A Tour of Squeak

Chapter 2

Squeak: Object-oriented design with multimedia applications
Story

- Six Basic Rules of Smalltalk
  - Sample code to walk through
- Doing "Normal" Things in Squeak
  - Variables and statements. Control structures.
  - Literals, numbers, operations, strings, and arrays
  - Files
- Doing "Object" Things in Squeak
  - Blocks, variables, classes, instances, and methods
  - How Squeak works
- Using Squeak
  - Projects
  - Example: Muppets in Squeak
- Finding More in Squeak
Six Basic Rules of Smalltalk

1. Everything is an object
2. All computation is triggered through message sends
3. Almost all of Smalltalk is `<receiverObject> <message>`
5. Every object is an instance of some class
6. All classes have a parent class, except for the root of the class hierarchy (was Object, now ProtoObject)
   - Parent is superclass, child is subclass
   - Subclasses inherit behavior and structure from parent class.
Sample Code

| anArray anIndex aValue | "Declare three local variables"
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue := 2.</td>
<td>&quot;Set aValue to 2&quot;</td>
</tr>
<tr>
<td>anArray := Array new: 10.</td>
<td>&quot;anArray is an Array 10 elems&quot;</td>
</tr>
<tr>
<td>1 to: 10 do:</td>
<td>&quot;Store 2*index at each array elem&quot;</td>
</tr>
<tr>
<td>[:index</td>
<td></td>
</tr>
<tr>
<td>anArray at: index</td>
<td></td>
</tr>
<tr>
<td>put: (aValue * index)]].</td>
<td></td>
</tr>
<tr>
<td>anIndex := 1.</td>
<td>&quot;Walk the array again, printing out the values&quot;</td>
</tr>
<tr>
<td>[anIndex &lt;= anArray size] whileTrue:</td>
<td></td>
</tr>
<tr>
<td>[Transcript show:</td>
<td></td>
</tr>
<tr>
<td>'Value at: ',(anIndex printString),</td>
<td></td>
</tr>
<tr>
<td>' is ', (anArray at: anIndex) printString ; cr.</td>
<td></td>
</tr>
</tbody>
</table>
|     anIndex := anIndex + 1.]}
Sample code output

Value at: 1 is 2
Value at: 2 is 4
Value at: 3 is 6
Value at: 4 is 8
Value at: 5 is 10
Value at: 6 is 12
Value at: 7 is 14
Value at: 8 is 16
Value at: 9 is 18
Value at: 10 is 20
Reviewing the Rules

1. Everything is an object
   aValue := 2
   Set the value of variable aValue to point to a SmallInteger object whose value is 2.
   aValue := 'fred'. Could follow immediately and would be perfectly fine
   All variables can point to any object.

2. All computation is triggered through message sends
   Even things like 1 to: 10 do: is just a message send
Reviewing the Rules

3. `<receiverObject> <message>

- 1 to: 10 do: [] is a message
  - 1 is the receiver
  - to:do: is the message
    - (Colons indicate an argument will follow)
  - 10 and the block [] are the arguments

- 2 + 3 is a message
  - 2 is the receiver
  - + is the message
  - 3 is an argument
Reviewing the Rules

4. Messages trigger methods
   - to:do: whileTrue: and + are all messages that have corresponding methods
   - There can be more than one method implementing the same message
     * Which method gets executed is based on the class of the receiver, and the decision is made at runtime
     * This is late-binding
   - More than one kind of object can respond to the same message in its own way
     * 3 + 5 and 3.1 + 5.2 are very different methods
     * This is polymorphism
   - Polymorphism lets you program in terms of goals not code
   - Methods in class Object are accessible by every object by inheritance, like printString.
Reviewing the Rules

5. Every object is an instance of some class
   - Class determines behavior
   - Every object responds to the message class with its class

6. All classes have parents
   - SmallIntegers have a method for *, but not the test method positive.
   - But Number knows positive, and SmallIntegers inherit from Number, so "2 positive" works and returns true.
SmallIntegers' lineage

Magnitude
  Character
  Date
  Number
    Float
    Fraction
    Integer
      SmallInteger
  Time
Starting to Squeak

- Download from http://www.squeak.org
- Launch by dragging the image onto the VM
  - squeak squeak.image on UNIX
- To do most things, open the Desktop or World menu
Opening a Workspace

- Choose Open…
- Open a Workspace
- PrintIt to print results of selection or current line (Alt-P/Command-P)
- HINT: DO THIS STUFF!
"Normal Things": Variables

- Variables: Any word starting with a letter
  - Start with lowercase, unless meant to be global

- Assignment
  - `:=` or `←` (underscore)

- Any expression is a statement
  - Periods separate statements

- Declare | local variables| in vertical bars
  - Or use "workspace scope"

```
a := 5.
a ← 5. "underscore"
a := 'String'.
a ← #(12 34 15).
```
Some notation comments

- "Double quotes delimit comments"
- 'Single quotes delimit strings' (collections of characters)
- $a$ is the single character lowercase a
- #(12 'a' $b) is a literal array of three elements
  - Smallinteger 12, String containing Character lowercase a, Character lowercase b
- #word is a Symbol
  - Specialized string optimized for lookup
  - Used for storing messages, Class names, etc. internally
Types

- There are none.
  - (Thought experiment: Do you need them? Why? How do you know?)
- Everything is just an object
  - Arrays and other collections can hold anything
- No distinction between public and private
- When types matter: When message is not understood.
  - 3 + 4 works—3 understands "+
  - 'fred' + 3 leads to…
Transcript: A place to print

- Open from Open… menu
- Transcript show: 'something'
- Transcript show: (3.1 + 5.2) printString
- Transcript cr "To get to the next line."
- Use ; to cascade messages
  - Transcript show: 'Hello world!'; cr.
Control statements

Examples

\[ a := 1. \]
\[ a < 12 \text{ ifTrue: } \text{[Transcript show: 'True!']} \]
\[ ((a < 12) \text{ and: } [b > 13]) \text{ ifTrue: } \text{[Transcript show: 'True!']} \text{ ifFalse: } \text{[Transcript show: 'False!']} \].
\[ [a < 10] \text{ whileTrue: } \]
\[ \quad [a := a + 1. \text{ Transcript show: (a printString), ' times'; cr.}] \]
\[ 1 \text{ to: } 9 \text{ do: [:i | Transcript show: (i printString),' times'; cr.}] \].
Control Statements

**Observations**

- At least one white space between tokens
  - Spaces can be duplicated (must be one as token separator)
  - Carriage returns are irrelevant
- Comma is string concatenation
- `[]` defines a block
  - Note: `()` just control evaluation
  - Blocks understand `whileTrue:` but not `ifTrue:`
Operations

- Binary operators
  - + - * // "quotient" \ "remainder" & | "No shortcut"

- Unary
  - abs sin positive asUppercase

- Keyword
  - anArray at: 5
  - to:do: max: and:

- Order of precedence (NOT ALGEBRAIC!)
  - Parenthesized terms first
  - Unary next
  - Binary next
  - Keyword selectors are last
Portion of Collections Hierarchy (Strings and Arrays)

Collection
  SequenceableCollection
    ArrayedCollection
      Array
      Array2D
    String
      Symbol
    Text
  OrderedCollection
    SortedCollection
  Set

9/19/00  Dictionary  Copyright 2000, Mark Guzdial
String examples

"A literal string"
'abc'

"Using with:with:with:"
String with: $a with: $b with: $c

"Creating a blank string then filling it."
aString := String new: 3.
aString at: 1 put: $a.
aString at: 2 put: $b.
aString at: 3 put: $c.
^aString "Returns the result"

'squeak' copyReplaceAll: 'ea' with: 'awwww'
    "Returns: 'squawwwwk'-- String sizes are immutable"
Array examples

"A literal array"

#(12 'b' $c)

"Using with:with:with:"

Array with: 12 with: 'b' with: $c

"Creating a blank array then start filling it."

anArray := Array new: 3.
anArray at: 1 put: 12.
anArray at: 2 put: 'b'.
anArray at: 3 put: $c.

^anArray
Collections operations

- Adding and removing elements
  - add: addAll: remove: removeAll:

- Testing
  - isEmpty includes: occurrencesOf:

- Enumerating
  - 'squeak' do: [:char | Transcript show: char printString].
  - 'squeak' select: [:letter | letter isVowel] "uea"
  - #(12 1 2) reject: [:num | num even].
  - #(12 1 2) select: [:num | num odd].
  - #(12 1 2) collect: [:num | num * 2]. "#(24 2 4)"
Streams

Streams support access or creation of a large data structure, one element at a time

Examples:ReadStream, WriteStream, PositionableStream, FileStream

All understand next, nextPut:, and nextPutAll:
FileStream examples

"Create a file"

aFile ← FileStream fileNamed: 'fred'.
aFile nextPutAll: 'This is a test.'.
aFile close.

"Read whole file at once"

aFile ← FileStream fileNamed: 'fred'.

^aFile contentsOfEntireFile

"Read character at a time"

aFile ← FileStream fileNamed: 'fred'.

[aFile atEnd] whileFalse:

[Transcript show: aFile next printString].
"Object" Things: Compiler and Blocks

- Compiler in Squeak is always available
  - Compiler evaluate: '3 + 4'

- Blocks: Code elements that are objects
  - aBlock := [Smalltalk beep].
  - aBlock value.

- Blocks can take arguments
  - anArgumentBlock := [:x | x + 1].
  - anArgumentBlock value: 5.

- Blocks are how all control structures are implemented in Squeak
**timesRepeat as example**

- 3 timesRepeat: [Smalltalk beep].
- Method for timesRepeat in Integer:

  ```smalltalk
  timesRepeat: aBlock
  "Evaluate the argument, aBlock, the number of times represented by the receiver."
  | count |
  count := 1.
  [count <= self]
  whileTrue:
    [aBlock value.
    count := count + 1]
  ```
Variables

- Variables point to objects
  - Uninitialized variables point to nil
  - Objects with no references are garbage collected
- You can have multiple variables point to the same object
  - `a := #(1 2 3).`
  - `b := a.`
  - `a at: 2 put: 75.`
  - `^b "Returns #(1 75 3)"
- Making copies: copy vs. deepCopy
- Checking for sameness
  - `=` is equality
  - `==` is equivalence
Creating classes

Object subclass: #NameOfClass
  instanceVariableNames: 'instVarName1 instVarName2'
  classVariableNames: 'ClassVarName1 ClassVarName2'
  poolDictionaries: "
  category: 'Kernel-Numbers'

Notes

- Smalltalk is case sensitive
- Classes start with uppercase
- # is significant: Classnames are symbols
- Wipe out the things you don't want, but leave the quotes
- Ignore poolDictionaries
Creating messages

Methods always have this format

messageForThisMethod

Smalltalk-statements-to-execute-for-this-message

Class can compile: messages

Person compile:

'greeting

Transcript show: "Hello world!"; cr.'

classified: 'Greeting'.
A note about classes

Classes are objects

- There are messages that classes understand that their instances do not
- For example, `new` is understood by classes, but not by instances

Classes are instances of classes

- Each class has its own `metaclass`
- All metaclasses are subclasses of the class `MetaClass`
Squeak model of execution

- Arguments are evaluated first
- Message and its arguments are sent to receiving object
  - Special variable `self` is bound to receiver
- Class for receiving object is checked for instance method
  - If so, execute that method
- If not, parent class is checked, and so on up to Object
  - If not found, a `doesNotUnderstand:` message is sent to original object (same lookup)
- If execution arrives at a primitive, execute the internal VM routine

(All very fast! 233Mhz G3: 15M bytecodes/sec; 792K sends/sec)
Using Squeak

- Four main files make up the release
  - Sources: In text, source for everything
  - Changes: In text, source for all new things
    - Note: You never lose any source!
  - Image: Binary, in virtual processor machine language
  - VM: Binary, your native processor program translator
- Other files: Readmes, plugins
- Different model of programming
  - Editor, compiler, debugger are all in Squeak
  - You don't create source files: You hack in Squeak
    - Source goes to changes file
    - Object goes to image file (when you save image)

**NOTE: ALWAYS WORK WITH A BACKUP COPY OF IMAGE AND CHANGES FILE!**
Starting out

- Save a copy of your image and changes file, and work with the new one
- Create a new project
  - MVC for slow machines (P75 or less), Morphic for faster ones
- Return with Previous Project from Desktop/World menu
- Open a Workspace to type in, a Transcript to show in
  - PrintIt to print results, Dolt to just do something (Alt-D, Command-D)
- Open a Browser to work in
# System Browser

## Class categories
- Numeric-Magnitudes
- Numeric-Numbers
- Collections-Abstract
- Collections-Unordered
- Collections-Sequenceable
- Collections-Text
- Collections-Arrayed
- Collections-Streams

## Method categories (or protocols)
- accessing
- arithmetic
- comparing
- inquiries
- converting
- printing
- private

## Methods
- `addDays:`
- `subtractDate:`
- `subtractDays:`

## Smalltalk code for method
```smalltalk
addDays: dayCount

"Answer a Date that is dayCount days after the receiver."

+Date newDay: day + dayCount
  year: year
```
Common errors in using the System Browser

- Almost *always* leave the Instance button selected
- Creating a class category does *not* create a class
- Categories are not relevant to the language, but are important for other users
- Use the provided templates for methods and class definitions
Mouse Buttons in Squeak

- "Red" is for pointing
- "Yellow" is for context-sensitive menus
- "Blue" is for window-specific menus

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Red Button</th>
<th>Yellow Button</th>
<th>Blue Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macintosh</td>
<td>Mouse click</td>
<td>Option-click</td>
<td>Command-click</td>
</tr>
<tr>
<td>Windows</td>
<td>Left-click</td>
<td>Right-click</td>
<td>Alt-left-click</td>
</tr>
<tr>
<td>UNIX</td>
<td>Left button</td>
<td>Middle button</td>
<td>Right button</td>
</tr>
</tbody>
</table>
Defining a Muppet

- Superclass is Object
  - Very common
- NameOfClass is Muppet
- No class variables
- Start out with name as only instance variable
- Accept (Alt-S/Command-S) to save/compile the class definition
- Already, some things will work
  - kermit := Muppet new.
  - Transcript show: kermit printString.
Defining methods for Muppets

**greeting**

"Return a pre-defined greeting"

^'Hello there!'

**greet**

"Have the Muppet greet us to the Transcript."

Transcript show: self greeting; cr.
Walking through a Muppet example

kermit := Muppet new.
kermit greet.

1. Muppet doesn't know new. It passes up to something that does, and returns a new Muppet.

2. \textit{kermit} is bound to the new instance of Muppet.

3. New instance is asked to \textit{greet}

4. Muppet's know \textit{greet}, so method asks \textit{self} for \textit{greeting}

5. \textit{greeting} returns 'Hello there!'

6. Transcript shows 'Hello there!'

7. Then \textit{cr} is generated
Saving your work in Smalltalk

- Save your image *frequently*
- FileOut your code
  - Category, class, protocol, method
  - Or even whole project
Finding more

- "There must be a class named X…"
  - Type X anywhere, select it, then Alt/Command-B
  - Find class from Browser class category pane
  - Find an instance of X and inspect it
Avoiding a bazillion inspectors

- World explore "In Morphic"
There's a method, 'at:' something...

Type a fragment of a selector in the top pane. Accept it.

Or, use an example to find a method in the system. Type receiver, args, and answer in the top pane with periods between the items. 3. 4. 7

Or, use examples to find a method in the system.
What is this method?

In Browser, menu item explain

If you know it's a method, implementors
Why doesn't this work?

- Transcript show: 34.2
Squeak's Debugger

Annotations pane (a preference)

Walkback of current stack

Currently executing code

Inspector on receiver

Inspector on context including arguments

Currently executing code

nextPutAll: aCollection

"Append the elements of aCollection to the sequence of objects accessible by the receiver. Answer aCollection."

aCollection do: [:v | self nextPut: v].

+aCollection

self all inst vars collection position readLimit writeLimit

collection

thisContext all temp vars aCollection

34.2
Other kinds of Browsers

- Hierarchy view
  - Shows superclasses and associated methods
- Protocol view
  - What this class understands
- In general, if you can see the code, you can change the code
  - Even in the debugger
Help

- Help menu in World/Desktop menu
- Balloon help in Morphic
- Tutorials from http://www.squeak.org
- http://minnow.cc.cc.gatech.edu/squeak/
  
  Of course, you could always read the book...:-)
How do you make an application?

- Image + VM - sources - changes
  - Can run Squeak in < 1Mb
  - One day: Merge image and VM

- Different model: Everything builds on everything
  - Goodies in the image

- Yet another model: Embedded computing