Squeak, learning oriented object programming with the right tool

Hilaire Fernandes

CRDP Aquitaine - OFSET

Fall 2005

- Working at CRDP Aquitaine - France, a resource centre for pedagogical documentation and ICT in education. We do consulting, support and development for the educational institution.
- President of the OFSET organisation, to support free software development in education.
- Author of Dr.Geo, an interactive geometry software. It is a *massive* oriented object C++ software.
- ....and a seduced Squeak/Smalltalk developer

Focus on

- Object programming concept
- Inheritance, polymorphism, attribute
- Writing proper object code:
  - subtypes instead of subclasses
  - is-a instead of part-of
  - tell don't ask (polymorphism vs test block)
  - write reusable code
- Refactoring code
- Writing tests

What are objects?

- An abstraction of a real world object with:
  - responsibilities
  - behaviours
- It is not a data structure

Key points

- Everything is an object

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.
- The model is consistent and uniform thanks to its pure object aspect

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.
- The model is consistent and uniform thanks to its pure object aspect
- High level iterators

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.
- The model is consistent and uniform thanks to its pure object aspect
- High level iterators
- Objects, messages and closures, that's it!

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.
- The model is consistent and uniform thanks to its pure object aspect
- High level iterators
- Objects, messages and closures, that's it!
- No *inextensible* operators

Key points

- Everything is an object
- No trouble with type: *only* references to object. One may says Smalltalk is strongly typed.
- The model is consistent and uniform thanks to its pure object aspect
- High level iterators
- Objects, messages and closures, that's it!
- No *inextensible* operators
- No public, protected or whatever object attributes $\rightarrow$ only protected

The object model

- Everything is an object
- Only message sends and closures
- Public methods
- Protected attributes
- Single Inheritance
- Nothing special for static

Smalltalk syntax in a postcard

```
exampleWithNumber: x
"A method that illustrates every part of Smalltalk method syntax
except primitives. It has unary, binary, and keyword messages,
declares arguments and temporaries, accesses a global variable
(but not an 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."

|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: ' '].
^ x < y
```

Simplicity to concentrate on what matter

- Single inheritence: easy to lookup for a method owner, it is in self or ancestors
- No trouble with type: only reference to object
- No trouble with attributes: all *protected*.
- No trouble with methods: all *public*.
- Garbage collector

Coercion - implicit type conversion

Object can *mutate* when necessary, for example bellow a is a reference to a `Fraction` instance, then to a `SmallInteger` instance:

```
a := 1/3.
a class -> Fraction
a := a + (2/3)
a class -> SmallInteger
```

High level iterators

High level iterators are used with block closure.

- selection in a number collection:

```
#(1 2 3 4 5) select: [:i | i odd] -> #(1 3 5)
```

- calculus over a number collection:

```
#(1 2 3 4 5) collect: [:i | i * i ] -> #(1 4 9 16 25)
```

- transforming characters:

```
'taiwan' withIndexCollect:
    [:c :i|
    (i odd)
        ifTrue: [c asUppercase]
        ifFalse: [c]]
-> 'TaIwAn'
```

As usual block closure are object:

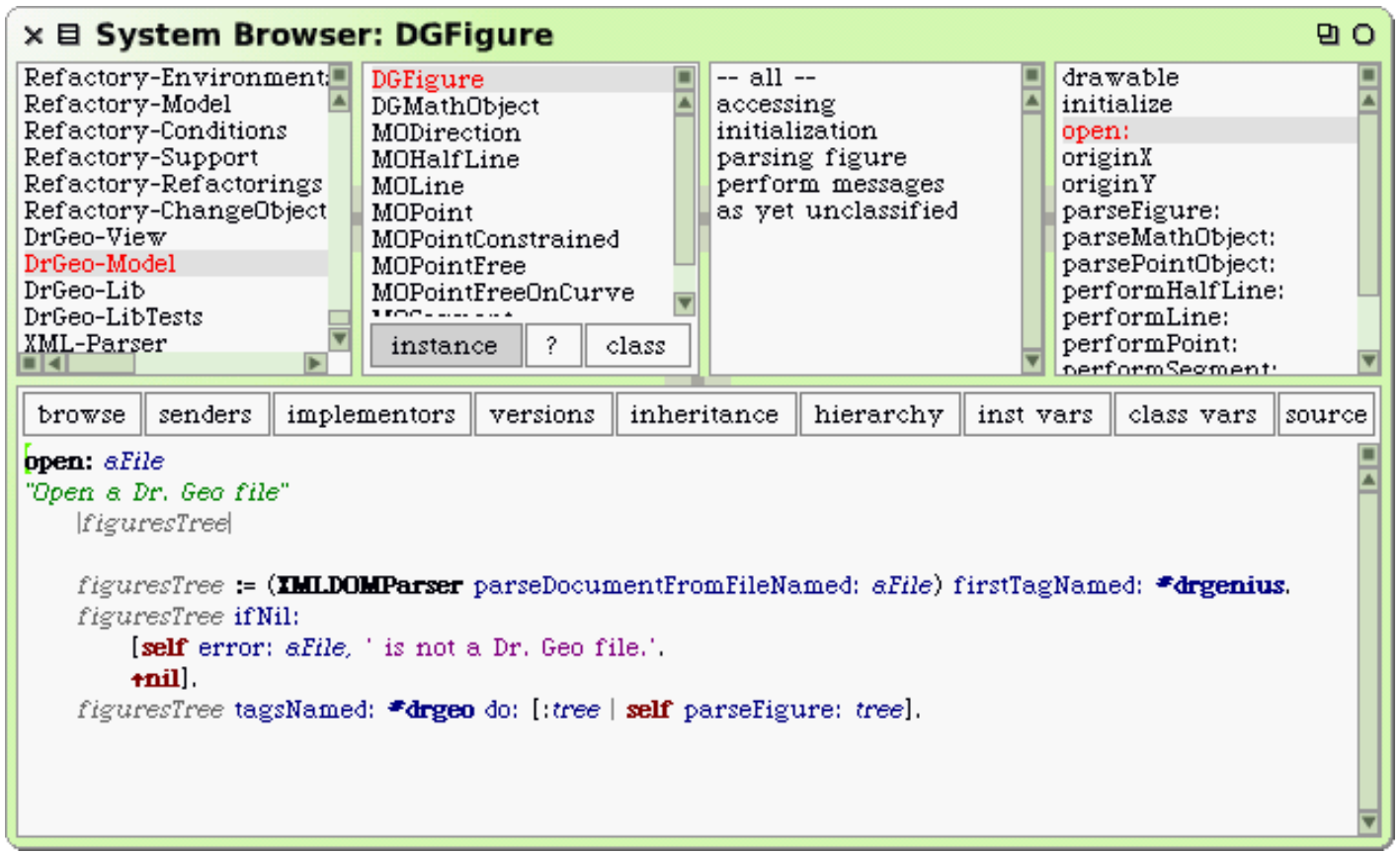
- It is anonymous method ($\approx \lambda$-function in Scheme).
- It can be passed to method as argument (see previous slide)
- It can be referenced by a variable and used as a function:

```
f := [:x| (x raisedTo: 3)]
f value: 5
  -> 125
(1 to: 10) collect: [:x | f value: x]
  -> #(1 8 27 64 125 216 343 512 729 1000)
```

- You will love it and you will use it a lot!

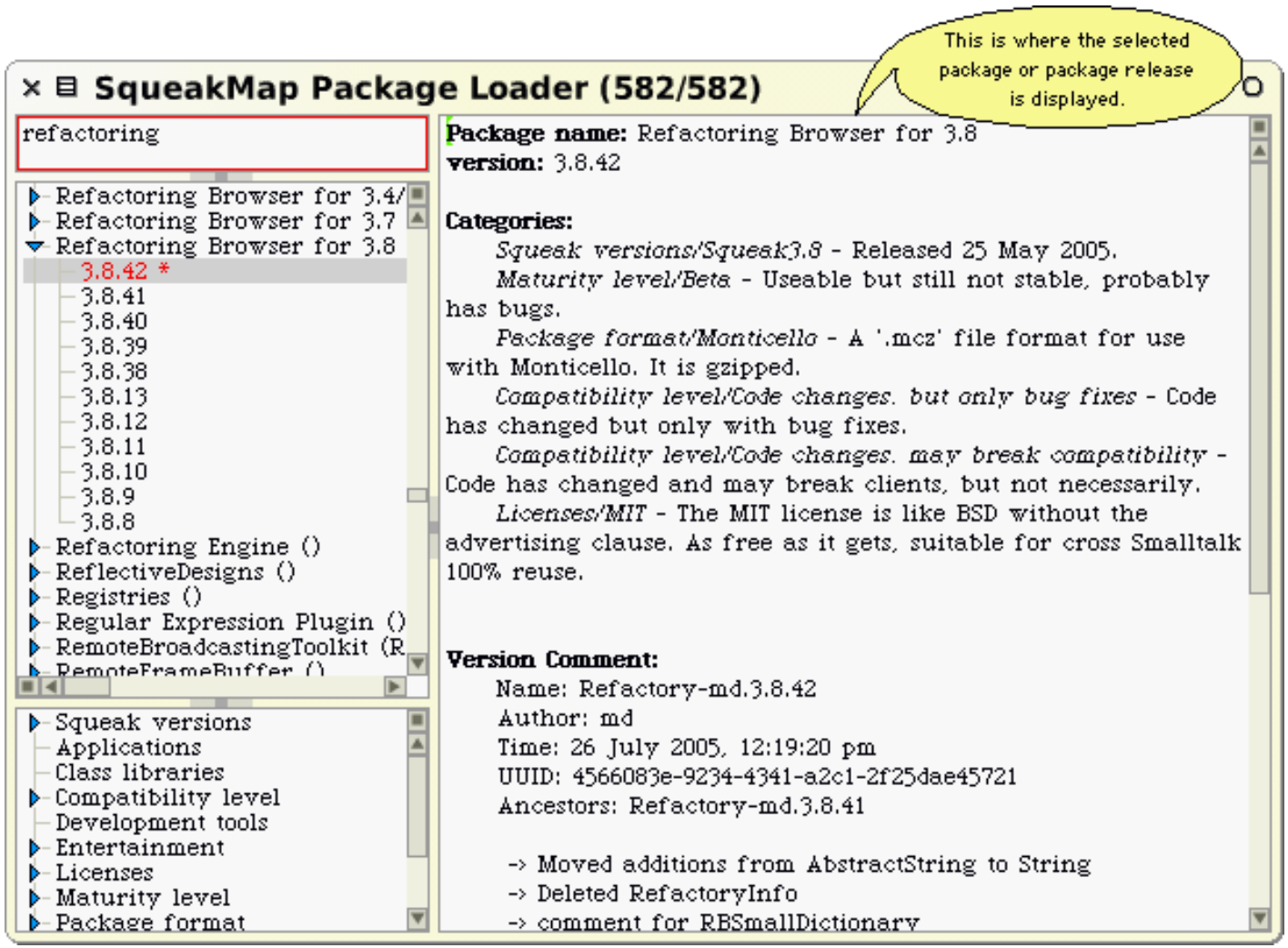
The code browsers

The code browser – or simply the browser – is a central tool in Smalltalk development. It greatly helps the developer to write and navigate the classes and methods.



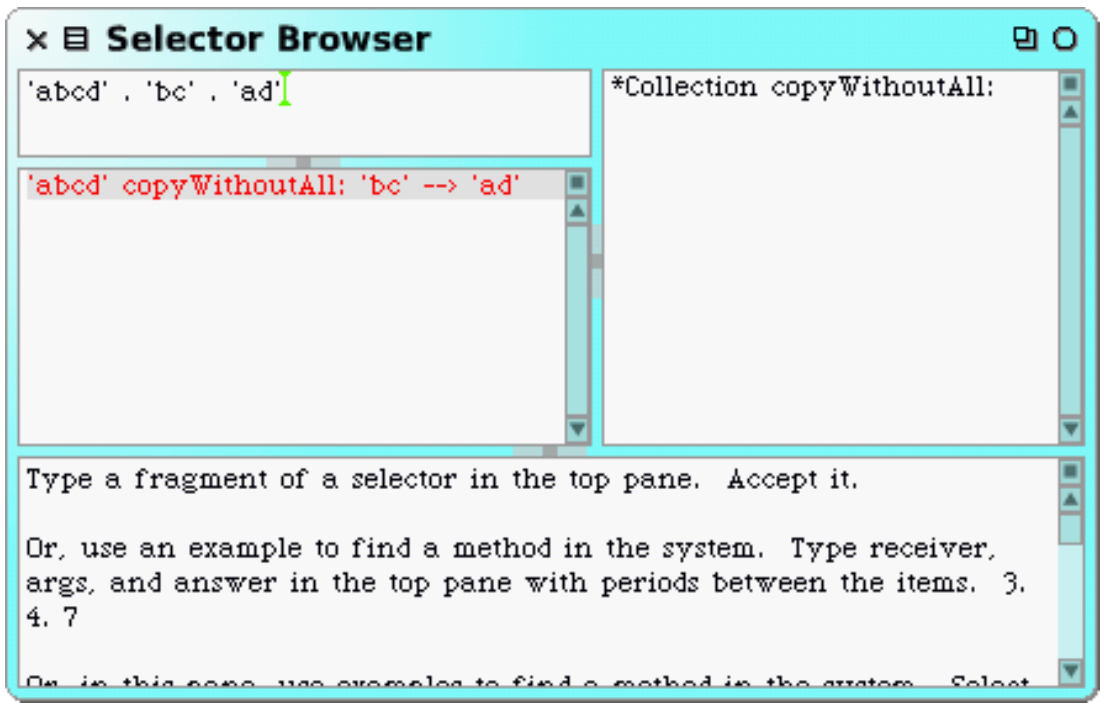
SqueakMap package loader

A tool to install remote components, libraries or applications:



The method finder

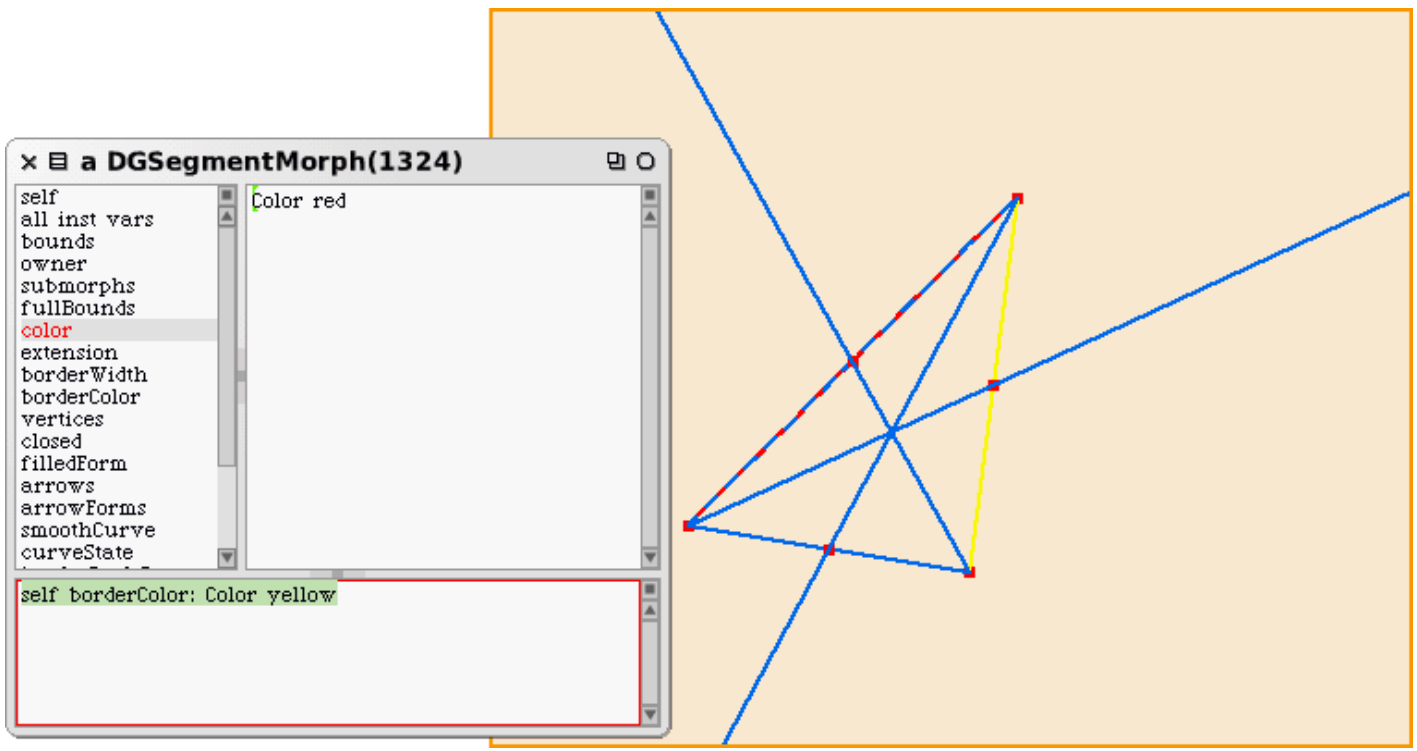
Squeak is able to inspect itself to search a method according to a pattern. Here with the pattern **'abcd' . 'bc' . 'ad'** we can find the string methods removing a substring from a string:



The inspector

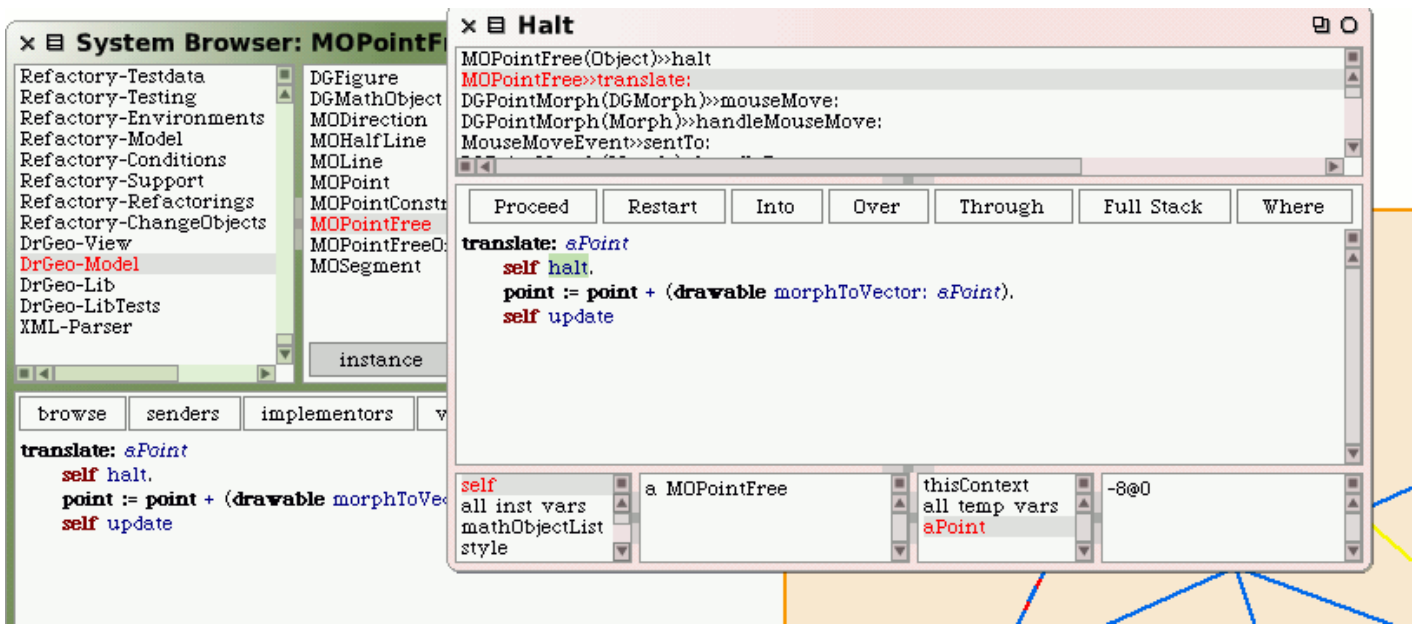
With the inspector, while your program is running, you can *very comfortably*:

1. inspect your classes and its attributes
2. change your classes and attributes value while your programme is running



The debugger

With the debugger you can debug your running code and also fix bugs and recompile *on the fly* the method. Then you resume your work. Your application does not need to be restarted. Bellow a halt point causes the debugger to open:

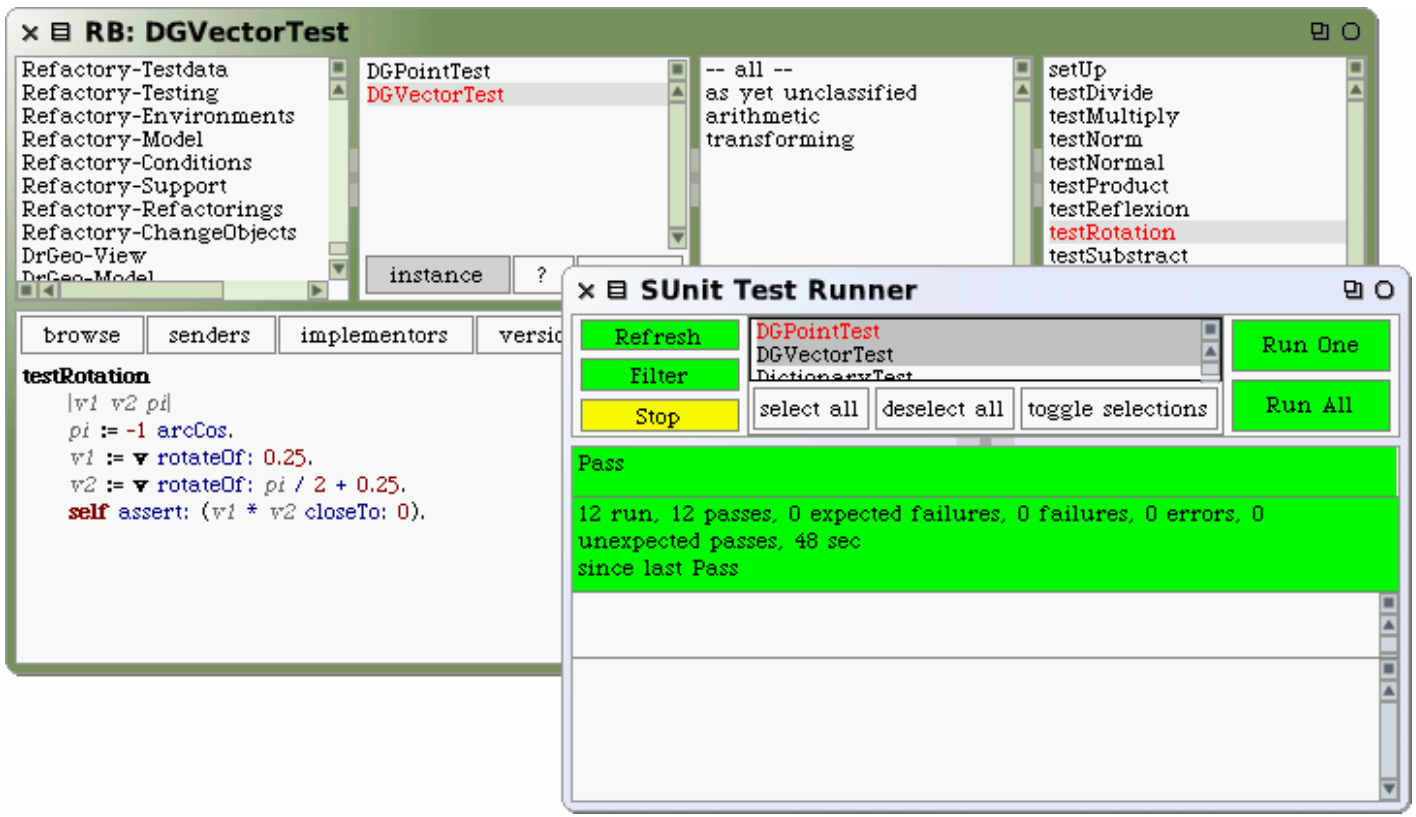


The refactoring browser – RB

- Refactoring code is frequent, it means moving, changing section of code. Developer tools can assist the programmer to refactor code safely.
- The Squeak RB extents the classic browser with code refactoring facilities. It helps – among other things – to safely rename instance variables and methods. It checks down the subclasses to rename the variable when necessary.

SUnit test runner

Unit tests help to automatically check validity of some piece of code (i.e. ensuring it effectively do the right computation)
Here a set of tests run over an application library:



The central tool

The browser is your central development tool:

- to write new system categories and define classes, methods
- to explore the hierarchies of the classes
- to explore implementors and senders of methods
- to explore the inheritance of your methods

The central tool

The browser is your central development tool:

- to write new system categories and define classes, methods
- to explore the hierarchies of the classes
- to explore implementors and senders of methods
- to explore the inheritance of your methods

Let's look at a live example...

Incremental programming

Incremental programming is about smooth step by step programming:

- Open a **workspace** to experiment your *pieces* of codes
- The **debugger** will open when necessary (faulty code or breakpoint), you will fix the code and *continue* your experiment
- With the **inspector** examine the attribute values of your classes

Incremental programming

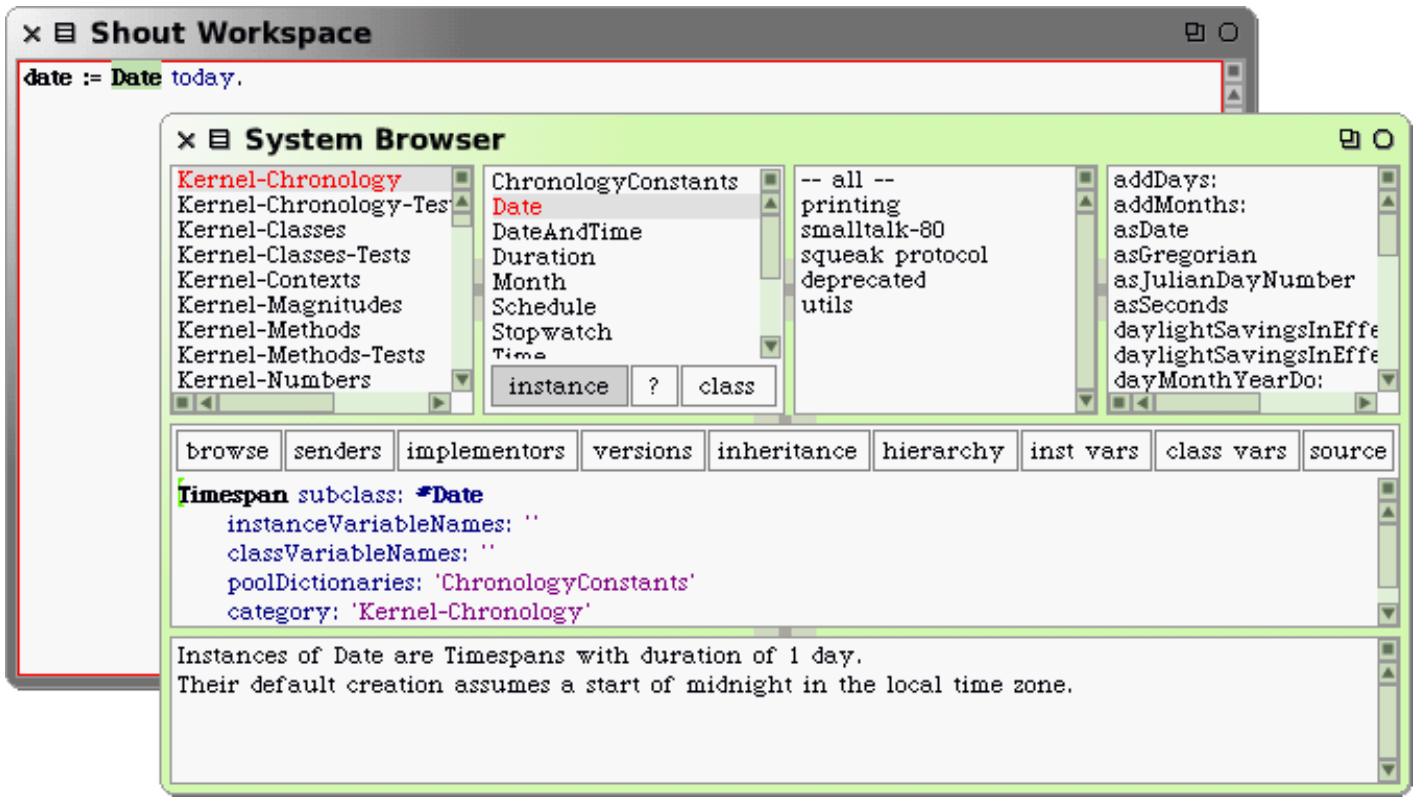
Incremental programming is about smooth step by step programming:

- Open a **workspace** to experiment your *pieces* of codes
- The **debugger** will open when necessary (faulty code or breakpoint), you will fix the code and *continue* your experiment
- With the **inspector** examine the attribute values of your classes

Let's look at a live example...

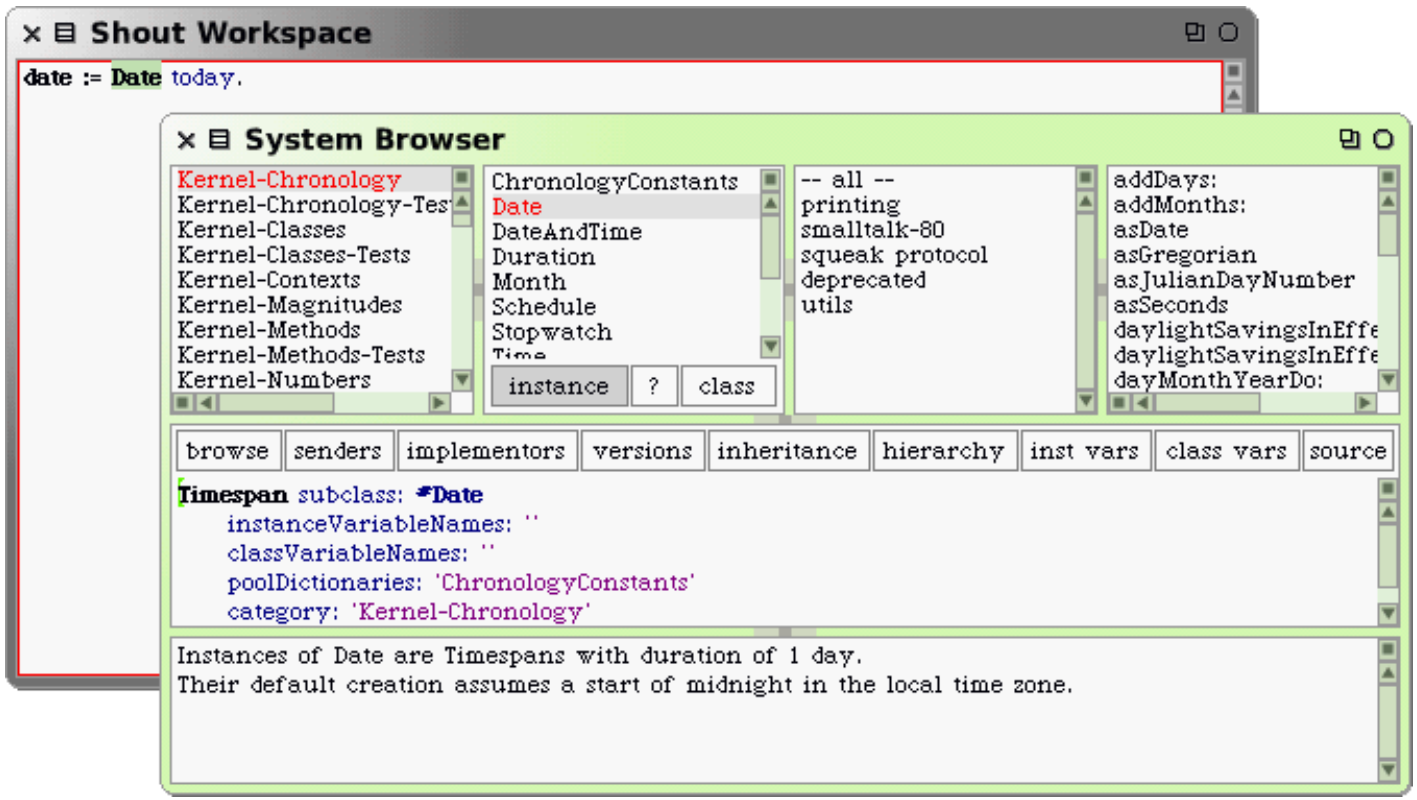
Learning from the class library

When you want to use an object but you don't know how, just use the browser to explore the object and look at its methods. Here exploring the system class **Date**:



Learning from the class library

When you want to use an object but you don't know how, just use the browser to explore the object and look at its methods. Here exploring the system class **Date**:



Let's look at a live example...

Saving our code

Beside saving the whole image, it is easy to save the specific code we are working on with the **Monticello browser**. When writing code, take care to:

- Define the *Toto* application classes in a *Toto-** categories (i.e. *Toto-view*, *Toto-model*, change *Toto* with your application name)
- When you need to add methods to a system class, define in this class a **Toto-myCategory* and define your methods inside.
- Use the **Monticello browser** to save your code, on disk or remotely.

For more information read: `http://www.iam.unibe.ch/~scg/Teaching/AdvancedLabs/PDFs/squeaktools.pdf`

Saving our code

Beside saving the whole image, it is easy to save the specific code we are working on with the **Monticello browser**. When writing code, take care to:

- Define the *Toto* application classes in a *Toto-** categories (i.e. *Toto-view*, *Toto-model*, change *Toto* with your application name)
- When you need to add methods to a system class, define in this class a **Toto-myCategory* and define your methods inside.
- Use the **Monticello browser** to save your code, on disk or remotely.

For more information read: `http://www.iam.unibe.ch/~scg/Teaching/AdvancedLabs/PDFs/squeaktools.pdf`

Let's look at a live example...

What to remember?

- Squeak is a virtual machine – VM – based environment.
- The VM is fed with a multi-platform image file composing the environment.
- It is based – and written – on Smalltalk, a pure object oriented language.
- Squeak offers a large range of high level developer tools
- Squeak offers new paradigm in software development as incremental-compilation.

The future

- The Squeak foundation – `http://smallwiki.unibe.ch/SqueakFoundation` – a legal entity to support the Squeak promotion.
- Seaside – `http://www.seaside.st` – it is a frame work to develop high level web application within the Squeak environment. Basicly your are writting web application as you are developping event controled desktop application plus you take benefice of the Squeak IDE to develop and debugs. ASTONISHING!
- Tweak – `http://tweak.impara.de` – a rewrite of the graphic morphic interface.
- wxSqueak – `http://www.wxsqueak.org` – a wxWidget wrapper for Squeak to write multiplateform application with native look and fell.
- Croquet – `http://www.opencroquet.org` – a 3D peer-to-peer authoring environment.

Academic resources

- http://www.iam.unibe.ch/~ducasse/Teaching.html
- http://www.eli.sdsu.edu/courses
- http://prog.vub.ac.be/~tjdhondt/POOL/HTM.dir/introduction.htm
- http://wiki.cs.uiuc.edu/VisualWorks/VisualWorks+in+Education
- http://www.whysmalltalk.com/universities/

Other resources in the net

- `http://www.squeak.org`
- Free Smalltalk books:
  `http://www.iam.unibe.ch/~ducasse/FreeBooks.html`
- `http://www.whysmalltalk.com`