OpenDoc Series'

Shift to Dynamic

-Smalltalk for Java Programmer

V1.0

作者：Raimundox

文档说明

参与人员：

作者	联络
Raimundox	raimundox(at)gmail.com

(at) 为 email @ 符号

发布记录

版本	日期	作者	说明
0.3	2005.12.10	Raimundox	原创
0.3	2006.01.01	夏昕	文档格式编排

OpenDoc 版权说明

本文档版权归原作者所有。
在免费、且无任何附加条件的前提下，可在网络媒体中自由传播。

如需部分或者全文引用，请事先征求作者意见。

如果本文对您有些许帮助，表达谢意的最好方式，是将您发现的问题和文档改进意见及时反馈给作者。当然，倘若有时间和能力，能为技术群体无偿贡献自己的所学为最好的回馈。

Open Doc Series 目前包括以下几份文档：

- **Spring 开发指南**
- **Hibernate 开发指南**
- **ibatis2 开发指南**
- **Webwork2 开发指南**
- **持续集成实践之 CruiseControl**
- **Shift to Dynamic: Smalltalk for Java Programmer**

以上文档可从 http://www.redsaga.com 获取最新更新信息

Shift to Dynamic

-Smalltalk for Java Programmer

Shift to Dynamic

-Smalltalk for Java Programmer

An Introduction to Smalltalk

Why Smalltalk?

有这样一门语言：它的程序运行在虚拟机上；它内置垃圾收集机制可以自动管理内存；它可以以脚本或者编译的方式执行；它是动态强类型面向对象语言。它是什么语言？我想大多数人的第一反应是 Ruby 或者 Python，但是我所说的是一个更加古老的语言——Smalltalk。

在大多数人眼中 Smalltalk 是一个古老甚至落后的语言（对于我们所处的这个日新月异的 IT 界来说，古老往往与落后相伴）。我们为什么要学习 Smalltalk 呢？现在已经很少有新的项目要求使用 Smalltalk 作为开发环境（我甚至怀疑在中国是否曾经有过大的 Smalltalk 项目）；作为 Smalltalk 最大供应商的 IBM 也已经宣布将于 2006 年开始终止对 Smalltalk 相关产品的支持。为什么要耗费大量的机会成本在一个古老而且不大实用的语言上？

从我个人的角度来讲，有这样一些理由来学习和使用 Smalltalk：

- Ruby 等动态面向对象语言更多的继承了 Smalltalk 的精神

  在最近一段时间里，我逐渐开始更多地使用 Ruby 而不是 Java，我发现 Ruby 和 Java 可以说是两种完全不同的面向对象语言。好奇心的驱使下，我开始探究 Ruby 为什么会和 Java 有这么多的不同，最终我发现，Ruby 比 Java 更多地继承了 Smalltalk 的精神。当我对 Smalltalk 了解得越多，我使用 Ruby 就更加有把握和自信。正是 Smalltalk 使得 Ruby 这门对我这个 Java 程序员而言的外语逐步开始变成我的母语。

- Ruby 正在沿着 Smalltalk 曾经的道路发展着

  在我最开始使用 Ruby 的一段时间里，重构这件对于我而言再平常不过的事情，在 Ruby 中令我感到无从下手，进而我甚至怀疑 Ruby 能否应用于大型项目而不仅仅是一些简单得脚本工作。这个问题最终我是借鉴了 Smalltalk 中的 Refactoring Browser 来理解 Ruby 中的重构的（不久在 ThoughtBlog 上(http://blogs.thoughtworks.com)也看到了类似的讨论）。而后我越来越觉得 Ruby 是沿着 Smalltalk 的方向继续发展着，这些困扰着我的问题都曾经困扰着那些 Smalltalker 们，“以史为鉴可以知兴替”啊。

- Smalltalk 具有一些其它面向对象语言所不具备的特征

  上面两个观点其实略显牵强，听起来就是说："为了学习英语我们必须先学习拉丁文，然后才能更好的学习英语"。不过我也是没有办法才出此下策，谁让 Ruby 最近很热呢，只好借它的名号来号召大家一下。但是就 Smalltalk 本身而言，它具有一些其它面向对象语言所不具有的特点，比如一个完全开放的对象系统；比如非常轻语法，可以很容易的实现 DSL（Domain Specified Language）；比如更加强大且成熟 MOP 系统（Meta-Object Protocol）等等。这些优秀且迷人的特性是在其它语言中看不到的。

- 使用 Smalltalk 是一种奇妙而有趣的体验

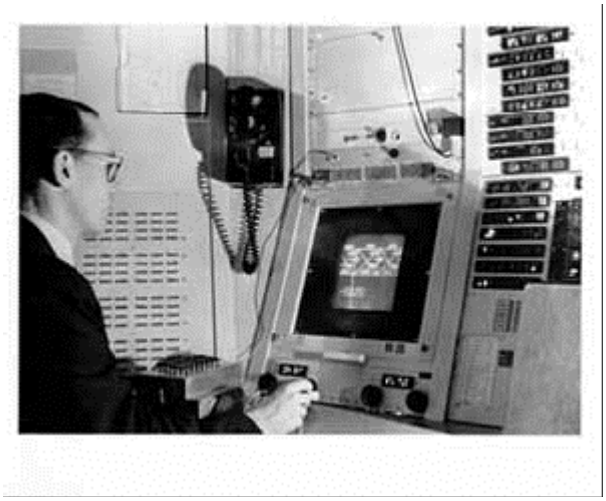
  作为一个写了 14 年程序的老程序员，Smalltalk 是我用过的最奇妙最有趣味性的语言，对于一个程序员来说，这一条理由就已经足够了，不是吗？

OK， 接下来我将与大家分享我——一个 Java 程序员——的 Smalltalk 体验。

A Little History

Smalltalk 产生的历史从某种程度上来说就是面向对象技术产生的历史，其中有两个东西对 Smalltalk 产生了巨大的影响。

第一个是 Ivan Suther 于 1963 在 MIT 开发的一个名为 Sketchpad 的图形编辑器，这个图形编辑器不仅仅通过单独的像素来编辑图形的，它创造了一种名叫"Master Drawing"的东西，可以根据已经做好的"Master Drawing"来创造一系列的"Instance Drawing"，之后一旦"Master Drawing"发生了变化，所有的"Instance Drawing"也可以跟着发生变化。下图所示为 Ivan Suther 在使用 Sketchpad（取自 http://www.sun.com/960710/feature3/sketchpad.html）

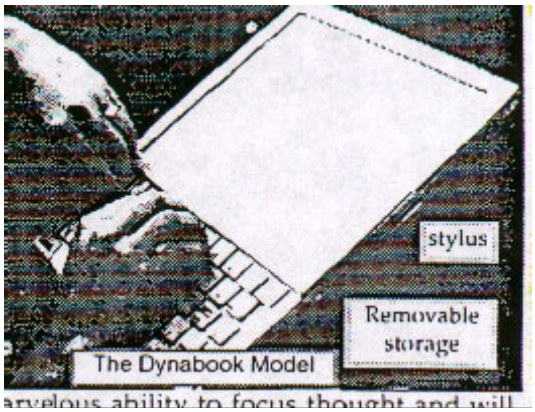


第二个自然就是被认为是所有面向对象语言的共同祖先的 Simula 语言。Simula 语言用来实现一些过程的模拟，它通过定义"Activity"来表述每一个被模拟的活动，根据"Activity"又可以创建一系列不同版本的"Process"。虽然 Simula 被认为是一个通用的过

程性语言（也有一部分人认为它就是一个完整的面向对象语言，但是我想将这个殊荣留给Smalltalk☺），但是事实上每一个“Process”都是一个对象，它有自己独立的数据和行为，不受其他的对象的影响。

1966 年，Alan Kay 还是一位犹他大学计算机系的研究生，他的导师给他布置一个任务，让他在一台计算上把 Simula 语言跑起来，与此同时他读到了关于 Sketchpad 的一些东西。他马上感觉到了这二者的相似，他也隐隐感到这种全新的构造软件的方式将会成为构造大型复杂系统得关键。

之后 Alan Kay 在他的论文中提出了一个构想，既然 Sketchpad 是一个面向对象的绘图系统，而 Simula 在程序设计领域中引入了面向对象的一些概念，那么能不能够造一台完全基于对象的可编程的计算机呢？Alan Kay 将这个计算机命名为 FLEX。FLEX 被认为是对于面向对象编程系统的第一次尝试，不过它的意义远不止于此，FLEX 被设计为一台个人电脑（要知道在 60 年代计算机都是占满一个屋子的庞然大物，Alan Kay 已经开始为个人设计电脑了），同时在了解到 Logo 语言的一些情况之后（呵呵，LOGO 是我学会的第二个编程语言，那时我 11 岁），Alan Kay 将个人计算机定位为个人动态媒体（Personal Dynamic Media），进而 Alan Kay 还提出，个人电脑应该是小巧得，支持键盘或者笔作输入装置，有无线网络以及 NLS（oNLine System），然后他给这个东西起了名字：Dynabook。想想看这样描述和我们现在所用的笔记本电脑是何其相似！多么伟大的洞察力啊！Alan Kay 在 30 年前开发出的 Smalltalk 至今仍具有很多先进的特性也就不足为奇了。下图是 Dynabook 的示意图（看看图中写的 Removable Storage 吧，想起什么？U 盘...）



而作为Dynabook的操作系统和编程系统自然就是Smalltalk对象系统。Smalltalk的开发历时8年经过了很多个迭代，其中Smalltalk 71和Logo非常的相似，Smalltalk 72则引入了很多针对媒体、音乐和图标的语言特性。直到Smalltalk 76才成为一个纯粹的通用面向对象设计语言，而Smalltalk 80则是Xerox第一个公开发布的Smalltalk版本。

早期的Smalltalk是按照Alan Kay的“Personal Computer for Children of All Ages”的设想来设计的，因此语法上尽可能的贴近英语，同时更多的依赖于图形而非文本进行程序设计，这一特点在当代Smalltalk语言里仍然保留了很浓厚的印记。

颇为讽刺的是，以个人电脑为受众而设计的Smalltalk对象系统在第一个真正意义上的PC平台上反而并不流行，当然这主要是受限于IBM PC和MS DOS孱弱的性能，而据说Apple在接受Xerox邀请实现Smalltalk的时候，从中借鉴了大量想法最终产生了Lisa 和Macintosh，对比同时期的Mac和PC的性能，可以看出Smalltalk对于个人电脑的性能要求已经超越了它所处的时代。

1990年，Xerox筹建的Parc Place Systems出品了一个新版本的 Smalltalk，就是ObjectWorks 和以后的 VisualWorks，它提供了可以使用了“本地”窗口系统的机制（其实也就是类似于今日的AWT和SWT了），大大的提高了整个图形系统的性能，也使得Smalltalk摆脱了PC机上丑陋而简单的界面。而后IBM发布了Visual Age for Smalltalk，就是Eclipse的前身了，也逐步成为Smalltalk的主要供应商之一。

Smalltalk的商用系统是昂贵的，1996年以后逐渐出现了一些开源或者免费的Smalltalk实现，其中 Squeak 格外的突出。它在一小组研究者活跃的开发下，包括Smalltalk 的最初建立者 Alan Kay 和 Dan Ingalls。很多人说在许多方面上 Squeak 代表了对最初的 Smalltalk 计划的价值的一个回归。Squeak 可免费获得当前的实现，存在于 Macintosh 、Windows PC 和 Unix 系统上。还有一个 Squeak 邮件列表让它的用户社区及时了解最新的发展。我将以Squeak为环境介绍Smalltalk语言。最新的Squeak可以从www.squeak.org上下载。

What’s Smalltalk?

Smalltalk这个名字通常被用来指代三个截然不同但又紧密联系的事物：一个编程语言，一套类库以及一个开发环境。

Smalltalk = a Language + a Class Library + a Development Environment

Smalltalk语言本身是非常非常小的，仅有5个左右的关键字（Java有40个左右）。通过这几个关键字，你可以定义变量；将某对象赋为某个变量的值以及向对某个象发送消息。Smalltalk中几乎所有东西都是对象，几乎所有工作都可以通过向特定对象发送消息来完成。这样的语法对于Smalltalk而言已经足够了。

Smalltalk并没有在语言中定义任何的对象或者消息，而是将这些通用的对象和消息是定义在类库中的，包括数学运算、逻辑运算甚至分支、循环等常用程序结构在Smalltalk里也是在通过类库中的对象来实现的，类库才是Smalltalk系统的核心。

所有Smalltalk实现都会提供一套类库，但是除了标准类库相似之外，不同供应商提供的类库也有所不同。比如Squeak除了基本类库之外，还提供了3D应用库、多媒体应用库等等。这些类库和你自己的程序一样也是由Smalltalk写成的，Smalltalk并不会因为这些是标准类库那些是你写的程序而区别对待。Smalltalk提供了类库的全部代码，甚至IDE以及编译器的代码。你可以按照自己的需要来修改这些类库。

Smalltalk实现还提供了一个开发环境，在这个环境中你可以非常容易的查找和修改类库的代码。Smalltalk实现中的开发环境可能会有非常大的差异，不过大多数Smalltalk实现都提供了相似的基础开发环境，比如System Browser，Workspace等。

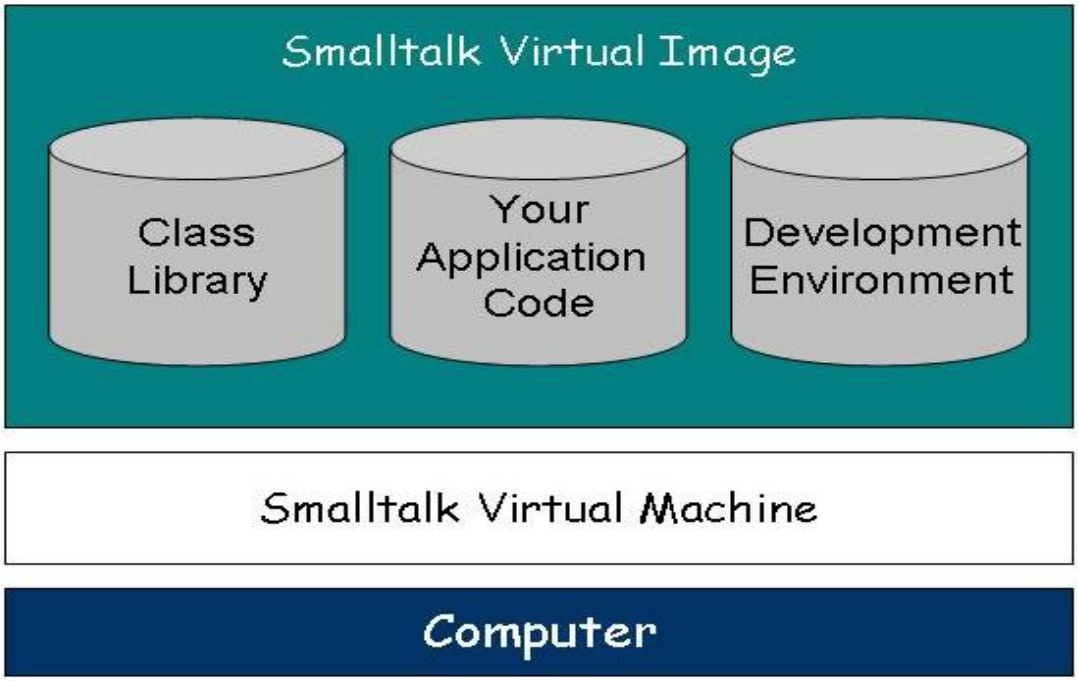
由于Smalltalk的这些特性，掌握Smalltalk不仅仅意味着掌握Smalltalk语言（这是非常容易的），还意味着掌握Smalltalk的类库和开发环境，因此在开始一个Hello World例子之前，需要花一些时间来了解Smalltalk的开发环境。

Java VM vs Smalltalk VM

Java 的程序运行在虚拟机（JVM）上，Java 代码会被编译为虚拟机可以理解的 bytecode，然后虚拟机再来解释这些编译好的 bytecode。JVM 表现为一个可执行文件，这个文件在 Windows 平台下就是 java.exe。而 Java 的基本库保存在 JRE\lib\rt.jar 这

个 JAR 包中，我们自己的程序保存在相应的.java 文件或者 JAR 文件里。我们可以通过 –cp 和 -jar 参数来指定 JVM 的搜索路径，然后根据相应的需要把文件或者 JAR 包中的.class 文件读入内存。

Smalltalk 的 VM 也是一个可执行文件，这个文件根据 Smalltalk 的实现不同而有所不同，如果是 Windows 平台下的 Squeak，就是 SQUEAK\Squeak.exe。Smalltalk 除了有 VM 的概念之外还有一个称作 Virtual Image（下简称 VI）的东西，VI 可以想象成 VM 的内存或者主存储器。如下图所示：



Smalltalk 的 VM 在启动的时候会读入相应的 VI 文件，VI 文件包括基础的 Smalltalk 类库和 Smalltalk 实现提供的开发环境，Smalltalk VI 中所有的东西都是可编程的。如下图所示，为 Smalltalk VI 文件构成：

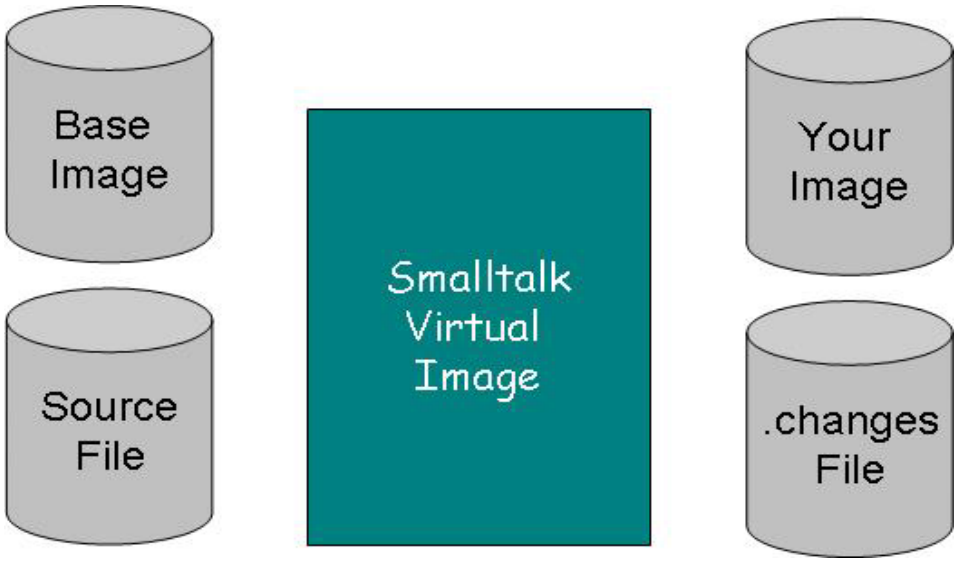
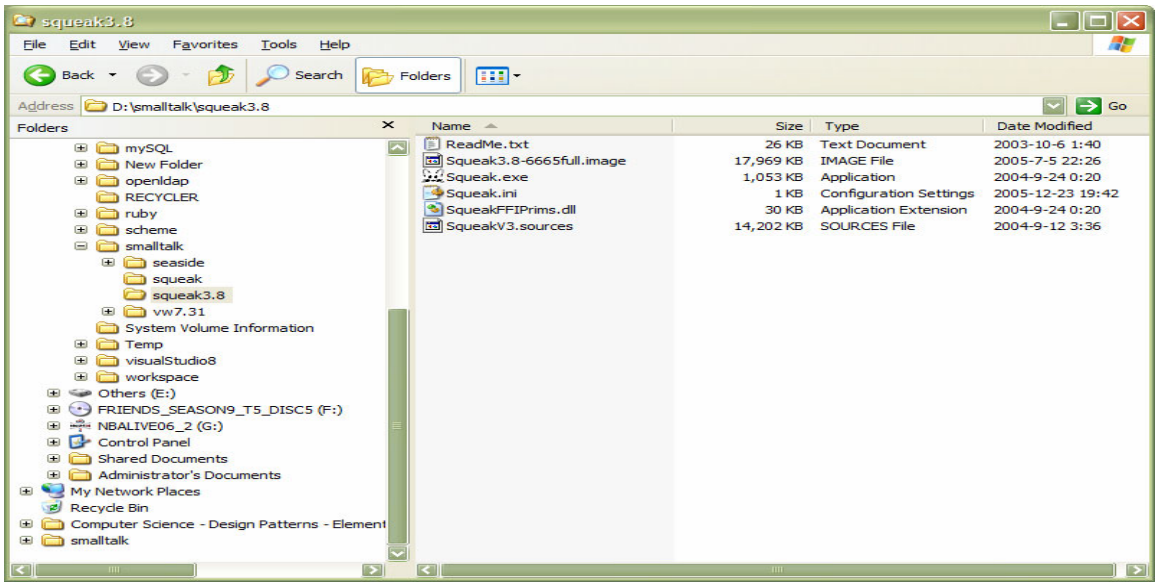


Image 文件可以看作某一个时刻 Smalltalk 虚拟机主存的一个快照，一般由两个文件组成：.image 文件代表了某一块主存的基本状态；.changes 文件代表了这个主存块的发生过的所有的修改。通过这两个文件，VI 就可以准确表示主存一个快照了。

由于 VI 的引入使得 Smalltalk VM 和 JVM 有很大的不同，JVM 每次在启动的时候，都会重新申请和分配一块内存区域（具体大小可由参数指定），然后将路径上的类按需读入内存。而 Smalltalk VM 每次启动的时候都会把 VI 文件读入，内存中的对象和上次 VM 关闭的时候是一样的，Smalltalk VM 是以一种持续的观点来看待 VI 的，不仅所有读入的 class 会存在于 VI 中，所有未被垃圾收集的 Instance 也会保存在 VI 里。从某种意义上讲，Smalltalk VI 本身类似一个 OODBMS。

下图是 Squeak 3.8 的文件布局：

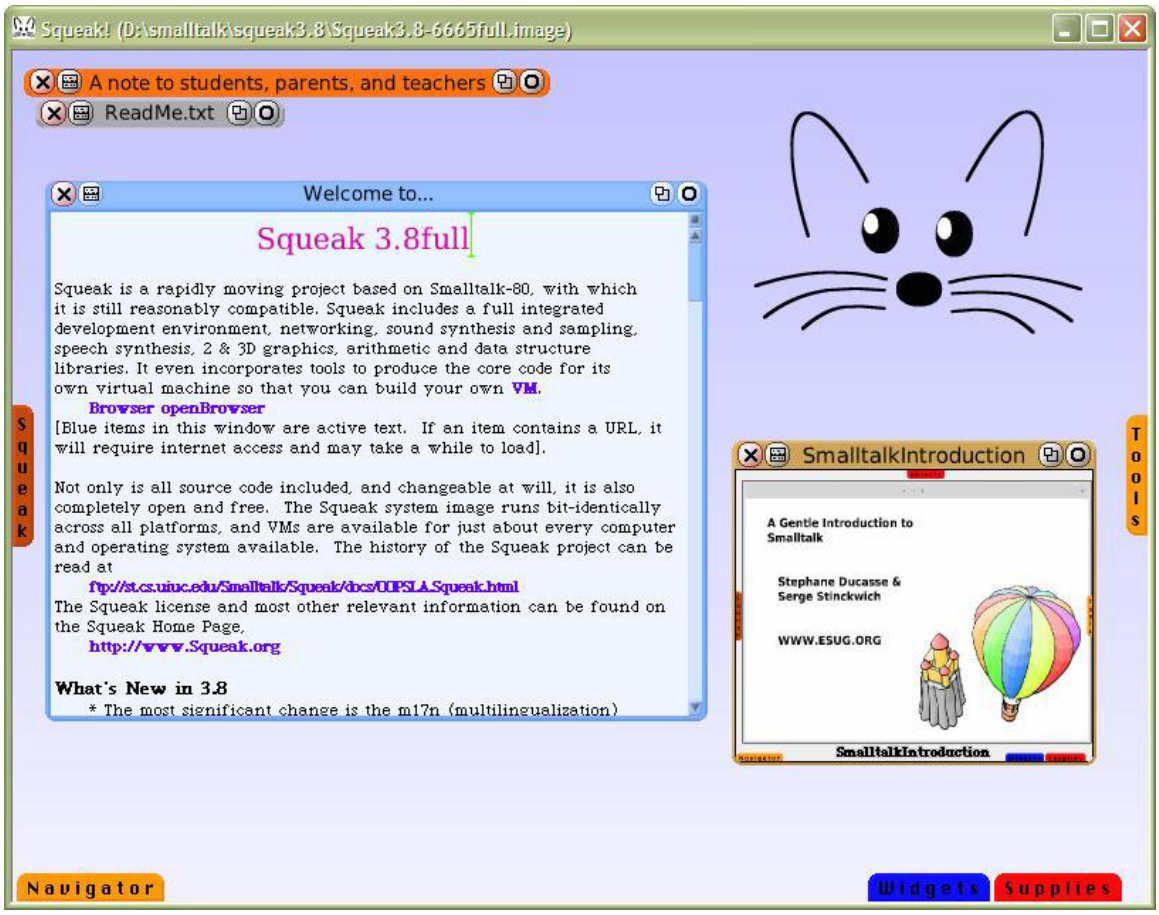


根据上面的介绍，Squeak3.8.-6665full.image 就是整个 Squeak 的 Base Image，SqueakV3.sources 代表了 Squeak 的全部源代码，由于这是一个全新安装的 Squeak 系

统，我还没有启动过，所以还没有.changes 文件。

Play with Squeak Environment

下面我们来启动 Squeak。



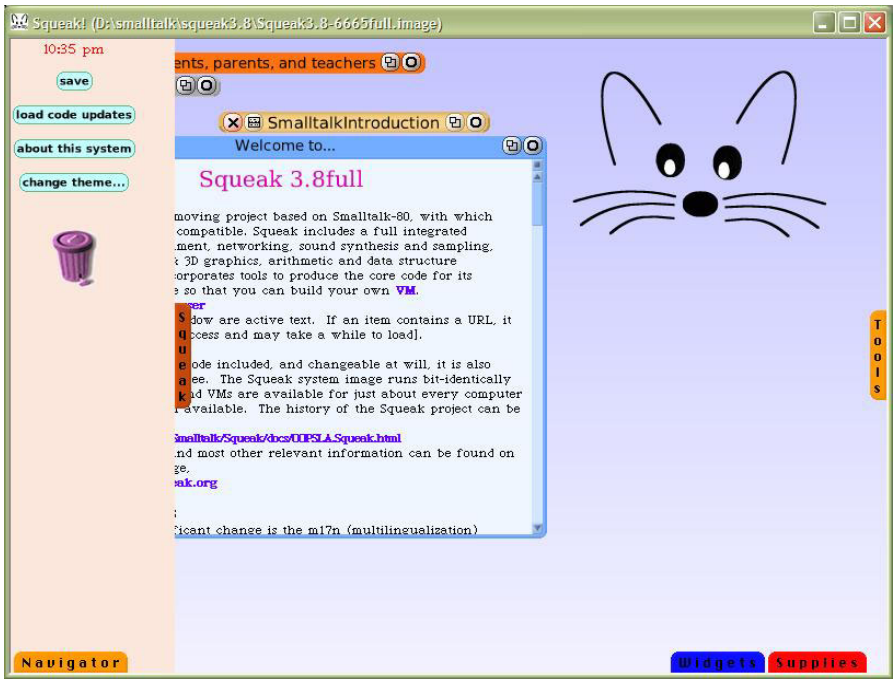
这个就是 Squeak，它不仅仅是一个 Smalltalk VM 还是一个 Smalltalk IDE。不过它看上去可能不大像一个 IDE，而更像一个操作系统的桌面。 这个类似桌面的东西称作 World，World 中的所有东西都是内存中的对象，而我们可以在 Squeak 中编程来操纵 World 中的对象。这和通常的 Java IDE 有很大的不同，Squeak（以及大多数 Smalltalk IDE）是开发环境的同时还是 VM 的管理终端，通过 GUI 或编程来控制 VM 中的对象。

下面我们做一些简单的试验，来看一下如何在 Smalltalk IDE 中操作 VM 里的对象。

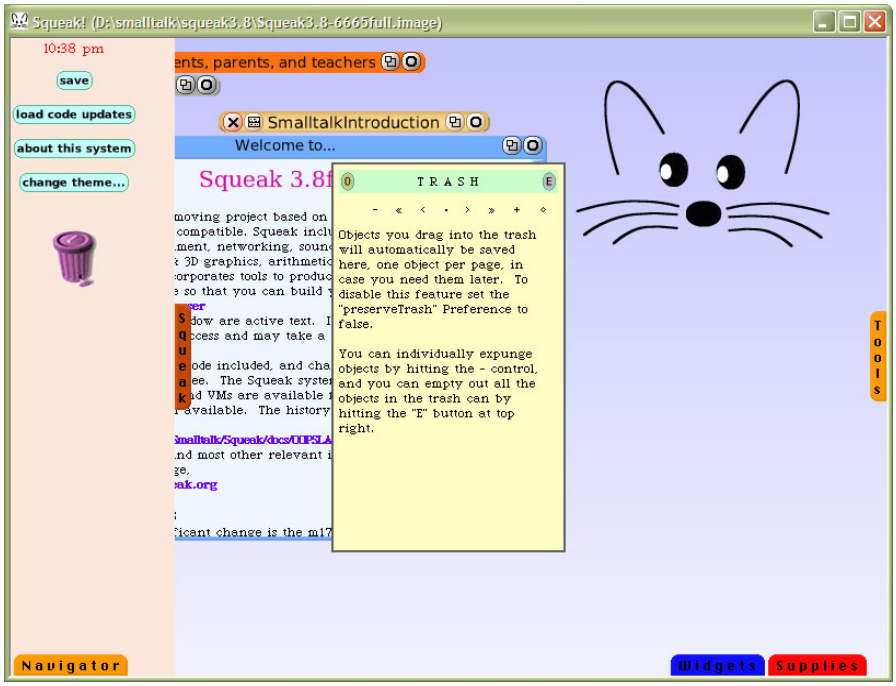
Deleting the SqueakLogo

看到那个最有玩具气质的大耗子了吧？如果你童心未泯可以留着，不过我看它很不爽，把他删掉！

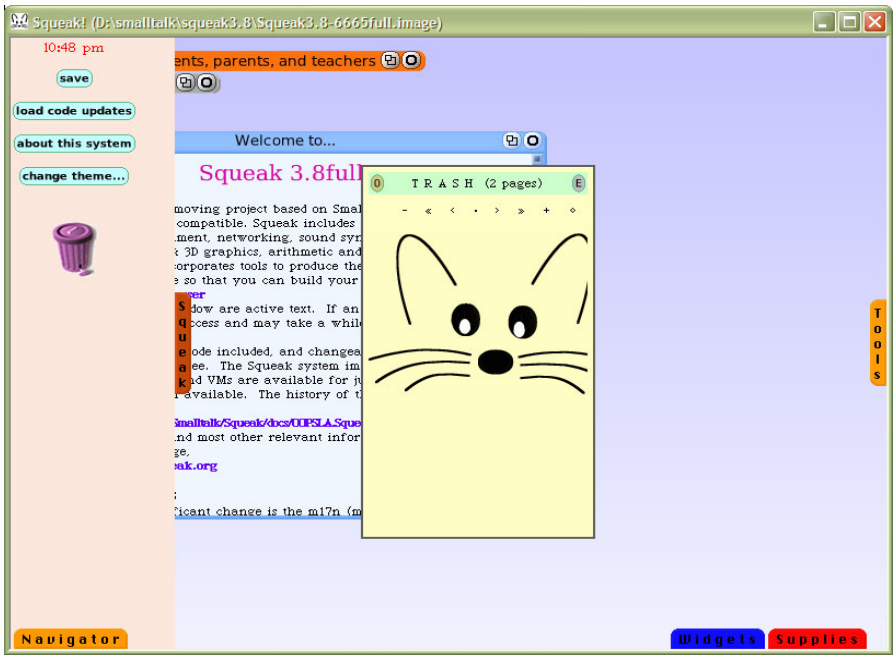
首先，点击屏幕左手的 Squeak 标签，会出现一个 Panel，上面有一个垃圾桶。



然后双击打开这个垃圾桶，Squeak 里的垃圾桶时分 Page 的，每一个页可以放很多不同的废弃的对象，然后可以把这些对象一起消灭掉。



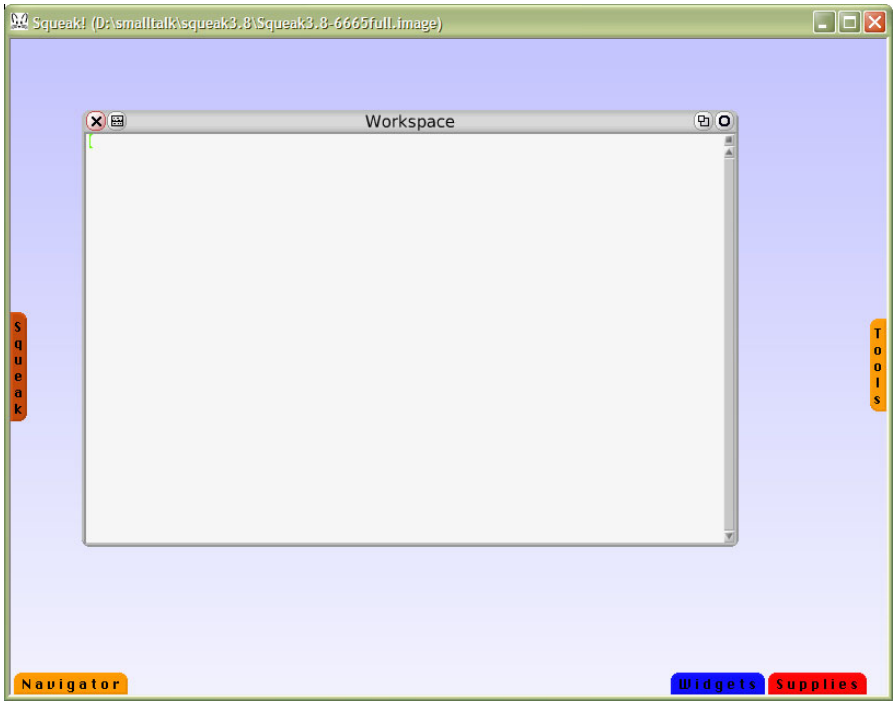
我们点击一下那个加号，就可以创建一个新的页面。然后把这个该死的大耗子垃进去。看，这样这个大耗子就在垃圾桶里了，它眼睛还一转一转的一幅无辜的样子。



点击垃圾桶又上角的椭圆里的 E，就把整个垃圾箱清空了。然后关上垃圾桶，整个世界清静了....

Adding a new buddy —— Slug

现在又觉得有一些空旷，那么我们再增加一些东西到 World 里吧，不要让他太空旷了。我们点开 Tools，会看到 Tools Panel，这个 Panel 上的东西都是与开发相关的。我们把 Workspace 拖到 World 中。



Workspace 可以不确切的认为是一个脚本解释环境。Squeak 的用户界面体系叫做 Morphic,Morphic 原本是 Sun 的一个 Smalltalk 方言 Self 的 GUI 系统,后来被 Squeak 吸收过来了。

现在我们要在 GUI 上增加一个东西，对于 Smalltalk 这个纯对象系统而言，就是创建一个新的对象，然后把这个对象添加到 World 中。

创建对象在 Java 中一般是使用 new 关键字。

```
SomeBuddy buddy = new SomeBuddy();
```

而在 Smalltalk 中情况略有不同，我们会用这样写：

```
aBuddy := SomeBuddy new.
```

看上去似乎只是一个语法的变化，但是实际上还有些微妙的不同：

1. Smalltalk 中的变量不用指定类型的
   这个并不意味着 Smalltalk 中没有类型，或者 Smalltalk 是一个弱类型的语言。严格来讲 Smalltalk 中没有变量，有的只是符号（Symbol)。上述表达式意味着，符号 aBuddy 代表了一个对象，这个对象是 SomeBuddy new 的返回值。只有符号而无变量，意味着 Smalltalk 中是只传引用而不传值，这一点和 Java 比较类似，Java 里除了基本类型也是只传引用的。

2. new 是一个消息而不是一个关键字
   在 Java 里 new 是一个关键字，而在 Smalltalk 中的只是一个消息的名字。我们可以用 SomeBuddy new.也可以用 SomeBuddy create.来表示这个创建的过

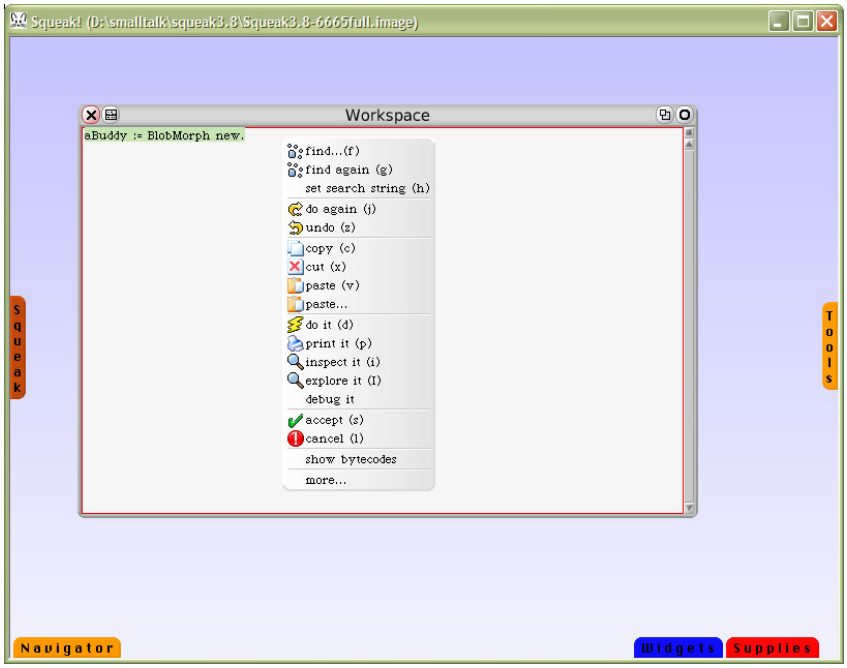
程。

现在我选择一只鼻涕虫来代替那支大耗子，这个鼻涕虫的实现类叫做 BlobMorph。当然这个名字已经告诉我们了，人家是个水滴不是鼻涕虫，不过它看起来还是很像鼻涕虫。在 Worksapce 输入如下的代码：

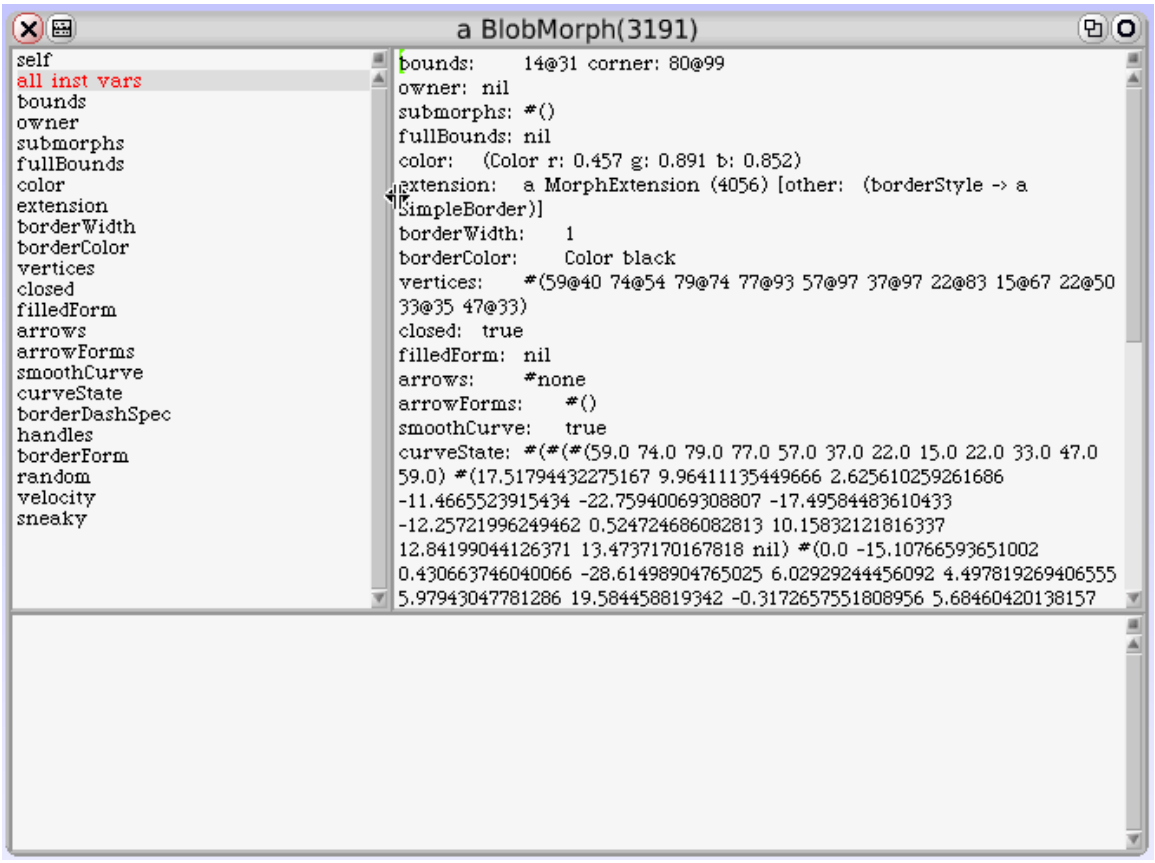
```
aBuddy := BlobMorph new.
```

输入完成之后，我们按 Alt + A 选中这些代码。在 Windows 平台下我们已经习惯了用 Ctrl + A 表示全选，而 Squeak 中用了 Alt + A，这可能是为了和 Mac OS 上保持一致吧。这个快捷键在 MacOS 上是 Apple + A，Apple 键相当于 Windows 上的 Alt。

之后我们单击鼠标右键，会看到一个菜单，如下图所示：

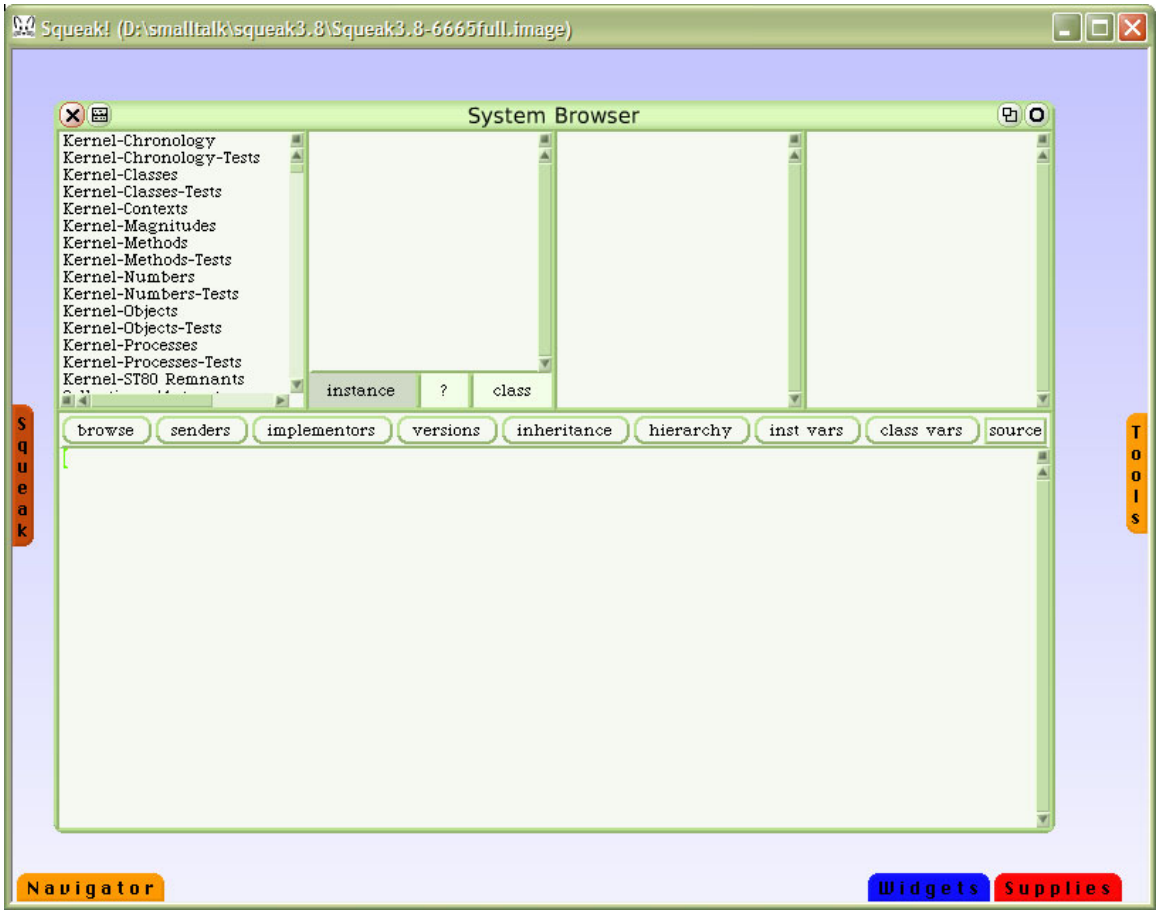


选择其中的 do it（也可以使用快捷键 Alt + D)。执行之后并没有任何的结果，但是这个时候 aBuddy 已经代表了一个构造好的对象了。为了证明这一点，我们选择 aBuddy，单击鼠标右键弹出菜单，选择 inspect it（快捷键 Alt + I)。会看到一个如下的窗口：

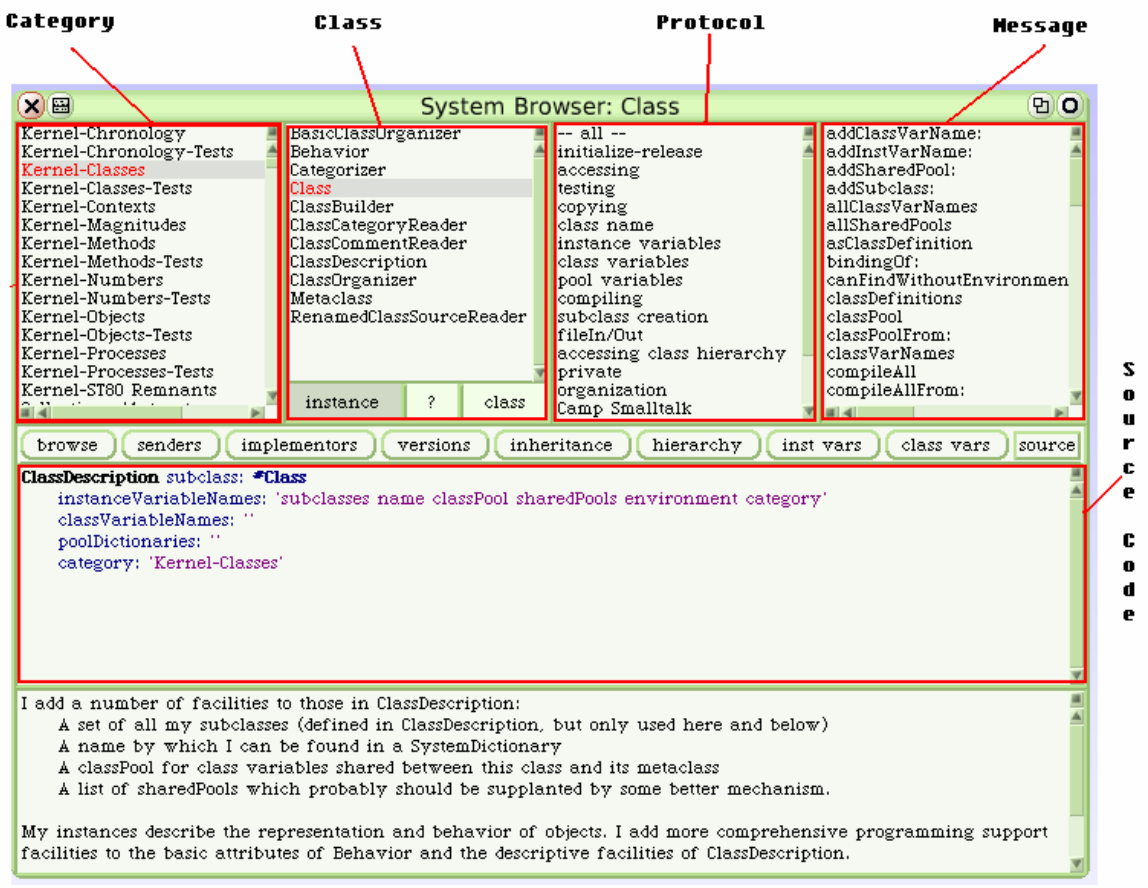


这个窗口里显示的内容就是 aBuddy 这个对象的所有实例级变量的值，我们可以通过这个窗口看观察对象的内容。它的作用类似于 Eclipse 中的 Variable View，是 Smalltalk 中 Debug 的基本手段。不过与 Variable View 不同的是，我们可以在 Inspection 中修改对象的值，并向这个对象对象发送消息。这个稍后再表。

现在我们已经知道了这个类已经创建了一个实例了，但是它还不能显示在 World 中，我们需要给这个对象发一个消息，让他把自己显示在 World 中。那么我们就要知道该发送什么消息给这个对象，首先打开 Tools Panel，从中选择 Browser，然后拖到 World 中。如下图所示：



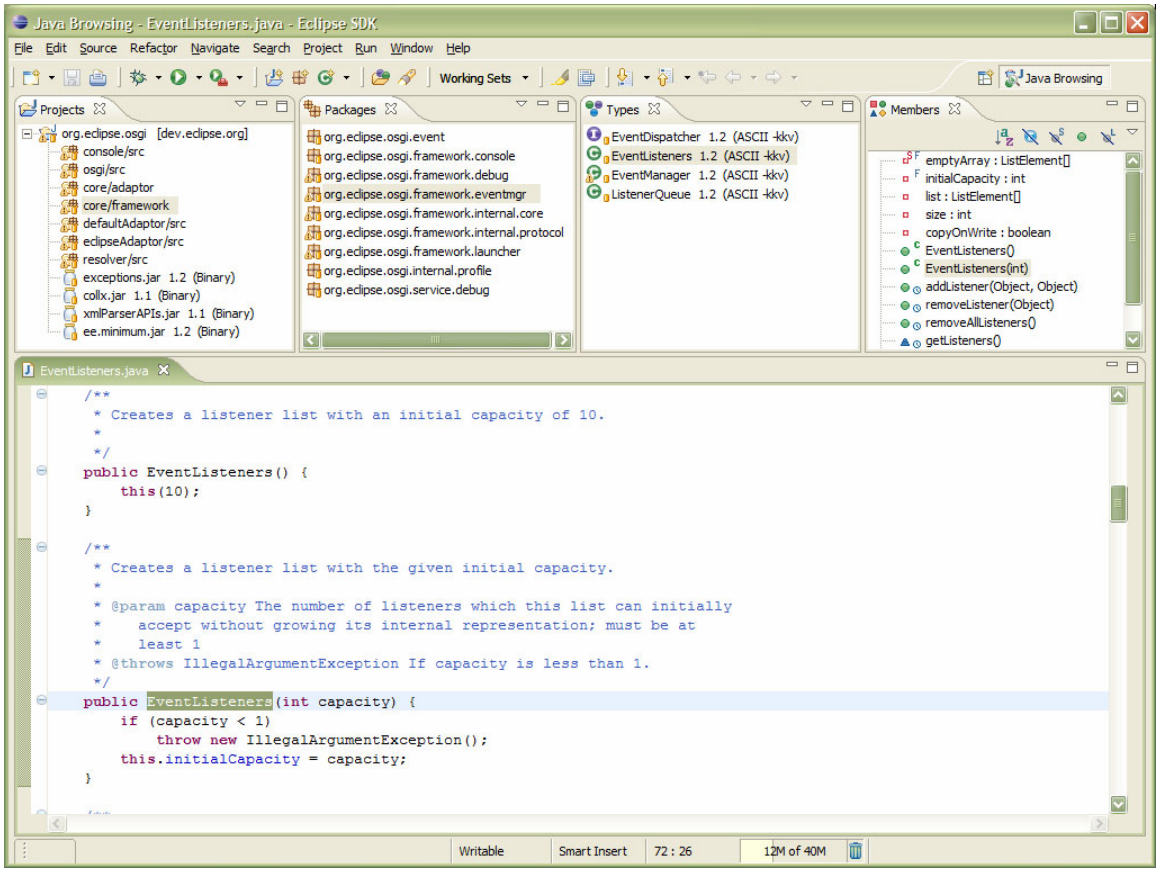
这个窗口是 Smalltalk 开发中最重要的窗口之一称作 System Browser，它可以用来浏览类库中的所有类以及他们所有的消息。同可以通过它向类库中增加新的类，修改原有类，以及删除一些类或类的消息。上面四个小窗口分别负责显示：Category, Class, Protocol, Message。如下图所示：



Java 程序员可能下意识将 Category 等同于 Java 里的 pacakge,但事实不是这样的，Category 仅仅是为了便于理解而将类分散到一些 Category 中， Smalltalk 中类名要求全局唯一，我们不能通过 Category 来避免名字冲突，Category 不具有任何的名字空间含义。因此在 Smalltalk 中类名尽可能的明确是非常重要的。

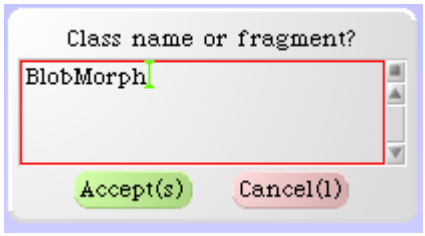
Protocol 对于 Java 程序员来说也是一个比较陌生的概念，这里 Protocol 也是仅仅代表一种对 Message 的分类，比如 initialize-release protocol 中，可以放置一些跟初始化和对象销毁相关的消息。而 testing protocol 里可以放置一些判断方法。Protocol 和 Category 一样，也没有名字空间的含义，仅仅是为了易于人类阅读和理解而设置的。

对于 Java 程序员，这个界面本身可能就有些许陌生和奇怪，但是如果你使用 Eclipse 的话，你可以打开 Eclipse，然后将 Perspective 选择为 Java Browsing，会看到如下界面：

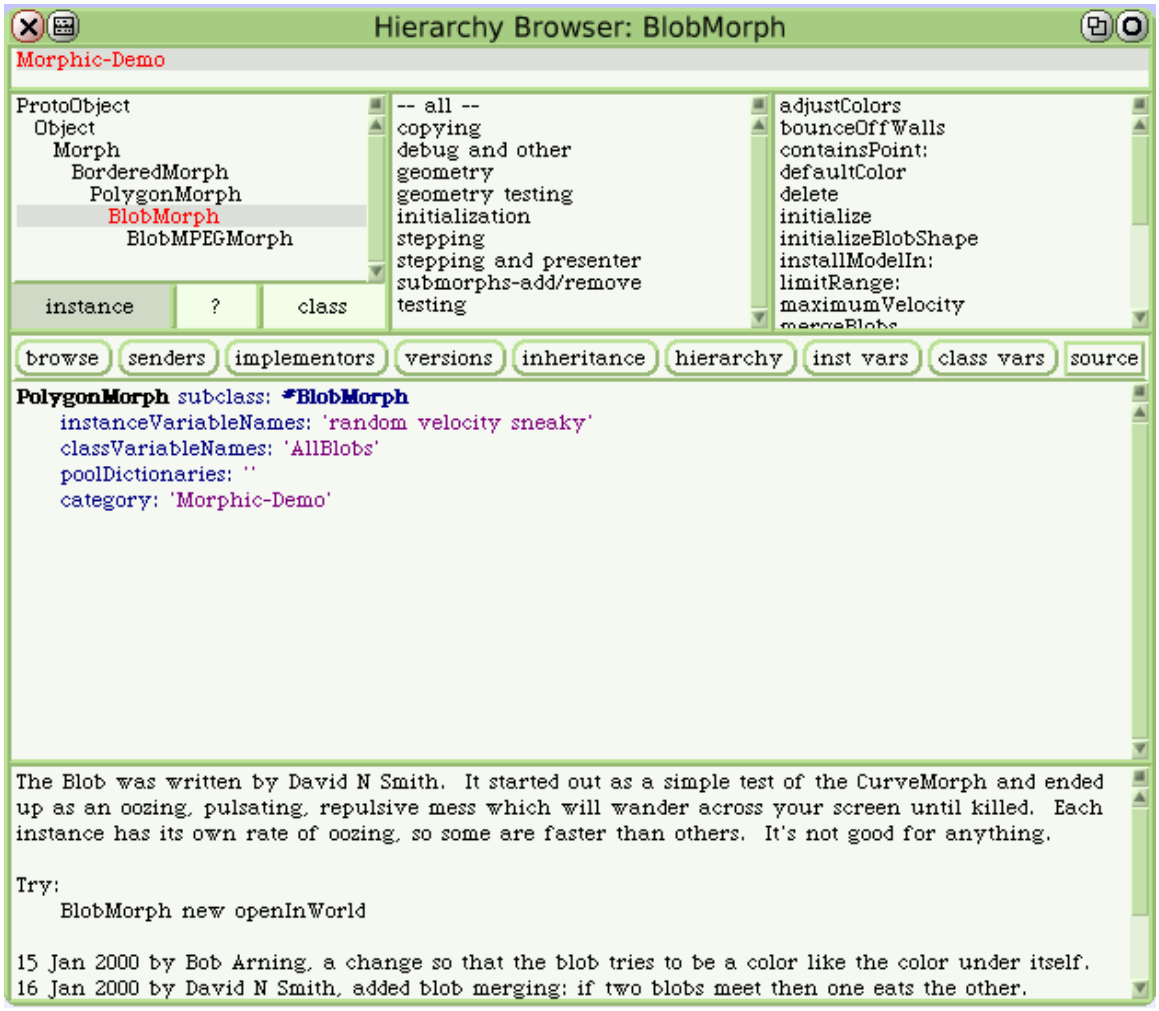


非常相似吧！毕竟 Eclipse 是来自 Visual Age for Smalltalk 的，自然会带有一些 Smalltalk 的遗风，因此如果某天你看到有人使用这个 Perspective，十有八九他曾经是 Smalltalk 程序员☺

继续我们刚才的工作，刚才我创建了一个 BlobMorph 类的实例，那么我们就看一下 BlobMorph 到底有些什么方法。在 Browser 上按 Alt + F，然后输入查找内容为 BlobMorph。



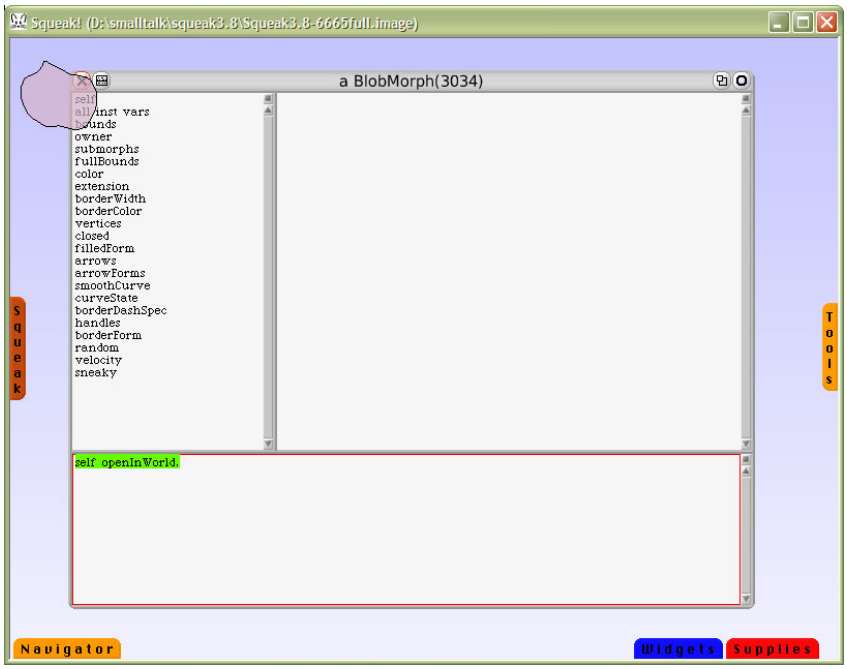
这样我们就可以找到 BlobMorph 类了，大致流览一下，似乎并没有我们所希望的方法。于是我怀疑这个消息应该是在他的父类中。我们点击 System Browser 显示源代码区域上方的工具条中的 hierarchy。将看到如下界面：



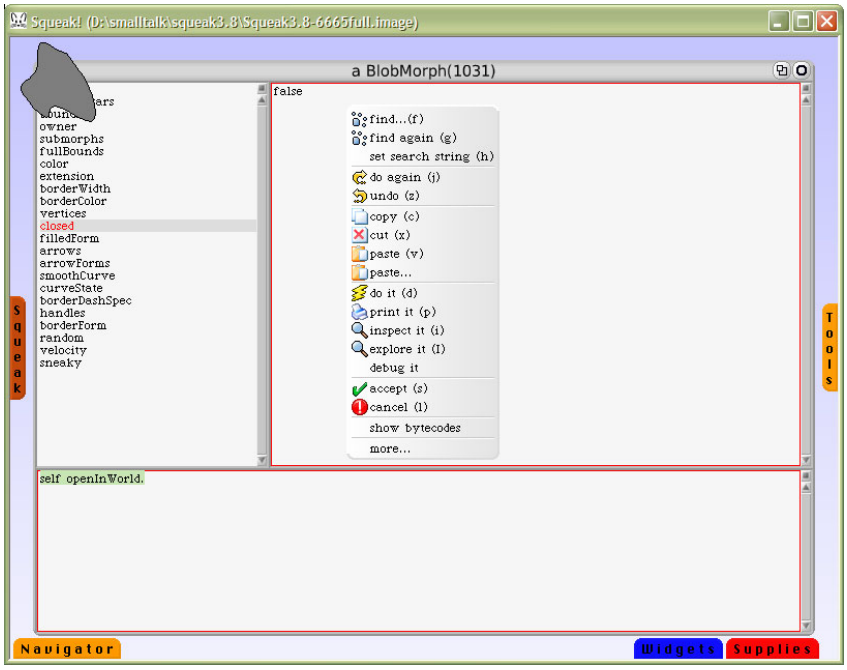
我们继续搜索 BlobMorph 的父类，发现在 Morph 类里有一个叫 openInWorld 的消息。貌似这个是我们要的东西。于是在刚才的 Inspection 窗口最下面的区域输入如下的代码。

```
aBuddy openingWorld.
```

然后选中这一行，Alt + D 来执行它。就会看到一个类似鼻涕虫（其实人家市水滴）的东西在 World 里爬啊爬的：



我们可以使用 Inspection 窗口来修改这个对象的一些值，比如现在我们看到的鼻涕虫是一个封闭的曲线，那么我们可以在 Inspection 中选择 closed 属性，在右边的窗口里会出现一个 true 的值。现在我们把这个值改成 false，然后单击鼠标右键，列出菜单：

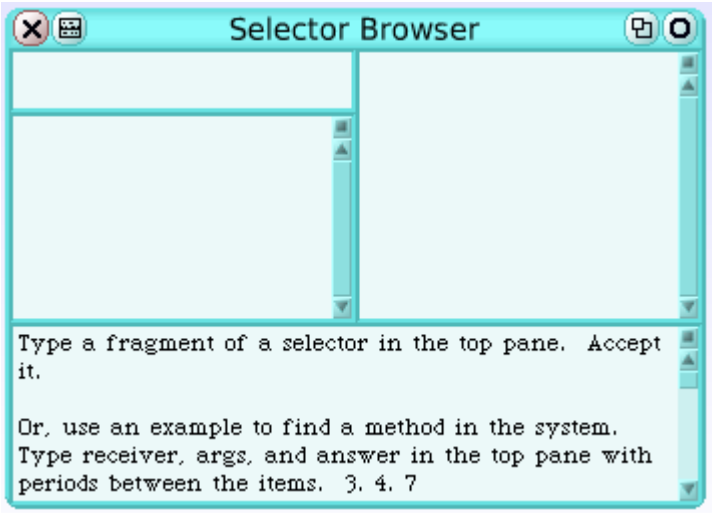


选择 accept it（快捷键 Alt + S），然后就会发现这个鼻涕虫在尾部断开了。当然这件事本身没什么意思，不过我们隐约可以感觉到 Inspection 窗口除了 Debug 之外，更是一种管理对象的方法，我们可以通过它直接修改对象的数据或者向对象发送消息。

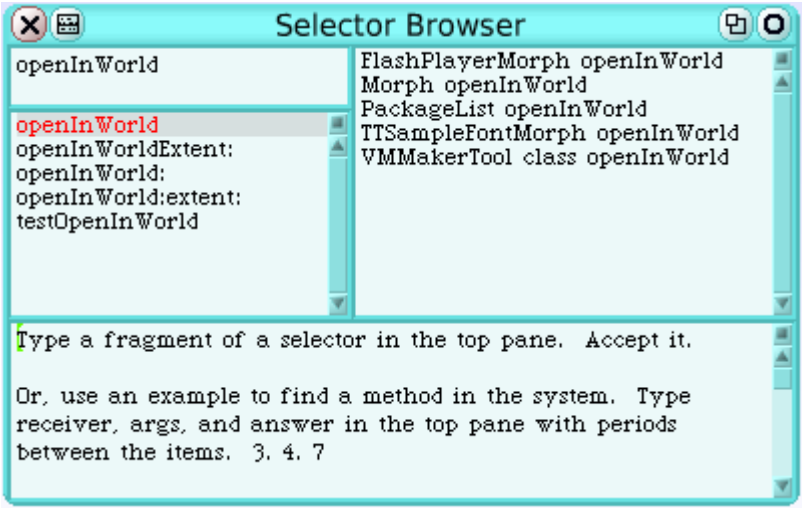
Brothers of Slug

现在我们知道怎么在 World 里搞一只鼻涕虫出来了，而且我们已经知道可以用 openInWorld 消息把一些东西扔到 World 里，我很想知道还有有什么东西可以放到 World 里。Let's go!

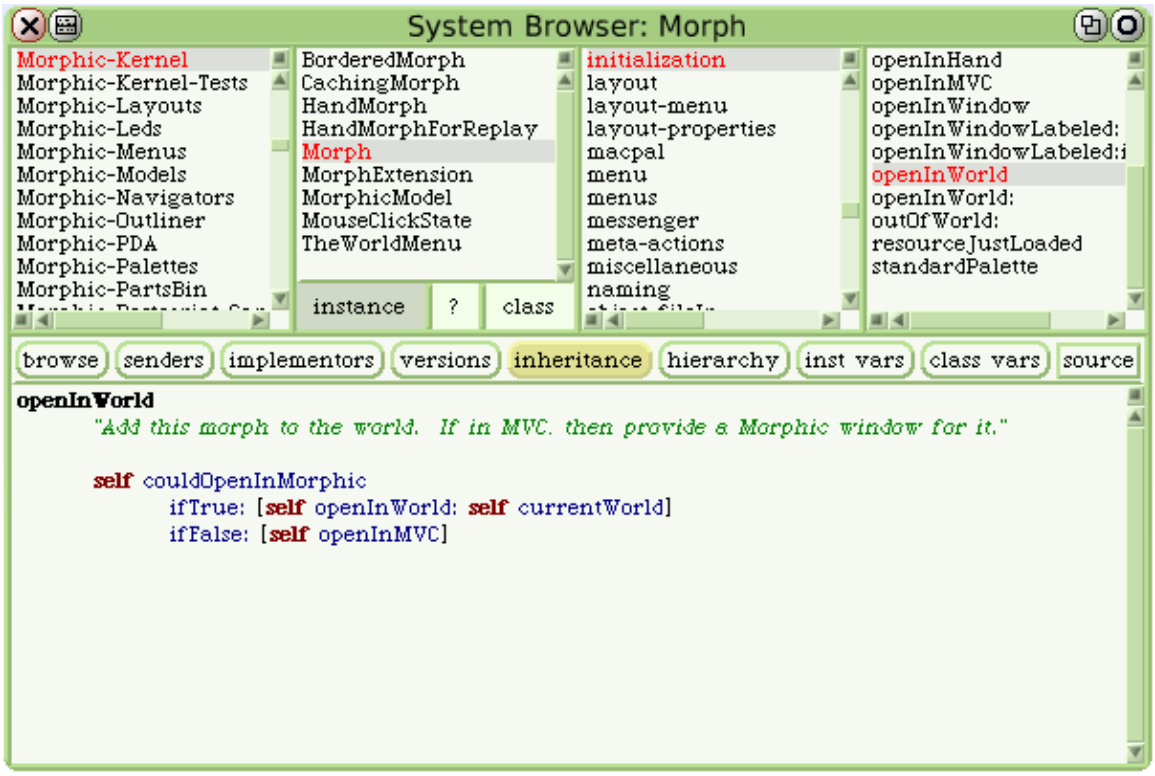
打开 Tools Panel，然后找到一个叫 Method Finder 的东西，然后把它拖到 World 中，会看到如下的一个窗口。



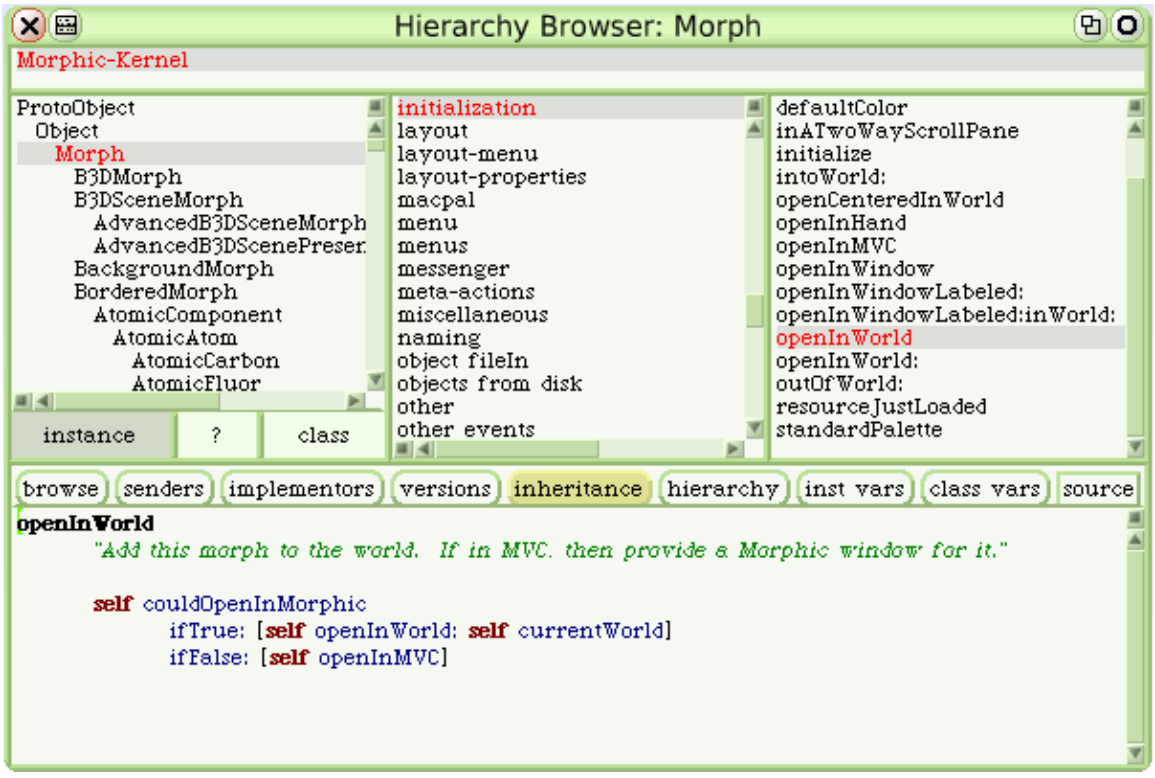
这个窗口叫做 Selector Browser，这也是在 Smalltalk 中非常常用的一个 Browser，它的作用其实是查找类型，这里牵扯到 Java 和 Smalltalk 中类型系统的差异，我们暂且不表。大体上认为这个界面就是 Eclipse 里的 open type，IDEA 里的 go to class 就行了。现在我们径直在左上角的输入框里输入 openInWorld，会有这样一些结果。



我们暂时只关心 openInWorld，提示找到这些定义了 openInWorld 的类，大概扫一眼，没有鼻涕虫兄，自然可以推断这些都是父类了。发现有我们刚才看过的 Morph 类，好吧这次让我看看鼻涕虫兄有什么同族，双击 Morph openInWorld。发现它自动把 System Browser 打开了。



老样子，我们们点击 hierarchy，会看到 Morph 的集成结构：

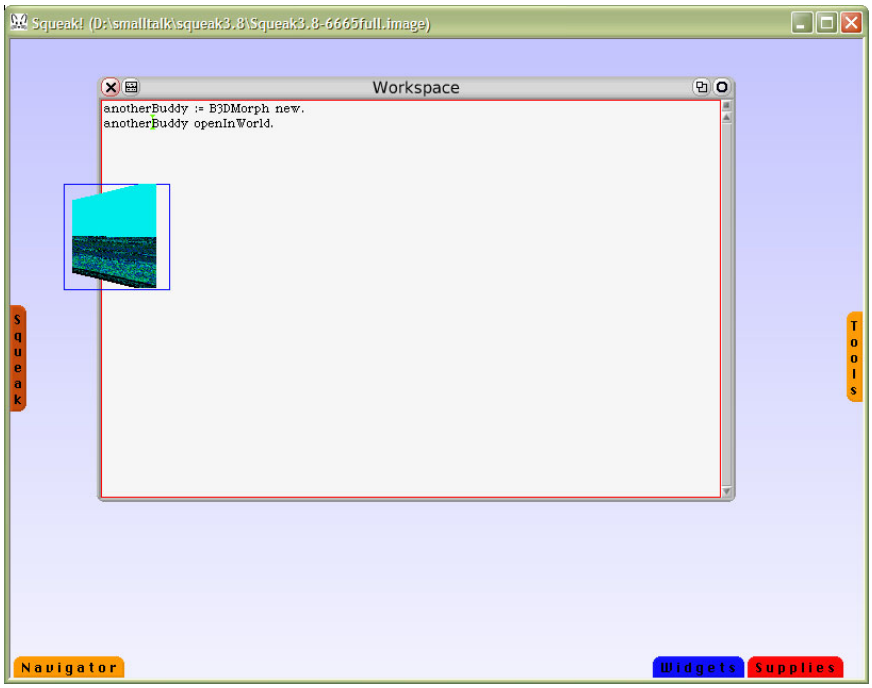


这次看到的和上次看鼻涕虫的结构有所不同，因为我们选择的子类结构也不大相同，这个功能和 Eclipse 里的 Hierarchy Outline 很像，根据选择类在继承树种所处的位置不同有不同的结果。

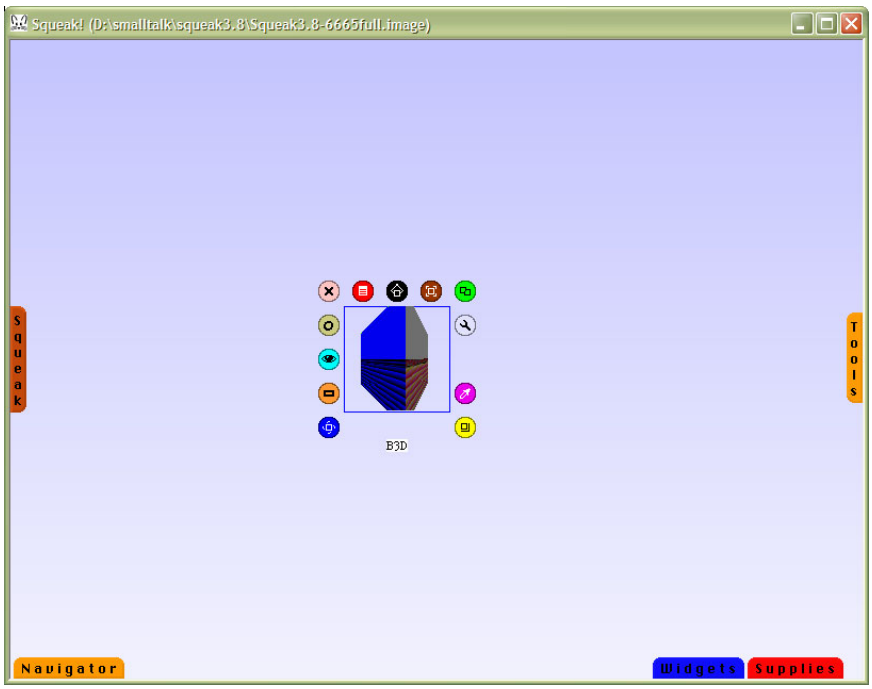
我们大概扫一眼，发现一个叫 B3DMorph 的东西，这次就是它了。这次我们不在 Inspection 里管理这个对象，直接在 Workspace 里输入如下代码，然后运行：

```
anotherBuddy : = B3DMorph new.
anothierBuddy openInWorld.
```

然后就可以看到一个旋转的三维方块了。

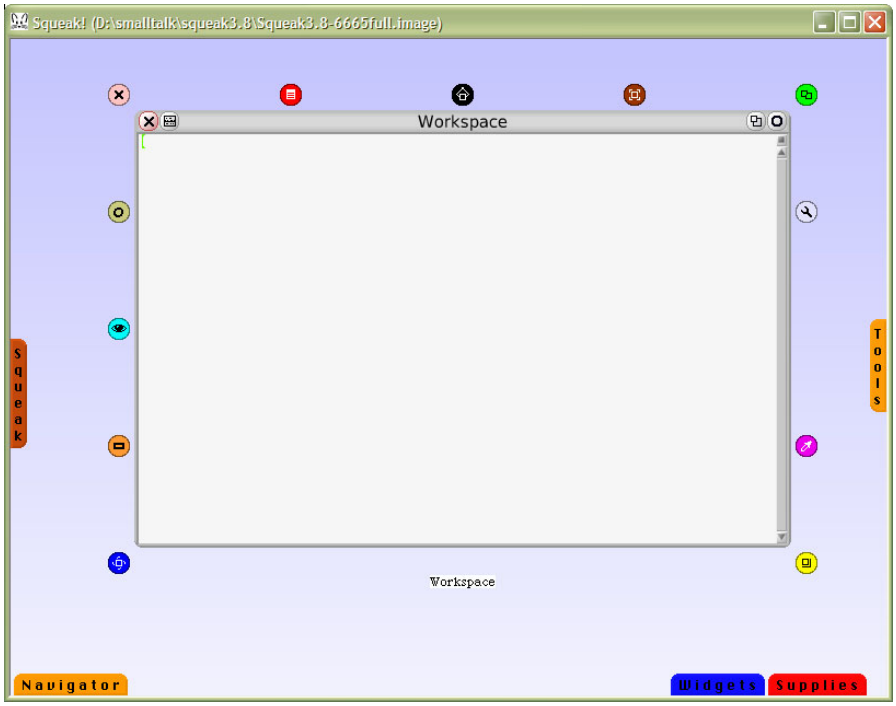


现在我们按住 Alt 然后用右键点击这个三维的方块，会看到如下的界面：

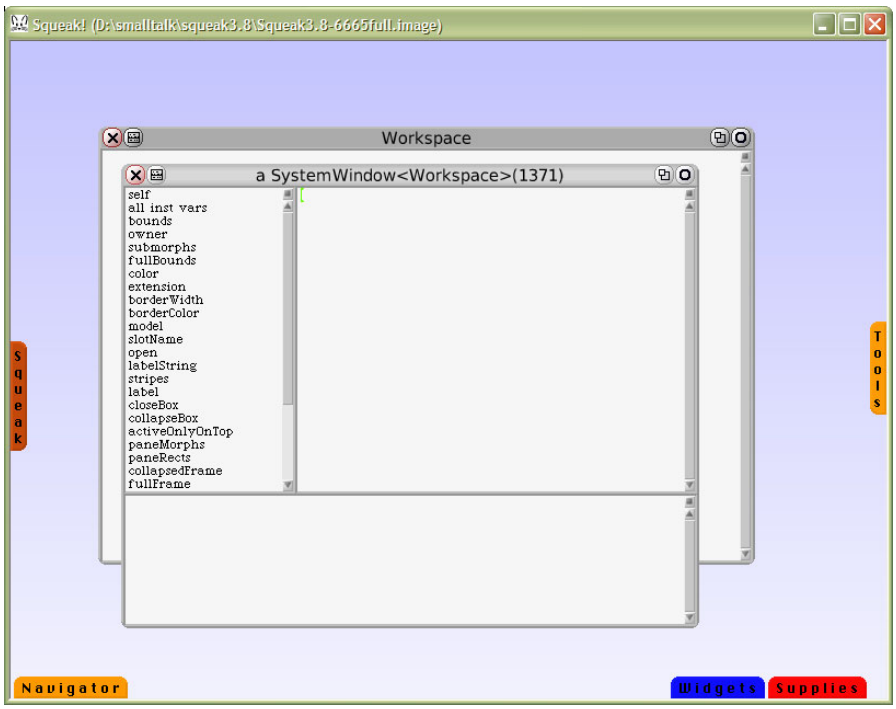


这些东西在 Morphic 里被称作 halo，halo 中的 button 代表了一些不同的可以操作的方法。在具体看这个方法之前，说一下 Alt + 鼠标右键这个操作。Smalltalk 的鼠标被设计为三键鼠标，这三个键分别被称做：红、黄、蓝。每一个键都有一个通用的操作的含义。红键表示选择，在 Windows 平台上是鼠标左键；黄键表示弹出应用程序相关的菜单，在 Windows 平台上是鼠标右键；蓝键表示激活特定对象的图形化的操纵界面，在 Windows 平台上是 Alt + 鼠标左键，而 Halo 正是 Morphic 系统中针对对象的缺省操纵界面。

在 Halo 里有很多控制对象的方法，比如更改对象的大小，或者旋转一个角度等等，其中有一个 debug button，我们按住 Shift 用鼠标右键点击一下，就会弹出这个对象的 Inspection 窗口。正如我们前面说过的，Inspection 窗口可以看作一种管理对象的方法。大部分 World 中的对象都可以通过 Debug 菜单里的 insepct it 来进行管理。比如 Workspace 窗口：



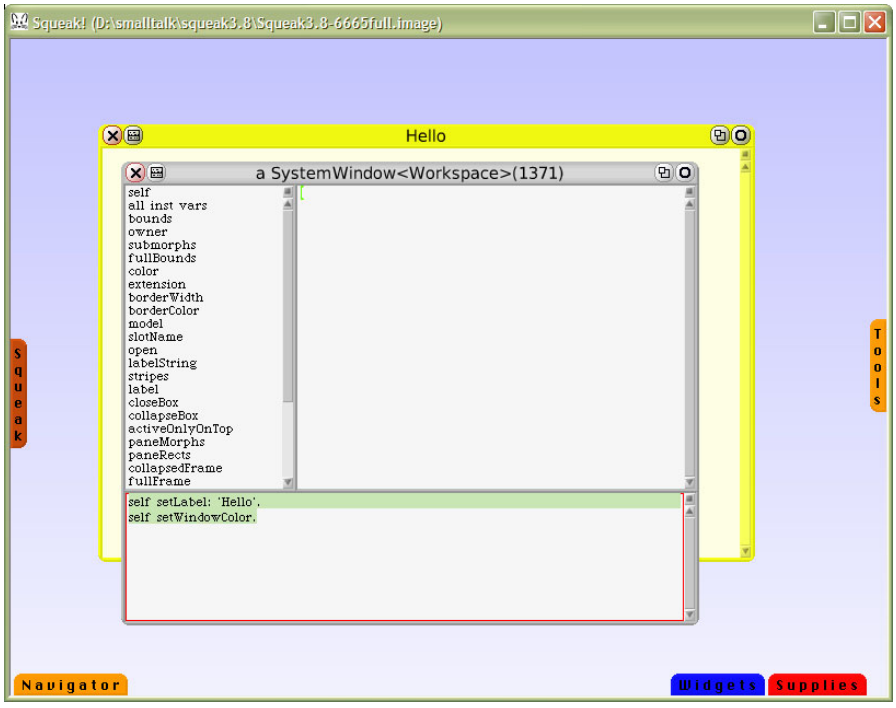
同样我们按住 Shift 用鼠标点击 Debug，会发现 Workspace 窗口是 SystemWindow 的一个实例，如下图所示：



我们可以在 System Browser 中查找一些 SystemWindow 的消息，然后调用它。比如我们看看这个两个消息:

```
self setLabel: 'Hello'.
self setWindowColor.
```

结果如下：



Squeak as VM Management Console

现在我们已经知道如何：

- 使用 Workspace 写一些简单的脚本（仅仅有对象创建和消息传递）
- 通过 Inspection 控制一个对象以及如何将 Inspection 当作通用对象管理工具
- 使用 System Browser 浏览系统中的类库
- 使用 Hierarchy Browser 浏览类的继承体系
- 通过 Method Finder 查找具有特性消息的类

虽然我们已经介绍了一些 Squeak 的开发环境，不过你可能还是非常迷茫，Squeak 还是怎么看都不像一个开发工具，这个缘于 Java VM 和 Smalltalk VM 理念上一些微妙的差异。

提起 Java VM 人们总是强调它的跨平台计算能力，然而我们往往忽略了一个简单的事实：我们通过 Java 编程来控制 Java VM 进行运算，我们程序是对 Java VM 的一种控制和管理。而 Smalltalk 和 Java 感觉上巨大的差异恰好在于，Smalltalk 认为编程是控制和管理 VM 的一种但不是唯一的方式，通过 GUI 或者其他方式同样可以管理 VM，Smalltalk 在程序之外提供了一些可能。

另外，Java 一般假设 VM 中只有一个 Application 在运行，不同的 Application 的交互体现为不同的 VM 交互。而 Smalltalk 是认为 VM 中总是有很多的不同的 Application 在运行，Application 间的交互就是 VM 内对象的交互，对于多个 VM 中对象灵活的管理就成 Smalltalk 中一个自然的需求。

最近 Java 领域中也出现了概念上更靠近 Smalltalk VM 的技术，比如 OSGi 和 Java 7 中将包含的 Java Module System。或许哪一天 JRE 里也会出现一个类似 Smalltalk IDE 的 VM 管理工具也未可知呢。

如果你喜欢 Smalltalk 这样的风格自然是最好的，如果你不喜欢也没有关系，我们一样可以只用编程的方式来控制 Smalltalk VM，那么让我们用一个很奇幻的例子结束这一节，然后进入 Smalltalk 的编程的领域吧。

在这个例子中，我们将把 Squeak 的 System Browser 变成一个 3D 的环境，下面两个地址是这个例子的操作性指南和 Flash 版的操作导引：

http://www.bitwisemag.com/copy/programming/smalltalk/squeak_3dbrowser.html
http://www.bitwisemag.com/media/flash/tutorials/prog/smalltalk/3dbrowser/ft_3dbrowser.html

我们用一个 Smalltalk 脚本达到一样效果。在 Workspace 中输入如下的代码，然后执行它：

```
Preferences enable: #systemWindowEmbedOK.

myBrowserWindow := Browser new openEditString:#Hello.
myBrowserWindow setLabel:'System Browser'; bounds: (Rectangle left:20 right:600 top:20 bottom:400).
```

```
my3DIDE := Wonderland new.
my3DIDE getDefaultCamera getMorph bounds: (Rectangle left:20 right:800 top:
20 bottom: 600).

my3DBrowser := my3DIDE makePlaneNamed: 'System Browser'.
my3DBrowser setY:50; turn:#right turns:0.38;
                setProperty: #adjustToTexture toValue: true;
                setTexturePointer: myBrowserWindow;
                setProperty: #activeTexture toValue: true.
```