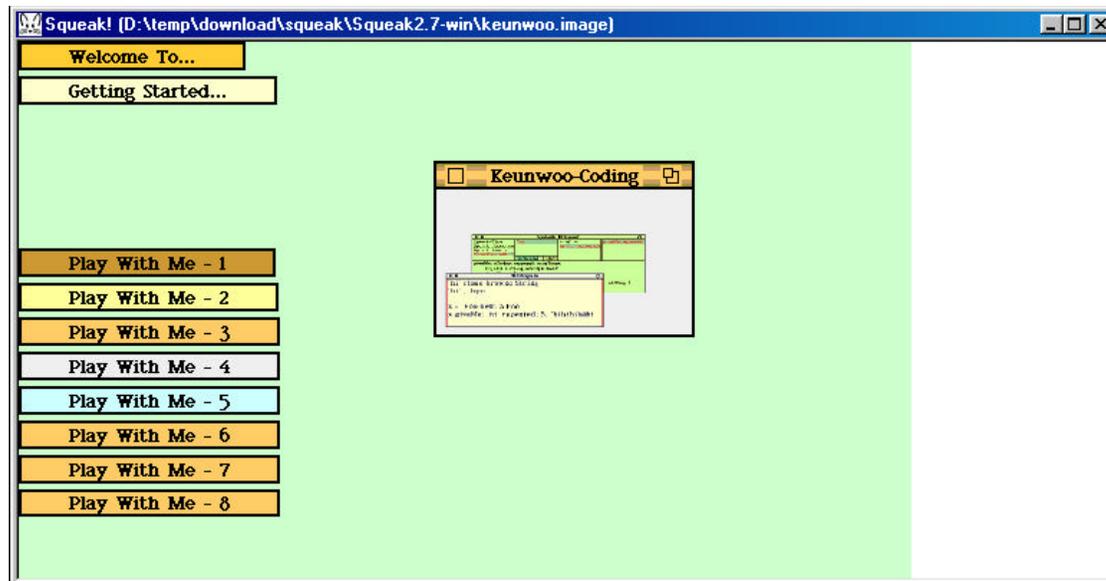
Red-click on your project icon and select “enter”.



You’ll get a blank desktop; this is your project.

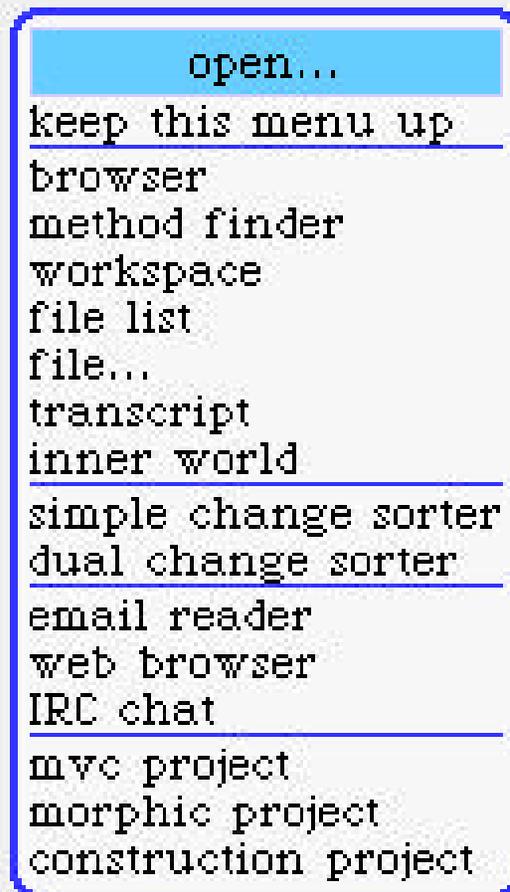
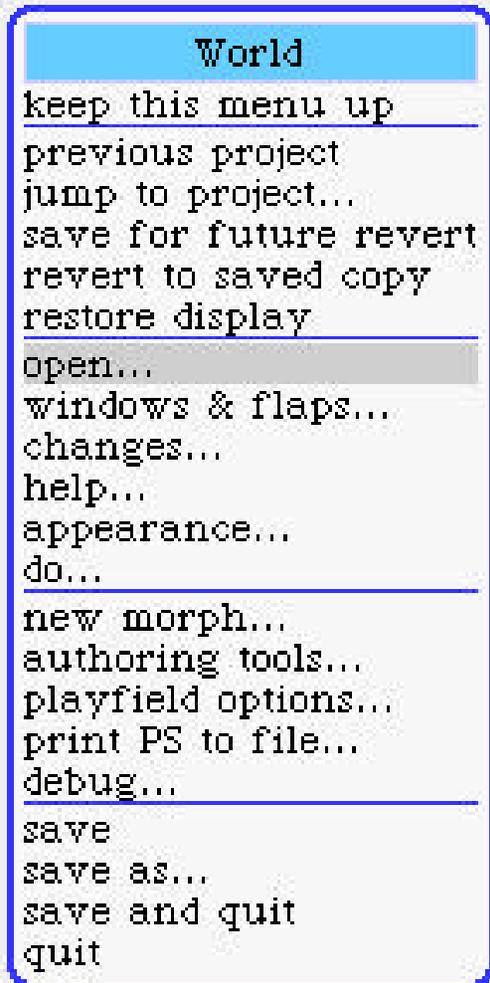
To leave your project, red-click on the desktop and select “previous project...”

# Resizing your Squeak desktop



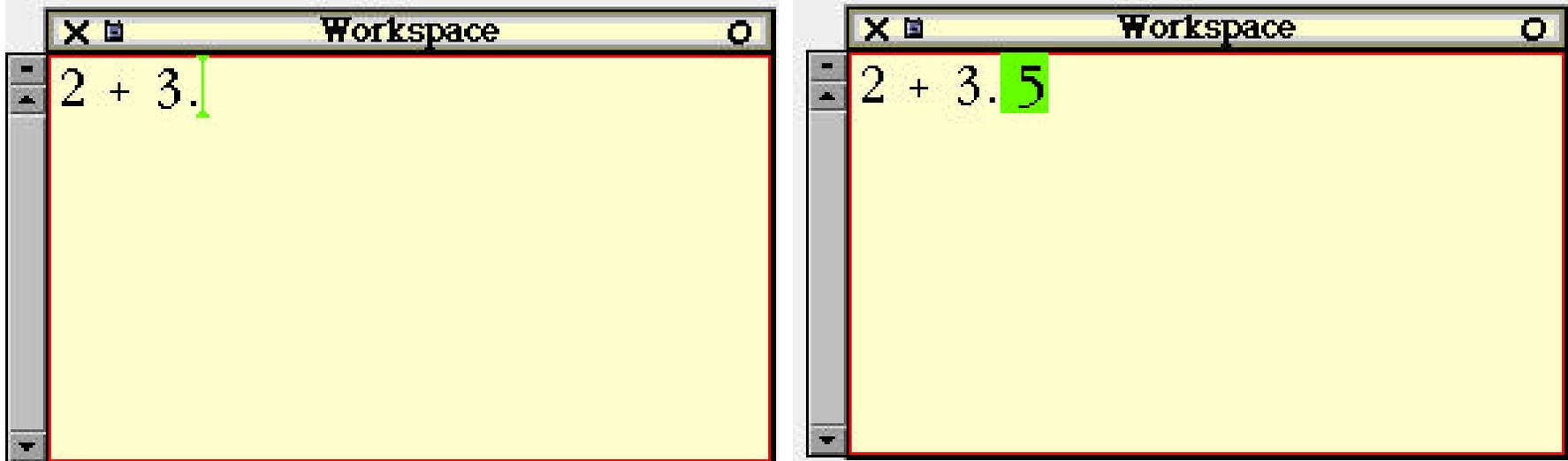
- Display can get messed up by some actions, e.g. window resize
- If you mess up your display, red-click on the desktop and select “restore display”

### 3. Bring up workspace, browser, and transcript



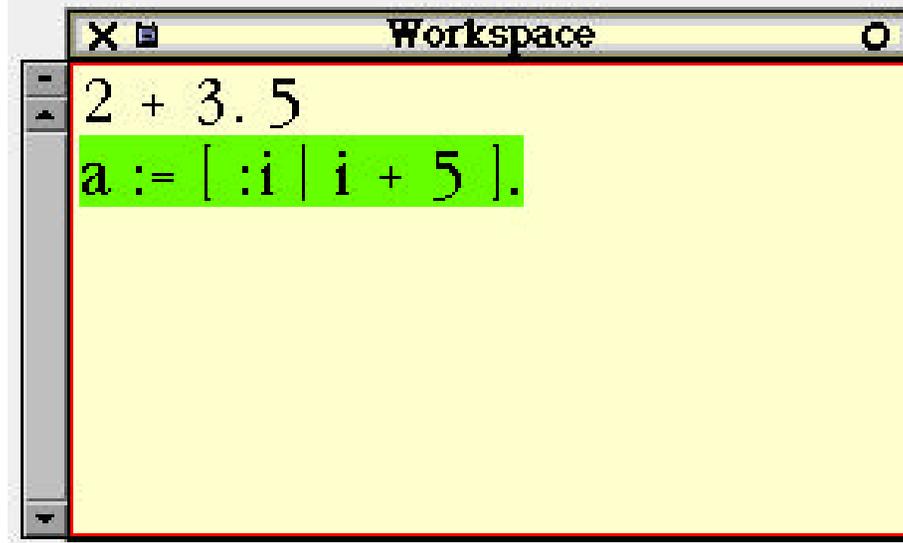
1. Red-click your project's desktop.
2. Select "open"
3. Select "workspace".
4. Repeat 2 and 3 for "browser" and "transcript".

## 4. Evaluate code in workspace



- Type in an arithmetic expression.
- Select it and press Alt-p (Command-p on Macs) to “printIt”
  - (Make sure you use lowercase p!)

# Executing code, ct'd

A screenshot of a software interface window titled "Workspace". The window has a yellow background and a red border. It contains two lines of code: "2 + 3.5" and "a := [ :i | i + 5 ].". The second line is highlighted in green. The window has standard window controls (close, maximize, zoom) in the top right corner and a vertical scrollbar on the left side.

```
2 + 3.5  
a := [ :i | i + 5 ].
```

To execute code without printing the result, select it and press Alt-d (Command-d on Macs).

You can run all the examples from today's handouts in the Workspace.

# 5. The system browser

The screenshot shows the 'System Browser' window with four main sections: 'class categories', 'classes', 'method categories', and 'methods'. The 'class categories' section lists various kernel and collection classes. The 'classes' section shows the superclass for the selected class, 'Boolean'. The 'method categories' section lists various method categories like 'logical operations', 'controlling', 'copying', 'printing', and 'misc'. The 'methods' section lists specific methods like '&', 'eqv:', 'not', and 'xor:'. The 'Code entry pane' at the bottom shows the implementation of the 'not' method.

class categories

classes

method categories

methods

Code entry pane

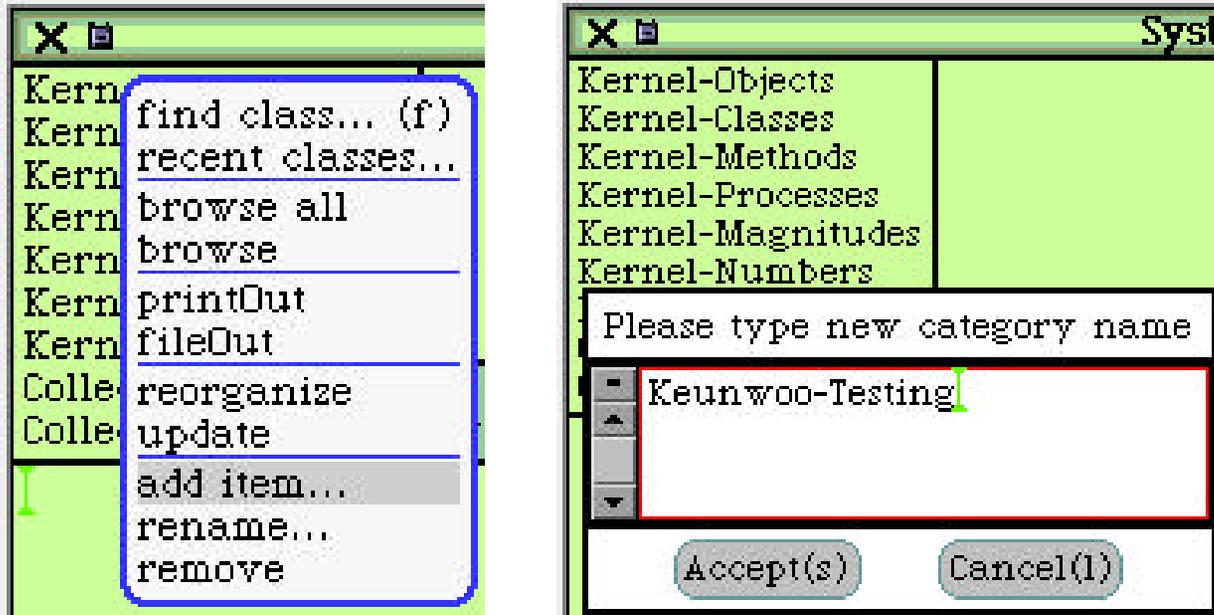
class categories	classes	method categories	methods
Kernel-Objects	Boolean	-- all --	&
Kernel-Classes	False	logical operations	eqv:
Kernel-Methods	Model	controlling	not
Kernel-Processes	MorphObjectOut	copying	xor:
Kernel-Magnitudes	Object	printing	
Kernel-Numbers	ObjectOut	misc	
Kernel-ST80 Remna	ObjectTracer		
Collections-Abstract	instance ? class		
Collections-Unorder			

```
not
"Negation. Answer true if the receiver is false, answer false if the
receiver is true."
self subclassResponsibility
```

# Creating a class: overview

- a. Create a class category
- b. Define a class
- c. Define methods for this class
- d. Try out the class by instantiating it and sending some messages

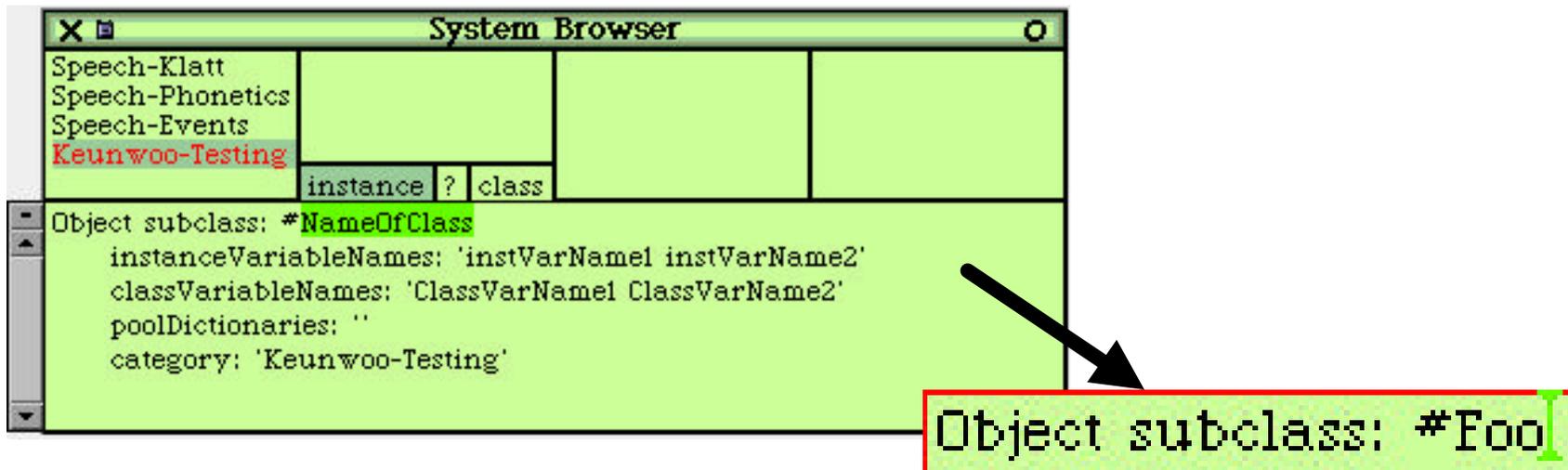
## (a) Creating a class category



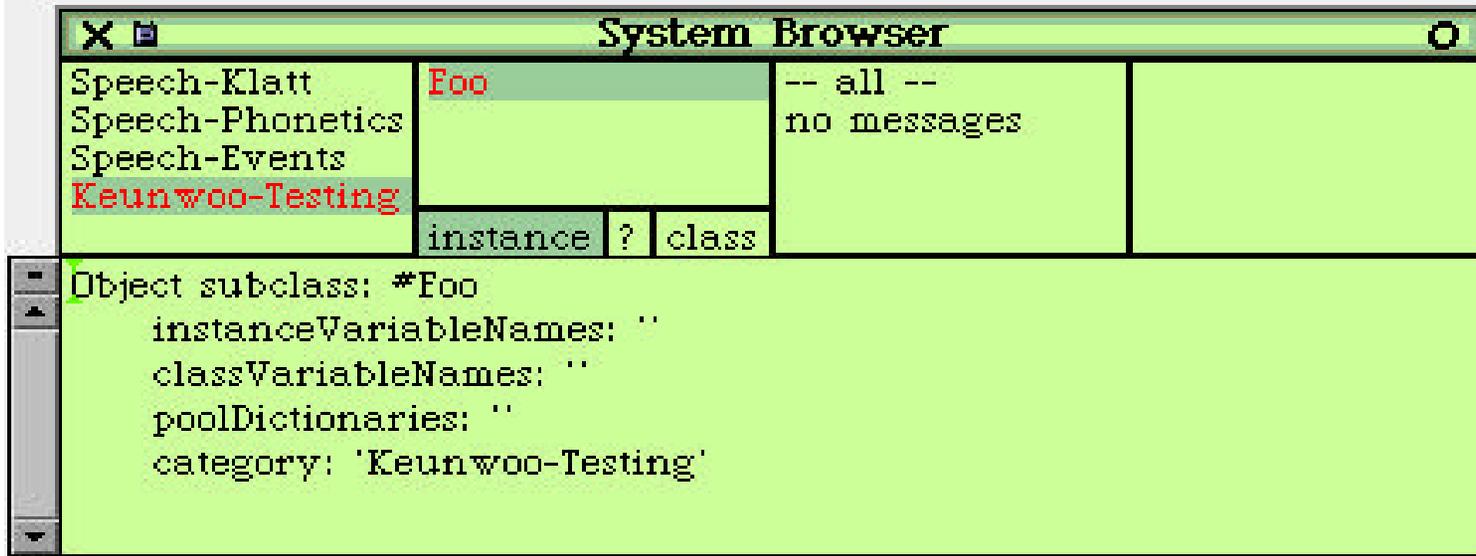
- Yellow-click on category pane
- Select “add item...”
- Type in a name and “accept”. Your new category will appear in the category pane.

## (b) Defining a class, part [1]

1. Red-click on your category.
2. An unfinished class declaration will appear in the code entry pane.
3. Change the “Object subclass #NameOfClass” line, replacing #NameOfClass with a class name, like “#Foo”.

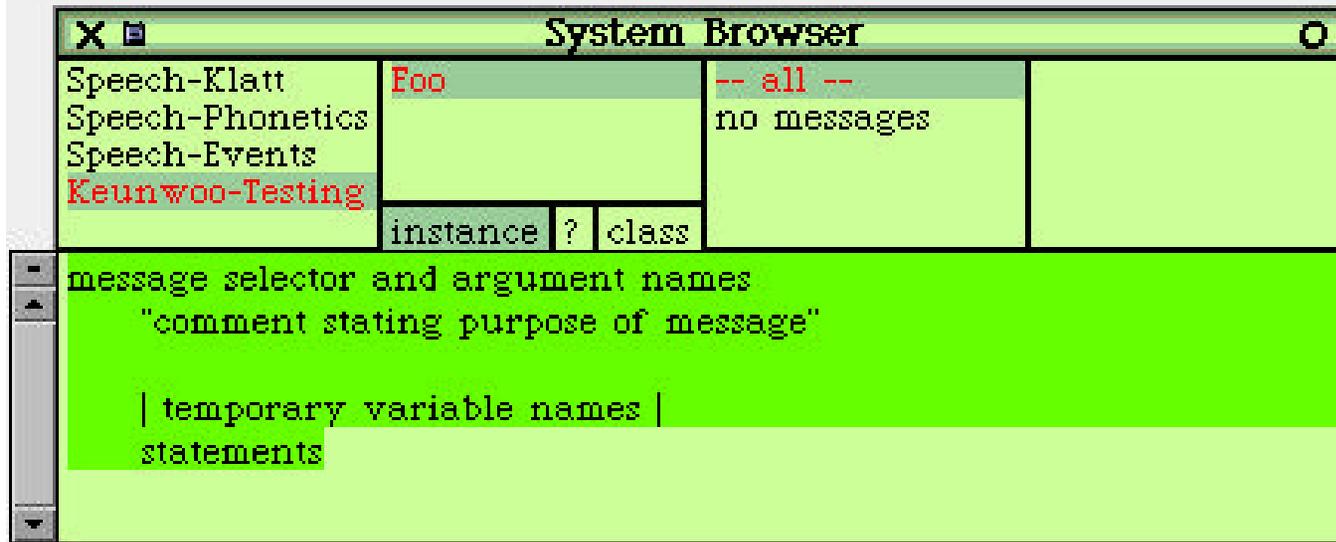


## (b) Defining a class, part [2]



4. Delete the class and instance variable names.
5. Use Alt-s (Command-s on Macs) to “Accept” your new class.
  - Your new class should appear in the class pane.

## (c) Defining methods [1]



1. Click on a category (e.g. “--all--”) in the method category pane.
  - An unfinished method declaration should appear in the code entry pane.
2. Enter some method text. For example...

## (c) Defining methods [2]

**giveMe: aString repeated: numTimes**

“Repeats a string multiple times”

| tempString |

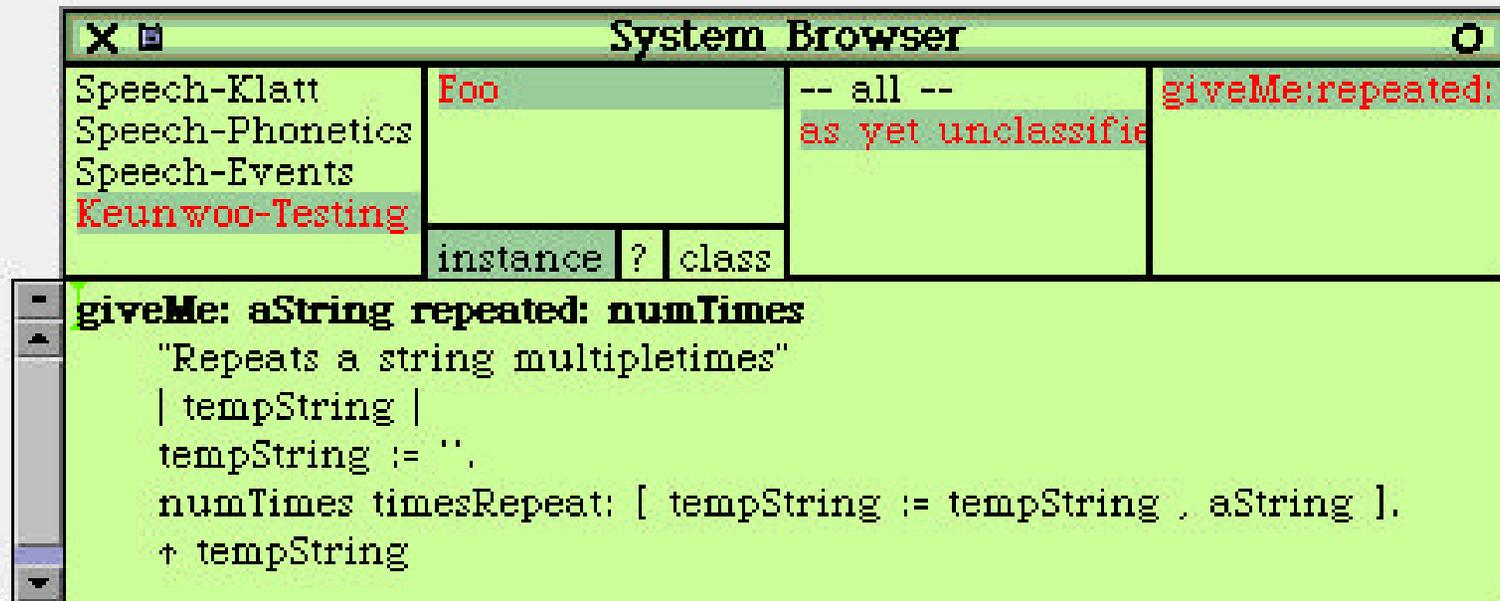
tempString := “”.

numTimes timesRepeat: [

tempString := tempString , aString ].

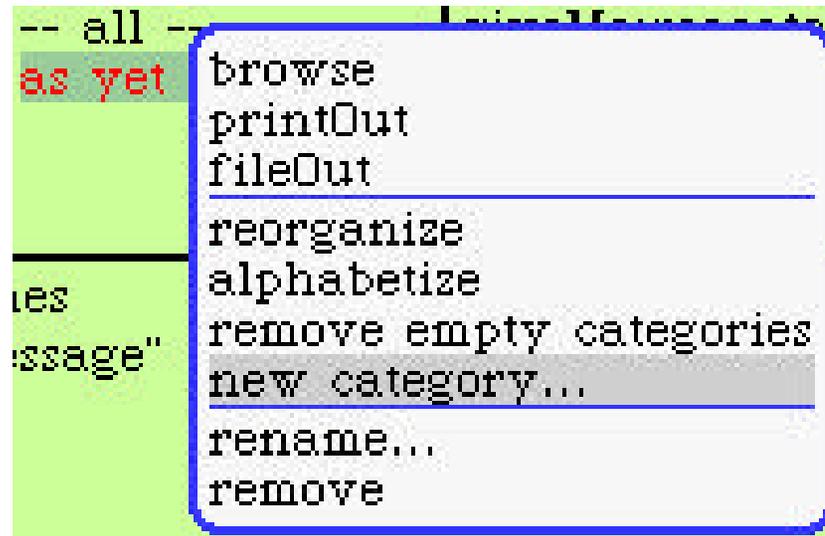
^ tempString

## (c) Defining methods [3]



- Pres Alt-s to “Accept”.
  - You may be asked for your initials.
  - Your method should appear in the method pane
  - A “not yet categorized” method category should be created.

## (c) Defining methods [4]



- To define a method category, yellow-click on the category pane and select “new category...”

## (c) Defining methods [5]

The image shows a screenshot of an IDE with a context menu open over a method named 'giveMe'. The menu is divided into two panes. The left pane contains options for browsing full details, file operations, senders/implementors, inheritance, versions, and variable references. The right pane contains options for browsing class hierarchy, inspecting instances, and changing categories. A third pane at the bottom shows the 'Choose Category' dialog with 'testing' selected.

giveMe

- browse full (b)
- fileOut
- printOut
- senders of... (n)
- implementors of... (m)
- method inheritance
- versions (v)
- inst var refs...
- inst var defs...
- class var refs...
- class variables
- class refs (N)
- remove
- more...

- browse class hierarchy
- browse class
- browse method (O)
- make stand-alone method pane
- toggle diffing
- implementors of sent messages
- change sets with this method
- inspect instances
- inspect subinstances
- remove from this browser
- change category...
- revert to previous version
- remove from current change set
- revert and forget
- fetch documentation
- more...

Choose Category (currently "as yet unclassified")

- testing
- as yet unclassified

- You can assign your method a proper category by red-clicking on the method name, yellow-clicking to bring up a menu, then selecting “more...” -> “change category...”.

## (d) Try out the class

```
x := Foo new. a Foo
```

```
x giveMe: 'hi' repeated: 5. 'hihihihihi'
```

- Go to your workspace or open a fresh one
- Type “x := YourClassname new.” (“Foo new” in this example).
- Select the text and doIt or printIt.
- Send a message to invoke the method you just defined. See what happens when you printIt.

# Conclusion: What next?

- Poke around in the System browser. Get to know the Squeak fundamental libraries.
- Read the Guzdial book. Learn about the drawing and Morphic libraries.
- Experiment with code fragments we give you in class. If you want to know what X does in Smalltalk, try it out!