

# Smalltalk - Pure OOP

R. Mark Volkmann

Object Computing, Inc.



<https://objectcomputing.com>



[mark@objectcomputing.com](mailto:mark@objectcomputing.com)



[@mark\\_volkmann](https://twitter.com/mark_volkmann)

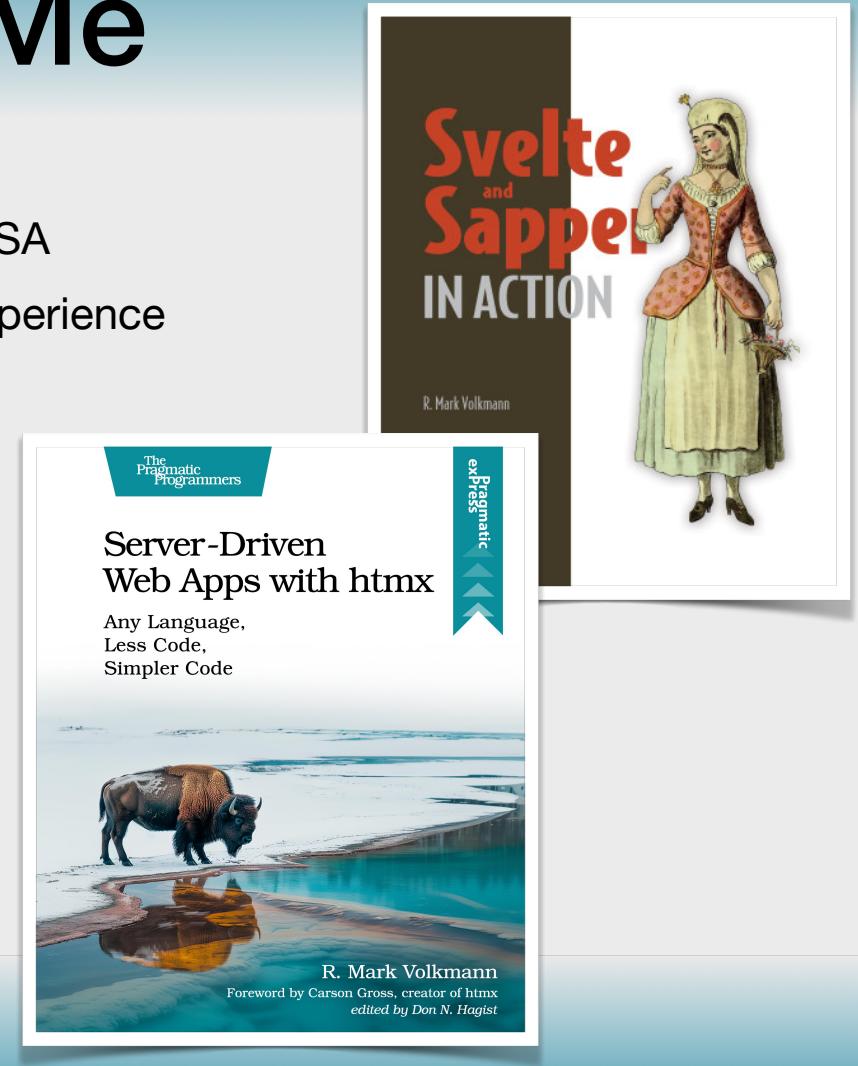


**OBJECT COMPUTING**  
YOUR OUTCOMES ENGINEERED

Slides at <https://github.com/mvolkmann/talks/>

# About Me

- Partner and Distinguished Software Engineer at Object Computing, Inc. in St. Louis, Missouri USA
- 44 years of professional software development experience
- Writer and speaker
- Blog at <https://mvolkmann.github.io/blog/>
- Author of Manning book “Svelte ... in Action”
- Author of Pragmatic Bookshelf book “Server-Driven Web Apps with htmx”



# Why Learn Smalltalk?

- Beautifully minimal syntax
- Excellent development environment
- Gain understanding of pros and cons compared to other languages
- Get ideas for features that can be added to other languages and their development environments
- Actually use it as an alternative to other languages

# Smalltalk Cons

- **Small community**, so it can be difficult to get answers to questions
- **Attitude** in community that because all the code is easily accessible, you can find the answer to any question on your own
- **Documentation** on some topics is lacking
  - how to package applications for use by non-developers
  - how to access databases
  - how to implement HTTP API endpoints (web servers)
  - how to deploy to cloud environments

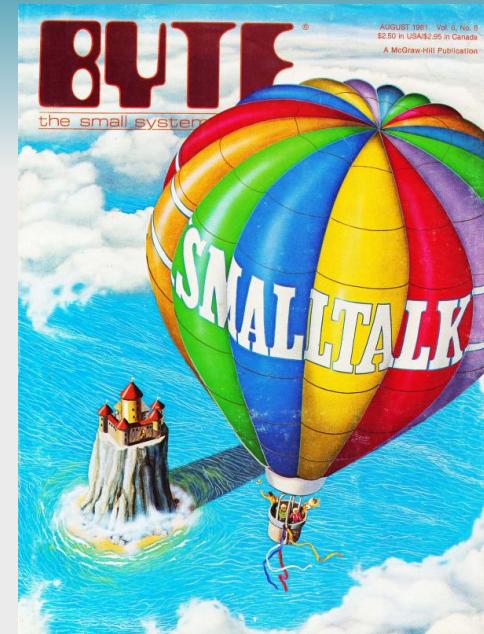
# Smalltalk Overview



- Purely object-oriented, dynamically typed programming language
  - first popular OOP
  - everything is represented by an object that is an instance of some class
  - includes classes themselves and all development environment GUI elements
- Everything happens by sending messages to objects
  - objects decide whether and how to act on messages
- Duck typing
  - types of variables, method parameters, and method return types are never specified
  - use any object as long as it responds to all messages sent to it
    - determined at run-time (late binding)

# Origin

- Invented at Xerox Palo Alto Research Center (PARC)
  - by Alan Kay, Dan Ingall's, Adele Goldberg, and others in 1970's
- First version in 1972
- Gained popularity in 1981 due to Byte magazine cover
- Was popular alternative to C++ in early 1990's
- But there were no free, open source implementations and licenses for commercials versions were expensive
- The free Java language was released in 1995 and the wind was removed from Smalltalk sails



# Current Implementations

- **Free, Open-Source**

- Squeak (1996) - fork of Smalltalk-80
- Pharo (2008) - fork of Squeak
- Cuis (2009) - fork of Squeak

Pharo and Cuis  
use Squeak VM

- **Commercial**

- Cincom Smalltalk - VisualWorks and ObjectStudio
- Instantiations VA Smalltalk - VAST Platform
- GemTalk Systems - GemStone/S
- Dolphin Smalltalk

**Notable ways in which Cuis differs from Squeak and Pharo:**

- ships with minimal set of Smalltalk-80-inspired classes for simplicity
- supports Unicode
- supports TrueType fonts
- supports high-quality vector graphics
- supports SVG

# Just-in-Time Compilation



- Not interpreted
- First programming language to use just-in-time (JIT) compilation
- Code is compiled to optimized bytecode that is executed by a virtual machine (VM)
- Compilation occurs when method code is saved
- Results in better performance than interpreting code

# VMs and Images



- Running Smalltalk programs requires two parts, VM and image file
  - VM reads and executes Smalltalk code found in image file
  - VM is specific to operating system and CPU architecture being used
- Image files are snapshots of current environment state
  - describes collection of all active objects
  - during development, changes can be saved to current image or a new image
- Image files can be moved between operating systems on different CPU architectures
  - displays same (pixel for pixel) across Windows, Linux, and MacOS, only differing based on screen size
  - no need to recompile code for different environments; makes code highly portable

# Installing Cuis Smalltalk

- Browse <https://cuis.st>
- Click Download link
- Click “Cuis Stable Release” or “Cuis Rolling Release” (preferred)
- Click Code button and copy GitHub repository URL
- Clone repository

# Starting, Saving, Quitting

- To start an image, double-click it
  - in a file like `Cuis-Smalltalk-Dev/CuisImage/Cuis7.1-6713.image`
- To save changes to an image, open World menu and select  
**Save Image**, **Save Image as...**, or **Save Image and Quit**
  - avoid saving changes to base image
- To quit an image, open World menu and select  
**Save Image and Quit** or **Quit Without Saving**

# Updating Image

- To install latest updates
  - `git pull` repository
  - open World menu and select **Changes ... Install New Updates**

# Delimiters

- **Single quote** - surrounds literal String instances
  - 'Hello, World!'
- **Double quote** - surrounds comments
  - "Answers largest number in collection"
- **Vertical bars** - surrounds list of temporary (local) variables in a method or block
  - | color price size |
- **Square brackets** - surrounds a block (anonymous function)
  - [:a :b | a + b]
- **Period** - separates statements
  - a := 2. b := 3. c := a \* b. c print

# Message Syntax ...



- Three kinds
  - unary - **receiver unaryMsg**
    - example: `'Hello, World!' print`
  - binary - **receiver binaryMsg argument**
    - example: `width * height`
  - keyword - **receiver kw1: arg1 kw2: arg2 kw3: arg3**
    - example: `dogs at: 'Comet' put: 'Whippet'`
- Precedence is unary, binary, then keyword
- Evaluated left to right
- Parentheses change evaluation order

Binary message names can only contain one or more of the following characters:  
+ - \* / \ ~ < > = @ % | & ? ,

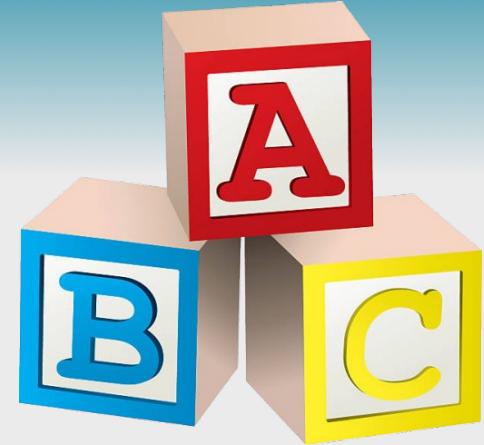
# ... Message Syntax

- An object sends a message to itself using `self` as receiver
- Statements in method bodies are separated by periods
- Message cascade ( ; )
  - sends multiple messages to same receiver
  - ex. `Transcript show: 'Hello'; newLine; show: ' World!'`
- Message chain ( : : )
  - sends message to result of previous message
  - removes need to surround previous expression with parentheses

```
1 + 2 squared -> 5  
(1 + 2) squared -> 9  
1 + 2 :: squared -> 9
```



# Blocks ...



- Deferred set of message sends that are not evaluated until block value is requested
- Value is that of last expression

```
[1 + 2] value 3
```

- Can take arguments

```
[:a :b | a + b] value: 1 value: 2 3
```

OR

```
[:a :b | a + b] valueWithArguments: #(1 2) 3
```

syntax for a  
compile-time  
literal array

# ... Blocks

- Can be assigned to variables
- Can be passed to methods
- Can be returned from methods
- Can declare temporary (local) variables
- Can contain multiple statements
- Are closures, so can access in-scope variables

```
average := [:a :b |  
    sum |  
    sum := a + b.  
    sum / 2.0  
]
```



# More Syntax



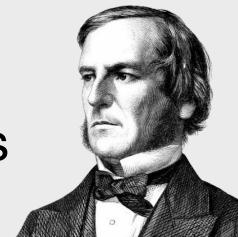
- Literal strings are delimited by single quotes

```
'Hello, World!'
```

- Comments are delimited by double quotes

```
"This is a greeting."
```

- Boolean values `true` and `false` are singleton instances of `True` and `False` classes which are subclasses of `Boolean`



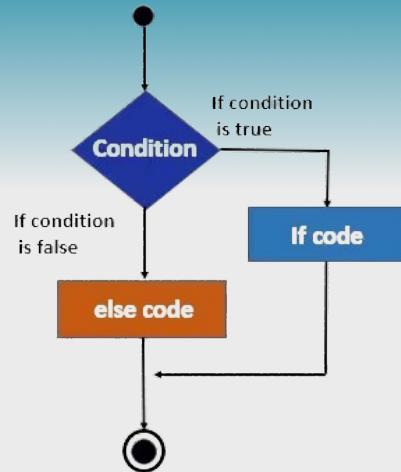
George Boole

# Conditional Logic

- Implemented as methods in **Boolean** class
- Examples

```
result := a < b ifTrue: ['less'] ifFalse: ['more']

color := 'blue'.
assessment := color caseOf: {
    ['red'] -> ['hot'].
    ['green'] -> ['warm'].
    ['blue'] -> ['cold']
}
```

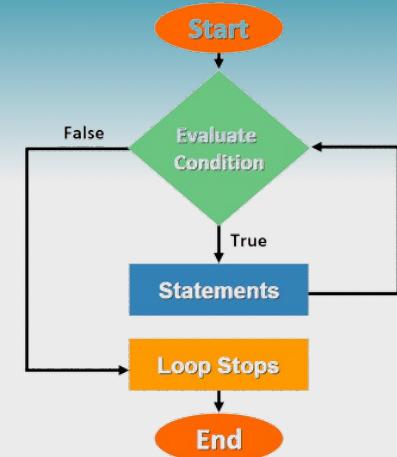


# Iteration

- Many methods in provided classes support iteration
  - such as `Integer`, `Interval`, and `BlockClosure`
- Examples

```
3 timesRepeat: ['Ho' print]  
  
interval := 1 to: 10 by: 2.  
interval do: [:n | n print]  
  
1 to: 10 by: 2 do: [:n | n print]
```

```
n := 0.  
[n = 10] whileFalse: [  
  n := 10 atRandom.  
  n print.  
]
```



- Also see `do:` method in collections on next slide

# Collections

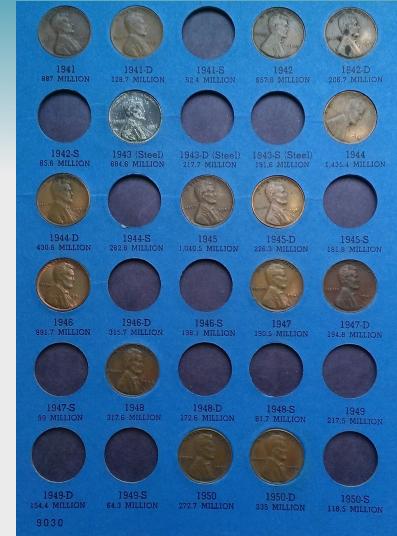
- Many provided collection classes in a class hierarchy (some shown here)
- **Collection** (abstract)
  - **SequenceableCollection** (abstract)
    - **ArrayedCollection** (abstract)
      - **Array**
      - **Heap**
      - **Interval**
      - **LinkedList**
    - **OrderedCollection**
      - **SortedCollection**
  - **Bag**
  - **Set**
    - **Dictionary**
      - **OrderedDictionary**

```
coll := OrderedCollection newFrom: #(1 2).
coll add: 3.
coll do: [ :n | (n * 2) print ]
```

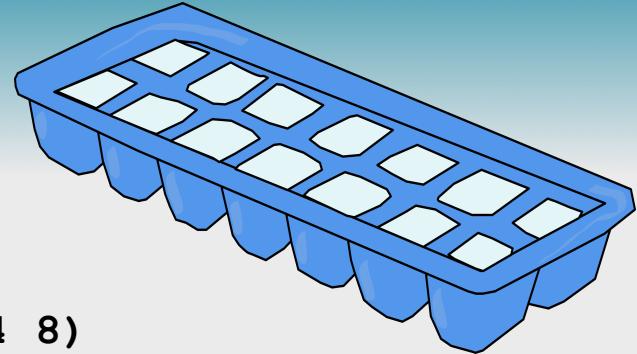
2  
4  
6

```
#(1 2 3) collect: [:n | n * 2]
```

```
#(2 4 6)
```



# Arrays



- Literal syntax
  - list of compile-time literal values separated by spaces: #(1 4 8)
  - list of run-time expressions separated by periods: {score1. score2. score3}
- Indexed from **1** rather than 0

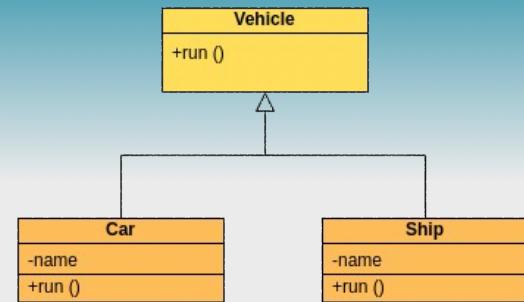
# Map, Filter, Reduce

- **Map** creates a new collection with same size as original
  - called “collect” in Smalltalk
- **Filter** creates new collection containing subset of original
  - called “select” in Smalltalk
- **Reduce** creates a single value from the elements of a collection
  - called “inject” in Smalltalk

```
numbers := #(1 2 3 4).
numbers collect: [:n | n * 2]. "#(2 4 6 8)"
numbers select: [:n | n odd]. "#(1 3)"
numbers inject: 0 into: [:acc :n | acc + n]. "10"
```

# Class Definitions

- Must be a subclass of some existing class
- Class name is a symbol
  - an interned **String**
- 3 space-separated lists
  - instance variable names
  - class variable names
  - pool dictionary names
- Category name



```
Object subclass: #Todo
instanceVariableNames: 'done text'
classVariableNames: ''
poolDictionaries: ''
category: 'TodoApp'
```

# Instance Variables

- Always private
- To expose, define accessor methods

```
text
```

```
^text
```

```
text: aString
```

```
text := aString
```

^ returns an object from a method  
and is typically rendered as ↑

:= assigns a value to a variable  
and is typically rendered as ←



# Methods

- Always public
- Always return an object
  - if none specified, `self` is returned to support chaining
- Methods that should only be used by others in the same class, should be in a method category whose name begins with “**private**”
  - ex. method `emptyCheck` in `Collection` class is in **private** method category

# Method Arguments

- Follow convention for argument names that describes expected type

```
findFirst: aBlock startingAt: aNumber  
...  
  
from: fromNumber to: toNumber do: aBlock  
...
```

- There's no such thing as passing the wrong type,  
only passing an object that doesn't respond  
to all the messages that will be sent to it

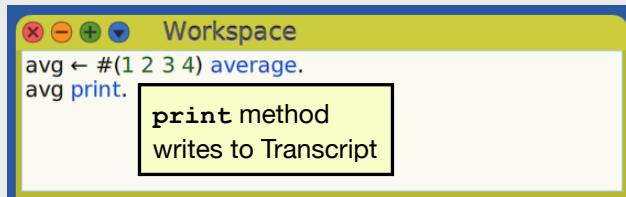
# Pseudo Variables

- Reserved words whose value is provided automatically and cannot be modified
- Six of them: **true**, **false**, **nil**, **self**, **super**, and **thisContext**
- **true** and **false** represent Boolean values
  - refer to singleton instances of **True** and **False** classes
- **nil** represents lack of a real value
- **self** is used in instance methods to refer to current object
  - also in class methods to refer to current class
- **super** is used in instance methods to refer to superclass of current object
- **thisContext** represents top frame of run-time stack
  - used to implement development tools like Debugger

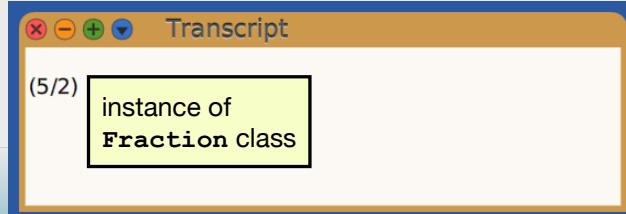
**self** and **super** are often used as the receiver of messages

# Development Environment

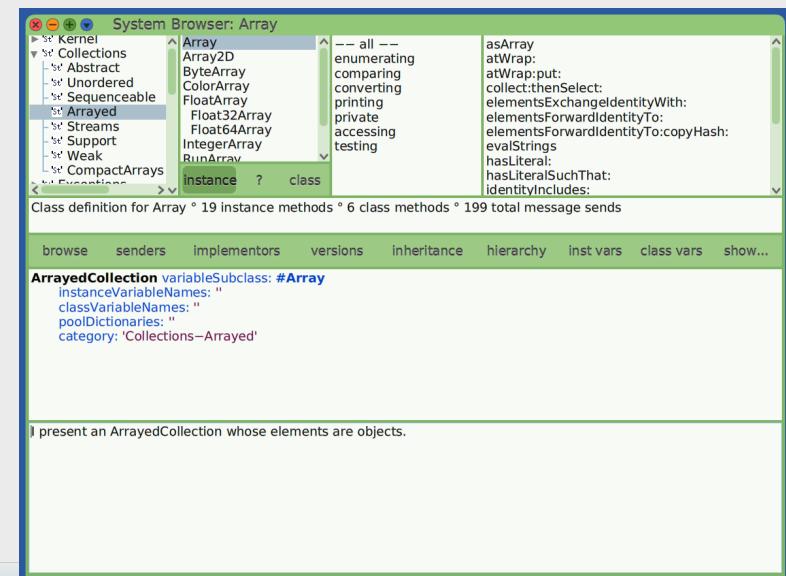
- Many kinds of windows
- Main ones are
  - **System Browser** - for reading, writing, and modifying code
    - see next slide
  - **Workspace** - for experimenting with message sends



- **Transcript** - for displaying output



By default, focus moves with cursor position.



# System Browser

class categories    classes in selected category    method categories    methods in selected category

System Browser: Array

Kernel  
Collections  
Abstract  
Unordered  
Sequenceable  
Arrayed  
Streams  
Support  
Weak  
CompactArrays  
Exceptions

Array  
Array2D  
ByteArray  
ColorArray  
FloatArray  
Float32Array  
Float64Array  
IntegerArray  
RunArray

-- all --  
enumerating  
comparing  
converting  
printing  
private  
accessing  
testing

asArray  
atWrap:  
atWrap:put:  
collect:thenSelect:  
elementsExchangeIdentityWith:  
elementsForwardIdentityTo:  
elementsForwardIdentityTo:copyHash:  
evalStrings  
hasLiteral:  
hasLiteralSuchThat:  
identityIncludes:

instance ? class

Class definition for Array ° 19 instance methods ° 6 class methods ° 199 total message sends

browse senders implementors versions inheritance hierarchy inst vars class vars show...

**ArrayedCollection** variableSubclass: #Array  
instanceVariableNames: ""  
classVariableNames: ""  
poolDictionaries: ""  
category: 'Collections-Arrayed'

I present an ArrayedCollection whose elements are objects.

Code is written and read one method at a time, not in one source file per class.

# Saving Code

- Three options
  - in an **image**
    - from World menu, select **Save Image**, **Save Image as...**, or **Save Image and Quit**
  - in a **package**
    - from Installed Packages window, select a modified package and click “save” button
    - for packages with an associated GitHub repository, push changes
    - can install in different images
  - in a **fileout**
    - from a Browser window, select a package, class, or method, right click, and select **fileOut**
    - uses “chunk” format described on next slide
    - can **fileIn** from a File List window

# Chunked Format

- `.st` files are human-readable text file that contain “chunks”
- Each chunk is delimited by exclamation marks (bang)
- A chunk can contain
  - `From` line that gives version of Smalltalk that created the file and creation date/time
  - `classDefinition` that associates a class with a class category
  - `subclass:` message send that creates a class
  - `methodsFor` which states that the method definitions that follow are in a given method category
  - method definition
  - message send that creates an object
  - message send that executes code for its side effects

# File List

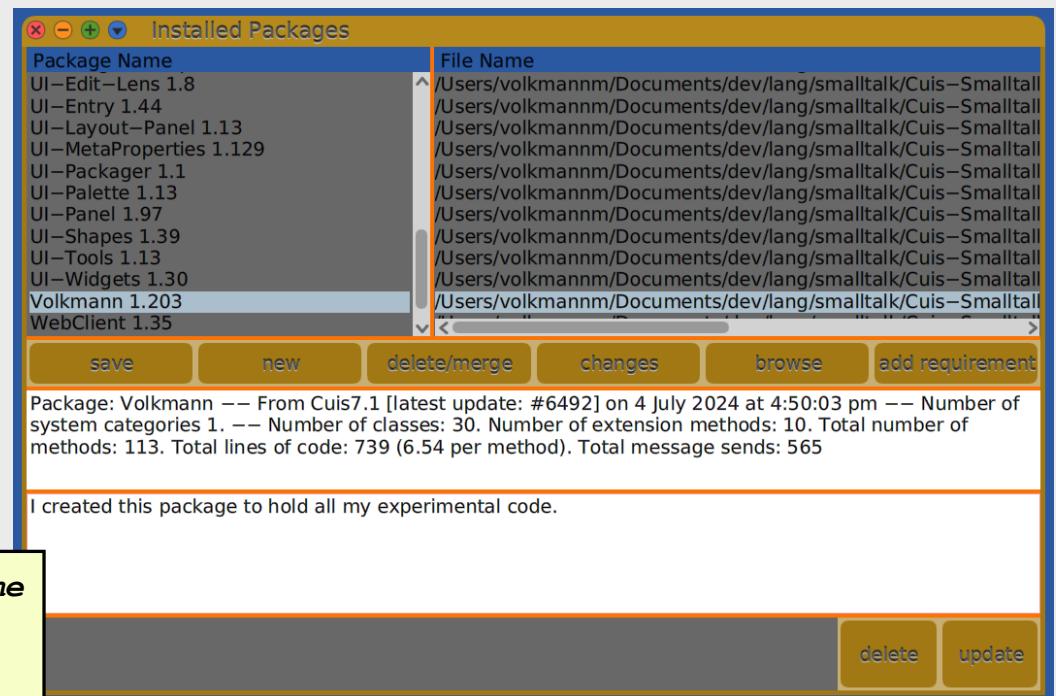
- Can navigate file system
- Can create, rename, view contents, and delete directories
- Can create, rename, view contents, edit (text files), and delete files
- For **.st** files
  - can open “code changes” window to view contents one class definition or method at a time
  - can “filein” to load into current image

		name	[v] – date	size
		UserPrefs.txt	( 2024/10/03 12:23:36	142 )
		hello	( 2024/09/28 13:49:07	33,432 )
		hello.c	( 2024/09/28 13:48:31	138 )
		pets.db	( 2024/09/23 13:40:56	12,288 )
		LayoutMorph.st	( 2024/08/18 20:12:38	53,354 )
		todo-app-workspace	( 2024/08/18 07:00:52	244 )
		my-workspace.st	( 2024/08/17 16:45:07	984 )
		layout-issue	( 2024/08/12 18:57:53	559 )
		BeverlyHillbillies.txt	( 2024/05/31 10:44:09	797 )
NO FILE SELECTED				
-- Folder Summary --				
		UserPrefs.txt	( 2024/10/03 12:23:36	142 )
		hello	( 2024/09/28 13:49:07	33,432 )
		hello.c	( 2024/09/28 13:48:31	138 )
		pets.db	( 2024/09/23 13:40:56	12,288 )
		LayoutMorph.st	( 2024/08/18 20:12:38	53,354 )
		todo-app-workspace	( 2024/08/18 07:00:52	244 )
		my-workspace.st	( 2024/08/17 16:45:07	984 )
		layout-issue	( 2024/08/12 18:57:53	559 )

# Installed Packages

- Displays list of installed package
- Can create, browse, uninstall, and delete packages
- Can view changes to a package and save them
- Can add dependencies to a package

Packages in a directory named **Cuis-Smalltalk-Name** can also be installed from a Workspace with **Feature require: 'Name'**



# DEMO #1

- Open a Browser
- Hover over class categories pane
- Press cmd-f and enter “**Stri**”
- Select “**String**” class
- Hover over methods pane
- Press “**i**” to scroll to first entry that begins with that
- Select “**isEmpty**” method
- Note that it sends the message “**size**” to **self**
- Click **size** method and discuss primitives (62 returns collection size)
- Click “?” button to see class comment
- Click “class” button to see list of class methods

# DEMO #2

- Open Browser
- Right-click in class categories pane and select “add item...”
- Enter “**Demo**”
- In bottom code pane, change “**NameOfSubclass**” to “**Greeter**”
- Press cmd-s to save new class
- Add “**message**” to **instanceVariableNames** list
- In message categories pane, select “**as yet unclassified**”
- Add these methods
- Place first two methods in method category “accessing” and last one in “printing”
- Open Workspace
- Enter following lines, select them, and press cmd-d to “Do it”

```
g := Greeter new message: 'Hello, World!'.
g greet
```

```
message
^message

message: aString
message := aString

greet
message print
```

# Inspect/Explore Windows



- Inspect windows examine a single object
- Explore window examine a tree of objects starting from one
- Both can send messages to selected object in bottom pane

**Inspect: a Dog**

self	breed: 'Whippet'
all inst vars	name: 'Comet'
breed	
name	

inspect window

self speak bark .

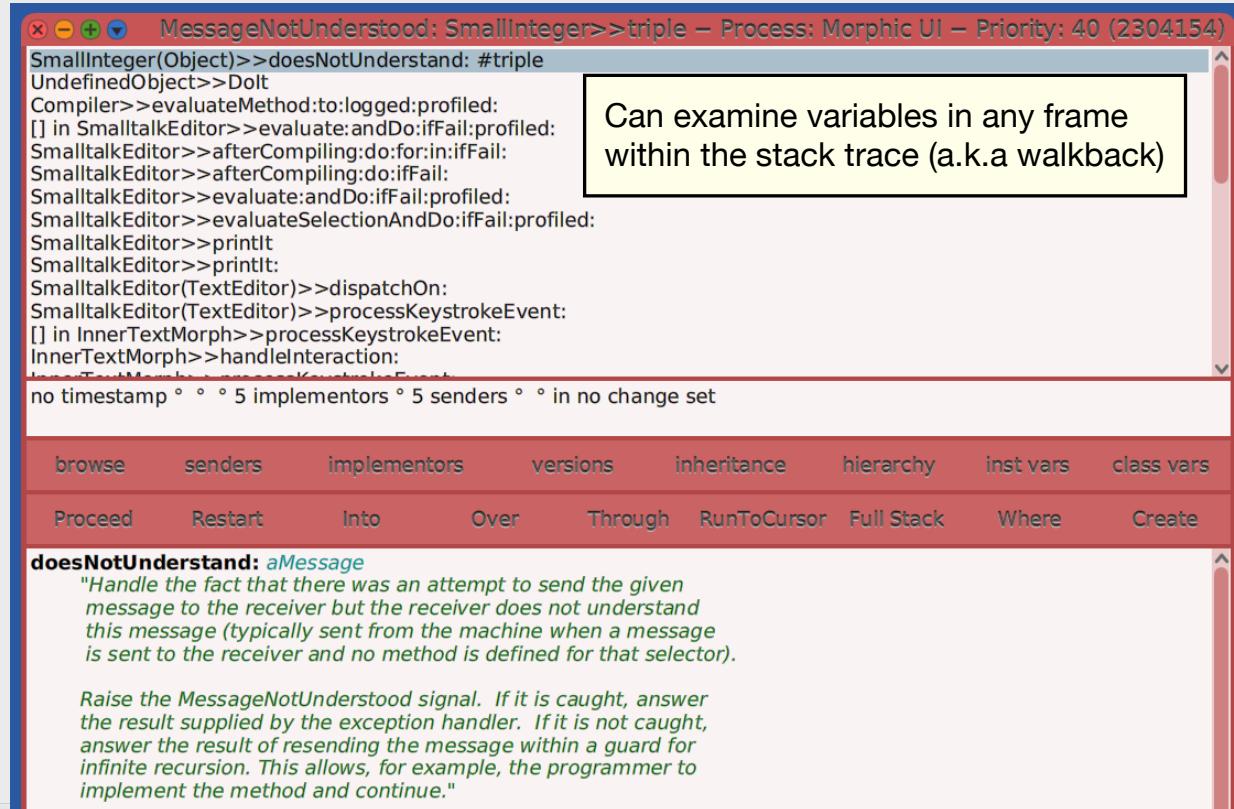
**a Dog**

root: a Dog	
▶ breed: 'Whippet'	
▶ name: 'Comet'	
└ 1: \$C	
└ 2: \$o	
└ 3: \$m	
└ 4: \$e	
└ 5: \$t	

explore window

self speak bark .

# Most Common Error



# DEMO #3

- In the Workspace, enter “g” (variable name)
- Press cmd-i to “Inspect it”
- Press cmd-l to “Explore it”
- In the bottom pane of either window, enter self message asUppercase
- Press cmd-p to “Print it”

# DEMO #4

- Open a “Message Names” window
- Enter “`at:put:`”
- Select “`Dictionary at:put:`”
- Study signature, method comment, and method body
  - `findElementOrNil` is implemented in `Set` which is the superclass of `Dictionary`
  - `ifNil:ifNotNil:` is implemented in `ProtoObject` which is the superclass of `Object` which is a superclass of every class

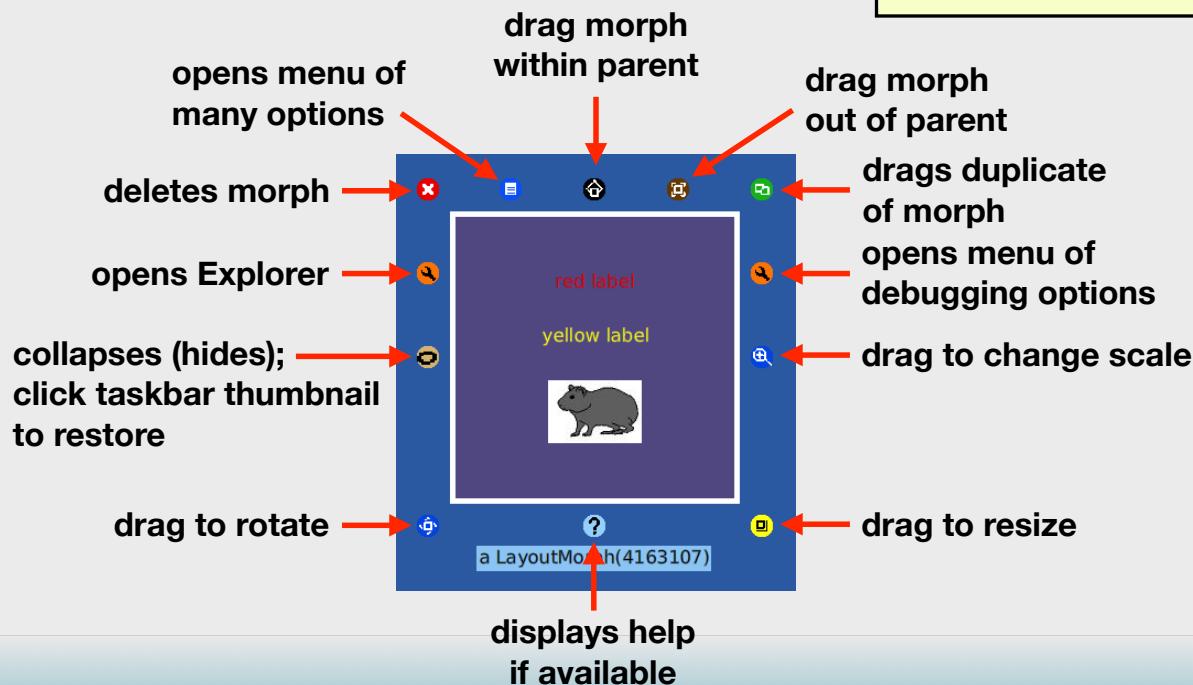
# Morphic

- GUI framework included in popular Smalltalk images such as Squeak, Pharo, and Cuis
- Defines user interfaces with “morphs”
  - objects that know how to render themselves in the Smalltalk environment
  - referred to as widgets or components in other GUI frameworks
- Examples
  - `CheckButtonMorph`, `ImageMorph`, `LabelMorph`, `LayoutMorph`, `RadioButtonMorph`, `TextEntryMorph`, `TextModelMorph`

# Halos

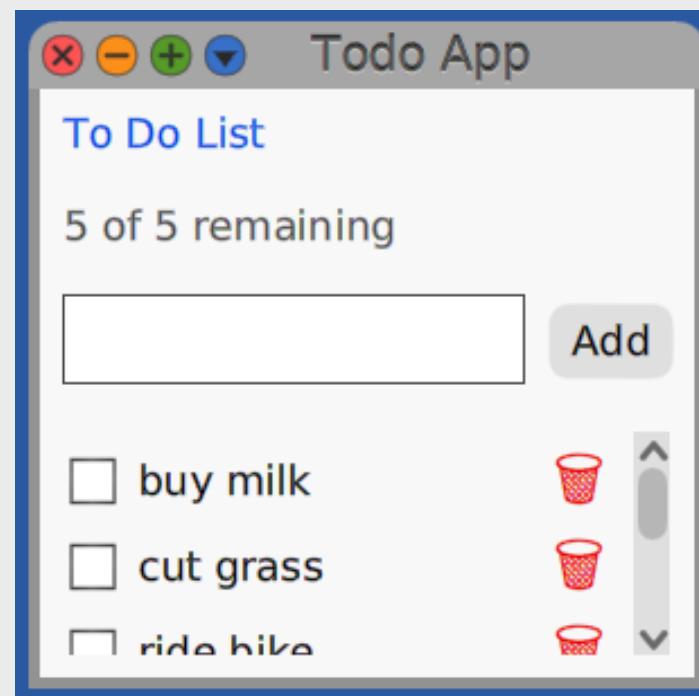
- cmd-click a morph to display its halo which is a collection of “handles” around its perimeter

Hover over any handle to see a tooltip that describes it.



# Todo App

- Demonstrate app in Cuis
- Walk through code in Cuis



# Resources

- **Smalltalk Wikipedia page**
  - <https://en.wikipedia.org/wiki/Smalltalk>
- **Cuis Smalltalk**
  - <https://cuis.st/>
- **Pharo Smalltalk**
  - <https://pharo.org/>
- **Squeak Smalltalk**
  - <https://squeak.org/>
- **Byte Magazine issue on Smalltalk**
  - <https://archive.org/details/byte-magazine-1981-08>



# Wrap Up

- For me, Smalltalk has the most elegant syntax of any programming language
- Even if you choose not to use Smalltalk, there are many ideas from it that can be applied to other programming languages
  - especially in its development environment
- Todo App code at  
<https://github.com/mvolkmann/Cuis-Smalltalk-TodoApp>