

 $u^{\scriptscriptstyle b}$ 

#### **2. Smalltalk Basics**



ST – Introduction

#### **Birds-eye view**



Less is More — simple syntax and semantics uniformly applied can lead to an expressive and flexible system, not an impoverished one.



#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

Selected material courtesy Stéphane Ducasse

#### Roadmap

#### > Everything is an Object

- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

#### **Objects in Smalltalk**

- > *Everything* is an object
  - Things only happen by message passing
  - Variables are dynamically bound
- > State is private to objects
  - "protected" for subclasses
- > Methods are public
  - "private" methods by convention only
- > (Nearly) every object is a reference
  - Unused objects are garbage-collected
- > Single inheritance

#### Accept, Dolt, Printlt and InspectIt

> Accept

— Compile a method or a class definition

- > Dolt
  - Evaluate an expression
- > Printlt
  - Evaluate an expression and print the result (#printOn:)
- > InspectIt
  - Evaluate an expression and inspect the result (#inspect)

🞑 find...(f) 🖸 find again (g) set search string (h) no again (j) 🎓 숙 undo (z) Copy (c) % cut (x) naste (v) 🛅 paste... 🙆 do it (d) 🖏 print it (p) (i) inspect it Q explore it (I) debug it accept (s) 🔀 cancel (I) implementors (m) methods containing (E) senders (n) more...



- > At anytime, *in any tool*, we can dynamically ask the system to evaluate an expression.
  - To evaluate an expression, select it and with the middle mouse button apply dolt.

Transcript show: 'hello world'

"Hel	0	Wo	orl	d	"
				· · · ·	

000	ThreadSafeTranscr	ipt 🗢	
hello world			
	$\Theta \Theta \Theta$	Shout Workspace	0
	Transcript sho Transcript cr	w: 'hello world'.	

> Transcript is a kind of "standard output"

— a TextCollector instance associated with the launcher.

#### **Everything is an Object**

## Smalltalk is a consistent, uniform world, written in itself

- You can learn how it is implemented, you can extend it or even modify it.
- All the code is available and readable.
  - The workspace is an object.
  - The window is an instance of SystemWindow.
  - The text editor is an instance of ParagraphEditor.
  - The scrollbars are objects too.
  - 'hello word' is an instance of String.
  - #show: is a Symbol
  - The mouse is an object.
  - The parser is an instance of Parser.
  - The compiler is an instance of Compiler.
  - The process scheduler is also an object.

#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

#### **Smalltalk Syntax on a Postcard**

```
exampleWithNumber: x
"A method that illustrates every part of Smalltalk method syntax
except primitives. It has unary, binary, and key word messages,
declares arguments and temporaries (but not block temporaries),
accesses a global variable (but not and instance variable),
uses literals (array, character, symbol, string, integer, float),
uses the pseudo variable true false, nil, self, and super,
and has sequence, assignment, return and cascade. It has both zero
argument and one argument blocks. It doesn't do anything useful, though"
    y |
   true & false not & (nil isNil) ifFalse: [self halt].
   y := self size + super size.
   #($a #a 'a' 1 1.0)
       do: [:each | Transcript
          show: (each class name);
          show: (each printString);
          show: ' '].
   ^ x < y
```

#### Language Constructs

^	return
""	comment
#	symbol or array
1 1	string
[ ]	block or byte array (VisualWorks)
•	statement separator
;	message cascade
•••	local or block variable
:=	assignment (also _ or ←)
\$	character
:	end of selector name
_er_	number exponent or radix
!	file element separator (used in change sets)
<primitive:></primitive:>	for VM primitive calls

#### **Examples**

comment:	"a comment"		
character:	\$c \$h \$a \$r \$a \$c \$t \$e \$r \$s \$# \$@		
string:	'a nice string'		
symbol:	#mac #+		
array:	#(1 2 3 (1 3) \$a 4)		
dynamic array (Pharo):	{ 1 + 2 . 3 / 4 }		
Integer:	1, 2r101		
real:	1.5, 6.03e-34,4, 2.4e7		
fraction:	1/33		
boolean:	true, false		
pseudo variables	self, super		
point:	10@120		

Note that @ is not an element of the syntax, but just a message sent to a number. This is the same for /, bitShift, ifTrue:, do: ...

#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

#### **Messages instead of keywords**

- In most languages, basic operators and control constructs are defined as language constructs and keywords
- > In Smalltalk, there are only *messages* sent to objects
  - bitShift: (>>) is just a message sent to a number

10 bitShift: 2

— ifTrue: (if-then-else) is just a message sent to a boolean

(x>1) ifTrue: [ Transcript show: 'bigger' ]

- do:, to:do: (loops) are just messages to collections or numbers

#(a b c d) do: [:each | Transcript show: each ; cr]
1 to: 10 do: [:i | Transcript show: i printString; cr]

- > Minimal parsing
- > Language is *extensible*

Smalltalk Syntax	
Every expression is a me	essage send
> Unary messages	Transcript cr 5 factorial
> Binary messages	3 + 4
> Keyword messages	Transcript show: 'hello world' 2 raisedTo: 32

3 raisedTo: 10 modulo: 5

Preceden	се
----------	----

(...) > Unary > Binary > Keyword

- 1. Evaluate *left-to-right*
- 2. Unary messages have *highest precedence*
- 3. Next are binary messages
- 4. Keyword messages have *lowest precedence*

2 raisedTo: 1 + 3 factorial 128

5. Use *parentheses* to change precedence

#### **Binary Messages**

- > Syntax:
  - aReceiver *aSelector* anArgument
  - Where *aSelector* is made up of 1 or 2 characters from:

 $+ - / \setminus * \sim < > = @ % | & ! ? ,$ 

- Except: second character may not be \$

> Examples:

#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

More syntax	
<pre>&gt; Comments are enclosed in double quotes     "This is a comment."</pre>	
<pre>&gt; Use periods to separate expressions Transcript cr. Transcript show: 'hello world'. Transcript or "NP: dep(t peed one here")</pre>	

> Use semi-colons to send a cascade of messages to the same object

Transcript cr; show: 'hello world'; cr



> Use := to assign a value to a variable

> Old fashioned assignment operator: ← (must type "\_")



Methods *always* return a value
 — By default, methods return self



#### > Use *square brackets* to delay evaluation of expressions



#### Variables

- > Local variables within methods (or blocks) are delimited by |var|
- > Block parameters are delimited by : var

```
OrderedCollection>>collect: aBlock
    "Evaluate aBlock with each of my elements as the argument."
    | newCollection |
    newCollection := self species new: self size.
    firstIndex to: lastIndex do:
        [ :index |
        newCollection addLast: (aBlock value: (array at: index))].
    ^ newCollection
```



#### **Control Structures**

> Every control structure is realized by message sends



1 to: 10 do: [:n Transcript show: n; cr ]

(1 to: 10) do: [:n Transcript show: n; cr ]



> Factory methods

1@2"a Point"1/2"a Fraction"

С	reating classes
>	Send a message to a class (!)
	<pre>Number subclass: #Complex instanceVariableNames: 'real imaginary' classVariableNames: '' poolDictionaries: '' category: 'ComplexNumbers'</pre>

#### **Some Conventions**

- > Method selector is a *symbol*, e.g., #add:
- > Method scope conventions using >>
  - Instance Method defined in the class Node

Node>>accept: aPacket

- Class Method defined in the class Node class (i.e., in the class of the the class Node)



Node class>>withName: aSymbol

> aSomething is an instance of the class Something

#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

#### **Change sets**

#### > Make sure your changes are logged to a new change set



ST - Smalltalk Basics

### **SUnit**



#### Money

> We will implement the Money example in Smalltalk

- First, we develop a test case for a single currency

```
TestCase subclass: #MoneyTest
  instanceVariableNames: 'chf2 chf8 chf10'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Money'
```

**NB:** This is just a message sent to the TestCase class object (!)

### SetUp

#### > We will need setters for the private Money state

# MoneyTest>>setUp chf2 := Money new currency: 'CHF'; amount: 2. chf8 := Money new currency: 'CHF'; amount: 8. chf10 := Money new currency: 'CHF'; amount: 10.

define new clas	5		
declare global			
declare class va	riable		
TestMoney			
MonitorDelay			
Month			
Model			
MonthTest			
MonitorTest			
Morph			
MostUsedMetho	dCategorySel	ectionStrategy	
MorphHierarchy			
Monitor			
cancel			

#### **Protocols**

#### Classify methods into protocols that reveal their intent



#### MoneyTest>>testEquals

> Some obvious tests

```
MoneyTest>>testEquals
  self assert: chf2 = chf2.
  self assert: chf2 = (Money new currency: 'CHF'; amount: 2).
  self assert: chf2 != chf8.
```

#### Money

> We define Money as a subclass of Object, with getters and setters

```
Object subclass: #Money
  instanceVariableNames: 'currency amount'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Money'
```

```
Money>>currency: aString
currency := aString. Money>>currency
^ currency
Money>>amount: aNumber
amount:= aNumber. Money>>amount
^ amount
```

## **Failing tests**

000	Test Ri	inner	0	
Tests-Localization Tests-Mac Tests-Monticello Tests-Morphic Tests-Multilingual Tests-Object Events	estMoney	1 run, 0 passes, 0 expected fa failures, 0 errors, 0 unexpecte TestMoney>>#testEquals	ailures, 1 ed passes	
Tests-ObjectsAsMeth	000	TestFailure: As	sertion failed	0
Tests-PackageInfo Tests-PrimCallControl Tests-Release Tests-ST80 Tests-ScriptLoader Tests-System Tests-System-Setting Tests-SystemTests-Su Tests-SystemTests-Su Tests-ToolBuilder Tests-ToolBuilder Tests-ToolS Tests-Traits Tests-Traits Tests-Traits Tests-VB-Regex Tests-VB-Regex Tests-VM ToolsTest-Inspector ToolsTest-PointerFinde Money	TestMoney(TestC TestMoney(TestC TestMoney>>test TestMoney(TestC [] in TestMoney( BlockClosure>> Proceed Re testEquals self assert self assert self assert	Case)>>signalFailure: Case)>>assert: stEquals Case)>>performTest TestCase)>>runCase ensure: start Into Over Thro : chf2 = chf2. : chf2 = (Money new currency : chf2 != chf8.	ugh Full Stack : 'CHF'; amount:	Run to Here Where
Run Selected Run	all inst vars testSelector chf2		thisContext stack top all temp vars	

```
Comparisons
```

Money>>= aMoney
 ^ self currency = aMoney currency
 and: [ self amount = aMoney amount ]

Сс	onstructors
	self assert: chf2 = chf2.
	<pre>self assert: chf2 =</pre>
	(Money currency: 'CHF' amount: 2).

#### We need a constructor on the *class side* of Money

#### **Class methods**

OB Package Browser: Money					
C (class search), i (implementor search), #C (class ref search), #s (sender search)					
ProtoObject Object Money	Money TestMoney	all as yet unclassified long (0) uncommented (1) required (?) local (1)	currency:amount:		
pkgs hier. groups currency: aCurrency amount: anAu ^ self new currency: aCurre	instance ? class traits nount ncy; amount: anAmount; yourself				

#### **NB:** Which "self" is referred to in the method body?

© Oscar Nierstrasz

Ad	d	iti	0	n
				_

MoneyTest>>testAdd
 self assert: chf2 + chf8 = chf10

And so on ...

#### Filing out your changes

> You can "file out" all your changes so they can be loaded into another image

000	Changes go to	ned"			
Money ImageForDevelopers-pha AutomaticMethodCategor AutomaticMethodCategor Nile-Base-cyrille_delaunar Ocompletion-damiencass SetUp testAdd testEquals	<ul> <li>Change Set</li> <li>Change Set</li> <li>make changes go t</li> <li>new change set (</li> <li>find(f)</li> <li>show category (s</li> <li>select change set</li> <li>rename change se</li> <li>file out (o)</li> <li>mail to list</li> </ul>	Money ImageFor Automatic Automatic Nile-Base to me (m) (n) )  t (r)	AngeForDevelopers-pha itomaticMethodCategor itomaticMethodCategor le-Base-cyrille_delaunation-damiencass he (m)		
Class definition was added.	browse methods (b browse change set copy all to other si submerge into other subtract other side add preamble (p) add postscript category functions destroy change set more	o) : (B) de (c) er side : (-)  t (x)			

#### **Change Sets**

```
'From Pharol.Obeta of 16 May 2008 [Latest update: #10418] on 8 September 2009
at 1:58:51 pm'!
Object subclass: #Money
    instanceVariableNames: 'currency amount'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Money'!
TestCase subclass: #TestMoney
    instanceVariableNames: 'chf2 chf8 chf10'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Money'!
!Money methodsFor: 'as yet unclassified' stamp: 'on 7/2/2007 13:21'!
!!= aMoney
    ^ (self = aMoney) not! !
!Money methodsFor: 'as yet unclassified' stamp: 'on 7/2/2007 13:26'!
+ aMoney
    self assert: [ self currency = aMoney currency ].
    ^ Money currency: self currency amount: self amount + aMoney amount! !
!Money methodsFor: 'as yet unclassified' stamp: 'on 7/2/2007 13:20'!
= aMoney
    ^ self amount = aMoney amount and: [ self currency = aMoney currency ]! !
. . .
```



chf8 := 8 chf.

chf10 := 10 chf.

#### **Extension protocols**

To extend an existing class for a category MyProject, define the protocol \*MyProject (or \*myproject) for that class.



#### Roadmap

- > Everything is an Object
- > Syntax on a Postcard
- > Three Kinds of Messages
- > Methods, variables and blocks
- > Test-Driven Development
- > Managing Projects

## SqueakSource.com

Sq	ueal	kSoι	ırce	-1-		113	51	1.23	Version 1.3
Home	Projects	Members	Groups	Help					
Actions RSS feed Back	1	Squ	eak Exa	amples <sub>Wiki</sub>	RSS Feed	Releases	Blessings	Versions	Latest
Authenti Login	ication	Project Example Memb Creator Admin: Regist MCRttp lo us pa Links http:// http://	ct Descrip es for the Sr pers : tration Despositor per: '' ser: '' ussword: ' www.squeak www.squeak www.squeak	otion nalltalk cou Oscar N Oscar N y http://ww source.com	rse http://www lierstrasz lierstrasz w.squeaksour //SqueakExample //SqueakExample	.iam.unibe.ch/~so rce.com/Squeak es.html	cg/Teaching/Sma	Iltalk/index.html	
		Registe Total R Total V Total D	rred: eleases: 'ersions: ownloads:	19 Marc 0 3 5	ch 2006 3:59:41	pm			
XHTML I	CSS   RSS								20 March 2006

#### **Categories, Projects and Packages**

- > A system <u>category</u> MyProject (and possibly MyProject-\*) contains the classes of your application
- > A Monticello <u>package</u> MyProject contains the categories MyProject and MyProject-\*
  NB: If you have sub extension, keep the ten one among
  - NB: If you have sub-categories, keep the top one empty!
- > A SqueakSource project MyProject stores everything in the Monticello package MyProject

#### Loading a project

- > To load a project from SqueakSource.com
  - Find the project in SqueakSource
  - Open the Monticello Browser
  - Add an HTTP Repository
    - Accept the code fragment from the SqueakSource page
  - Open the repository and load the latest version





## Loading a project

Information Required	
HTTP Repository:	
MCHttpRepository location: 'http://www.squeaksource.com/SqueakExamples' user: 'on' password: '*****	Refresh       Browse       History       Changes       Load       Merge       Adopt       Copy       Diff         SqueakExamples       SqueakExamples-on.20.mcz       SqueakExamples-on.19.mcz       SqueakExamples-on.11.mcz       SqueakExamples-on.11.mcz       SqueakExamples-on.11.mcz       SqueakExamples-on.11.mcz       SqueakExamples-on.13.mcz       SqueakExample
	Name: SqueakExamples-on.20 Author: on Time: 28 October 2007, 3:51:35 pm UUID: 58d2c45b-e2a9-4758-b33b-03d978195a99 Ancestors: SqueakExamples-on.19 Added comments to metaclass hierarchy test

#### **Creating a project**

- > To create a project on SqueakSource.com
  - Define your categories MyProject or MyProject-\*
  - Create a SqueakSource project <u>named MyProject</u>
  - Define a Monticello package MyProject
  - Add an HTTP Repository for MyProject
    - Accept the code fragment from the SqueakSource page
  - Save the package

 $\Theta \Theta \Theta$ Monticello Browser  $\bigcirc$ Browse Changes Save +Repository Open Scripts History Backport +Package \* SqueakExamples (SqueakExamples-on.20) /Users/oscar/Documents/Projects/SqueakImages/pharo1 AST (AST-damiencassou.171) http://www.squeaksource.com/SqueakExamples Announcements (Announcements-adrian lienhard.22) 🥅 AutomaticMethodCategorizer (AutomaticMethodCategoria) AutomaticMethodCategorizerOB (AutomaticMethodCat Balloon (Balloon-marcus denker.30) Collections-Abstract (Collections-Abstract-sd.22) Collections-Arraved (Collections-Arraved-StephaneDuc Collections-Sequenceable (Collections-Sequenceable-c Collections-SkipLists (Collections-SkipLists-adrian lien) Collections-Stack (Collections-Stack-stephane ducasse Collections-Streams (Collections-Streams-marcus den) Collections-Strings (Collections-Strings-StephaneDuca: = < (\_\_\_\_\_

#### **Register Project**

Name: Title: MyProject My funky project

#### What you should know!

- Solution Not the second se
- What is the difference between a comment and a string?
- S Why does 1+2\*3 = 9?
- What is a cascade?
- Solution >>> Now is a block like a lambda expression?
- Solution → Solution → Solution → Solution → How do you create a new class?
- Solution Not the second section of the second sected and the second second

#### Can you answer these questions?

- Why does Smalltalk support single (and not multiple) inheritance?
- Why do you need to declare local variables if there are no static types?
- Solution Not State S
- Substitution Solution Soluti Solution Solution Solution Solution Solution Soluti

#### License

#### http://creativecommons.org/licenses/by-sa/3.0/



#### **Attribution-ShareAlike 3.0 Unported**

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

#### Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.