SmallTalk Tutorial for Java Programmers!

Giovanni Giorgi <gio@objectsroot.com>

Jan 2002

A small paper to teach yourself the incredible Smalltalk language! It's so easy, it's so fun!

1. Introduction

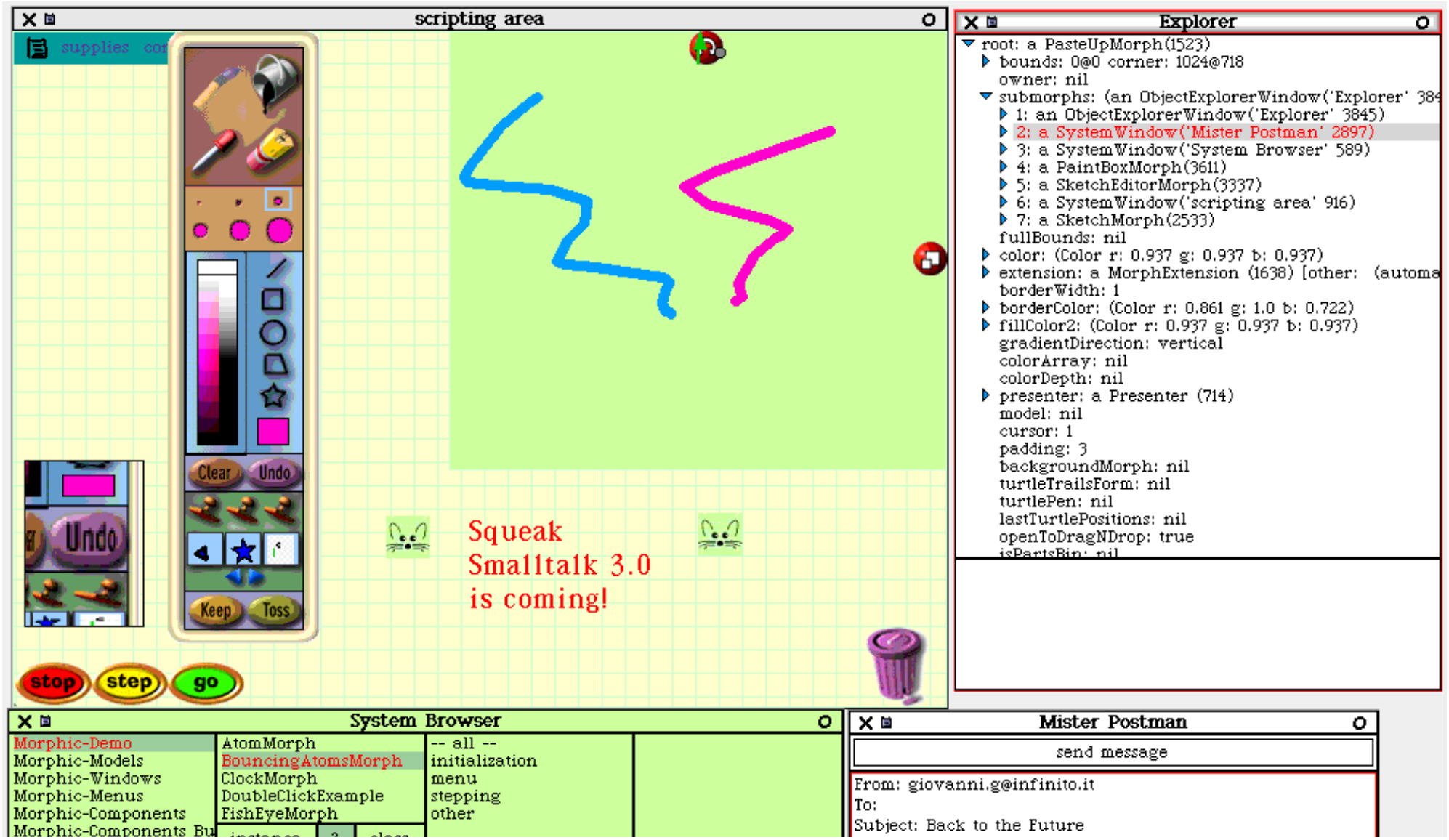
2. Why Smalltalk is so strange? (or a brief history of Smalltalk)

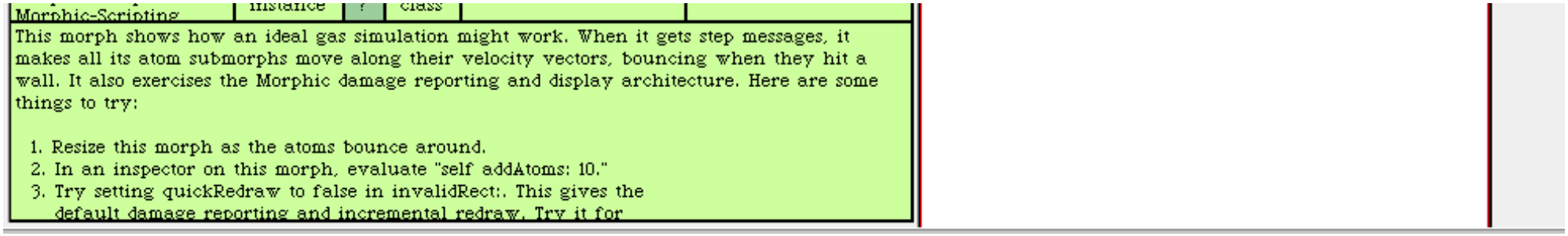
3. Main Differences between Java and Smalltalk

4. Ending Words

1. Introduction

This paper will teach you the basics of Smalltalk80 language. This tutorial suites the needs of C and Java programmers. But the tutorial can be understanded by everyone knowing a bit of C and/or OOP concepts, as we'll see. Because I will refer a lot to other books and use a lot of technical terms, I'll try to enjoy you while reading.





Squeak Smalltalk, image.gif

Finally, this is a tutorial, not a reference manual: I cannot say to you "Jump over this chapter is you are not interested" because **all** the chapters are equally important in this sequence.

1.1 Revision history

This is the Version 1.1, Date: Jan 2002 with minor fix and an explanation of the metaclass concept.

Version 1.0 Date: February 2001 First Version for Squeak 2.8

Native Format: Sgml

New version of this document

The document is available from the Author at his main site (http://objectsroot.com/squeak/squeak_tutorial.html) in these formats:

- Html Optimized for WWW Browsing (small files)
- Postscript
- Text format

Please look at the medium limitations (beyond) for choosing the right format which meets your needs.

Contact Information and feedback

If you have any suggestions, corrections, or comments, please send them to me, and I will incorporate them in the next revision of this document. I expect to do the next mayor revision of this document near April 2002, but a small revision will be done near the Feb 2002 for adding a MVC section.

You can reach me using one of these email addresses: gio@objectsroot.com, daitangio@tiscali.it

I thank a lot Stephen T. Pope (stp@create.ucsb.edu) for including this document in the Squeak cd-rom and for his supervision. I thank also my friends for reading this document before its final revision: Cristian Ghezzi, Daniele Bufarini, Marco Lamberto.

About the Author

Giovanni Giorgi is born in 1974 and is tall 1.80 meters, with black hair and brown eyes He got a Master Thesis at Dep. of Comp. Sci. (DSI) of Milan, on 23th Feb 2000, with a Thesis on Design Patterns and UML. He works with Java, likes Smalltalk and loves cats. Preferred Film: Blade Runner by Ridley Scott, The tiger and the Dragon.

1.2 Conventions used in this paper

This small paper has been written using the SGML-Tool (see http://pobox.com/~cg/sgmltools) of the linux-doc project. The arrow of the html version and all the screen-shots has been done using the GIMP http://www.gimp.org. I have changed *Html2Html.pm* module to fit my needs.

The valuable editor is Emacs 20.6.1 and the O.S. is linux

I thank a lot the GNU projects for their efforts.

For *medium limitations*, the best version is the html version. The Postscript version is a nice printed version, but for images see the images directory!

The text version is the final, desperate solution for who wants to read it in a palm computer with no-so-much-ram :-)

Typographical conventions

```
We will use this text for code you must type in your Smalltalk implementation and try
it.
```

Smalltalk versions

This tutorial is for Squeak, but you can adapt a lots of concepts to other Smalltalks. For instance, VisualAge for Smalltalk and VisualWorks should work with most of this examples.

1.3 Distribution Policy

This document has been written by Giovanni Giorgi (called the "Author" from this point).

You can copy/distribute this document providing that:

1. You do not alter or modify the document in any part
2. You do not ask money for it, excluding for the distribution.

The Author do not provide ANY WARRANTY for the content of this document, for errors or omissions and so on. There is NO WARRANTY even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

1.4 Why Smalltalk?

This small paper is addressed to the very good Java/C programmers out of there, who wants to enter in the Object Oriented World in a fast, easy and fun way. Watching the growing Java world we can say a thing: become a Java programmer can be not so difficult. But becoming a good Java programmer in a short time is not easy!!

Smalltalk is a old programming language, born before Java, C/C++. Smalltalk has a good base library (first attempt to build it was done before 1980!) and a very productive engine.

If you use Java in your work-of-every-day, you can run Smalltalk without loosing speed or efficency (and probably you can get a good improvements with a commercial Smalltalk version!).

Why Smalltalk is not so used? The reason was the high cost of Smalltalk in the '80 years and until the 1995. Worst, the fastest Intel-PC in the 1990 was too slow for using Smalltalk (or Java or Lisp) in the all-day work.

If you feel C/Java language is nice, but you want a way for increase your productivity without looking for bugged code every day, Smalltalk can give you a solution. C++ is not often a solution because a good C programmers continue to use is as an "enhanced C" without using the full power of the OOP. C++ is for efficency: but efficency can lead to bad Object Oriented code, because you do NOT tend to isolate the classes as they should be.

I used Java, C and C++, then I discovered Smalltalk. It seems incredible, but you can do a lot of thing with Smalltalk: you can find the bug very quickly and enjoy a lot!!

In the last five years you can download free version of Smalltalk, which are quite powerful and fast, and you can buy a professional Smalltalk at a reasonable price.

This because the competition of Java has lowered the costs of the implementations.

Best, you can find the same Smalltalk implementation under Unix, MS-Windows and Mac, so it is widespread as the Java/C languages.

So, why not invest a bit of our time to learn a dynamic, powerful and fun language?

1.5 Background Required

Okay, now we will start talking about what you should know for understand this small paper:

- You must know Java: you must not be a wizard, but you must know all the Java reserved words, data types, and so on. You can ignore how to do fantastic tricks as Enterprise Java Beans (but you know what is a Bean, *right*?!)
- You should be a bit familiar with Object Oriented concept like class, inheritance and so on. If you are not do not worry, but you should have a vague idea of the meaning of this words, because I am *not* going to do a course on O.O.P. in a *strict* sense. As we'll see Smalltalk is the first Object Oriented language that induce the concept of Object and inheritance as we know it now. But Smalltalk has some lacks (no a strong information-hiding concept, no protection, no static behavior).
- You should know how to start a Smalltalk image, execute some code.

- You should have a little of time to invest, and an open mind....

2. Why Smalltalk is so strange? (or a brief history of Smalltalk)

Smalltalk introduces three big concept in the 1980:

1. The concept of GUI as we know now and the concept of IDE, an Integrated Development Environment with a integrated Code Manager and Database.
2. The concept of class/object and so on. Best, Smalltalk it is a *pure Object Oriented language*, so all the GUI, the IDE and the tools used for editing the code are written in Smalltalk

   Even the bytecode compiler, which translate our code in bytecodes is written in Smalltalk!

3. Is the first pure dynamic language.
4. It is funny to use!

Alan Kay wanted a language for *ordinary people* and he gets it. Smalltalk uses only 2-3 concept, is easy to use and understand, and incredible powerful. Smalltalk can be used and understood by children. This has the well-known cost: the speed is not so high, but RISC chip have solved this problem: for example, Squeak is used for multimedia application and it is quite fast.

2.1 Download Right Now!

For teaching, we must have a free Smalltalk to start with.

We have a wide choice, but I suggest to you Squeak (you can find it in Squeak main site) because it have a good large library, it is free and you can get it for Unix-X11, Ms-Windows, Mac, or your preferred PDA (!).

After you have downloaded your zip (or tar or other archive type) expand it in a directory (for instance called Squeak).

A Smalltalk implementation is composed of an image (binary code), a major source file, a second "changes" file. The image is called Virtual Image (VI) because is independent form the platform you use for running Smalltalk.

In Figure1 you can see a taste of Smalltalk. The green window is a Browser, the yellow window is a workspace, and finally you see an Inspector titled "SmallInteger: 3". As you see, every window with a different purpose has a different color.

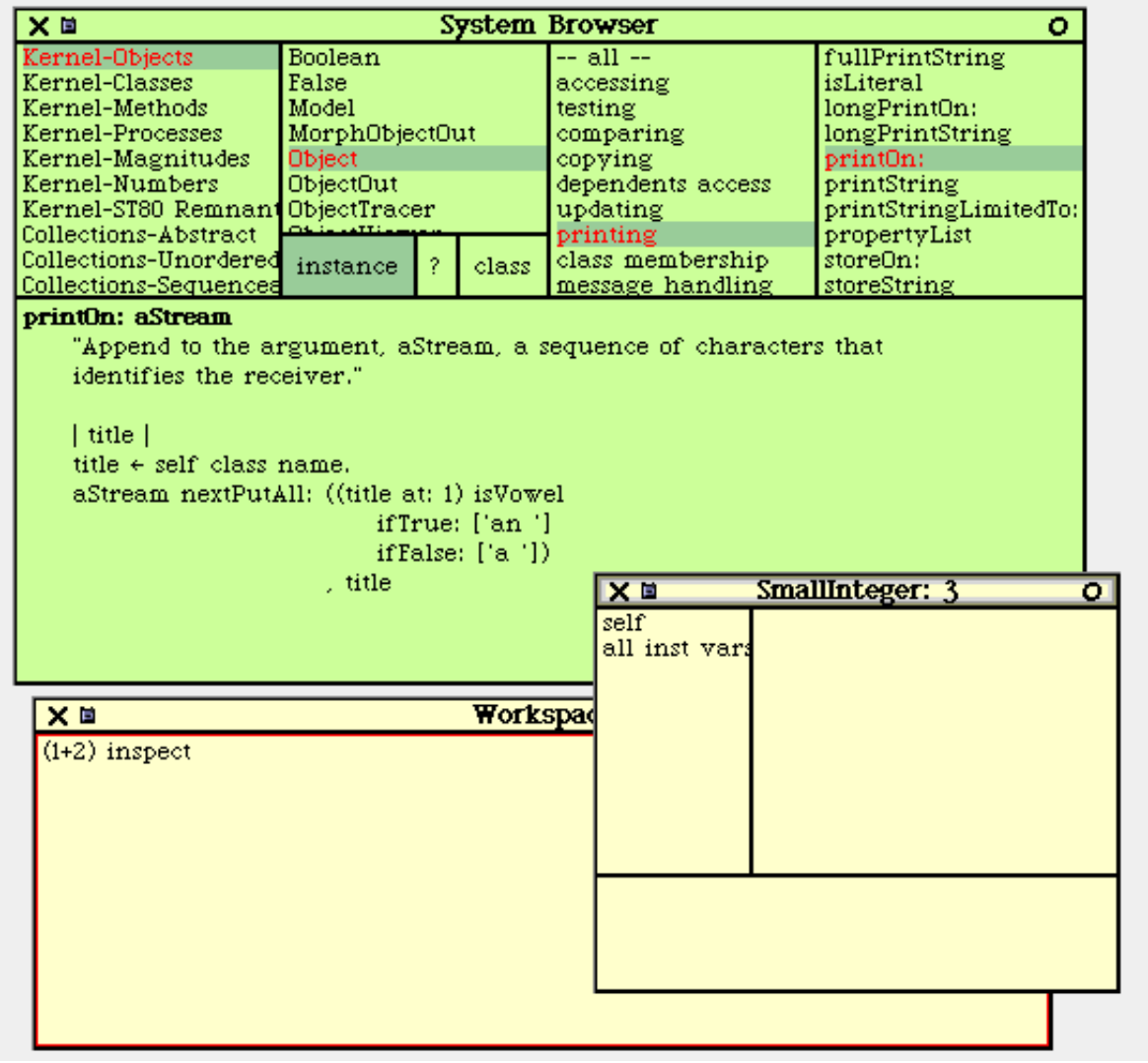


Figure 1, A taste of Smalltalk

The **Browser** is your best friend, and it is used for exploring the class stored in Smalltalk. In the leftmost panel, you see list of Categories. used for grouping the classes. Then you see a list of classes (Object is selected) of the selected category. Then you see a list of protocols, used for grouping methods (the protocol for printing is selected). Finally we have selected the printOn: method we see in the panel below.

The **Workspace** is a window where you can type what do you like and then ask Smalltalk to execute it. In Figure2, you see the menu you can pop-up pressing the middle button of the mouse. Note: The exact mouse button can change depending of your environment (for example on Mac you have only one button...). Refer to the Squeak documentation for more information,

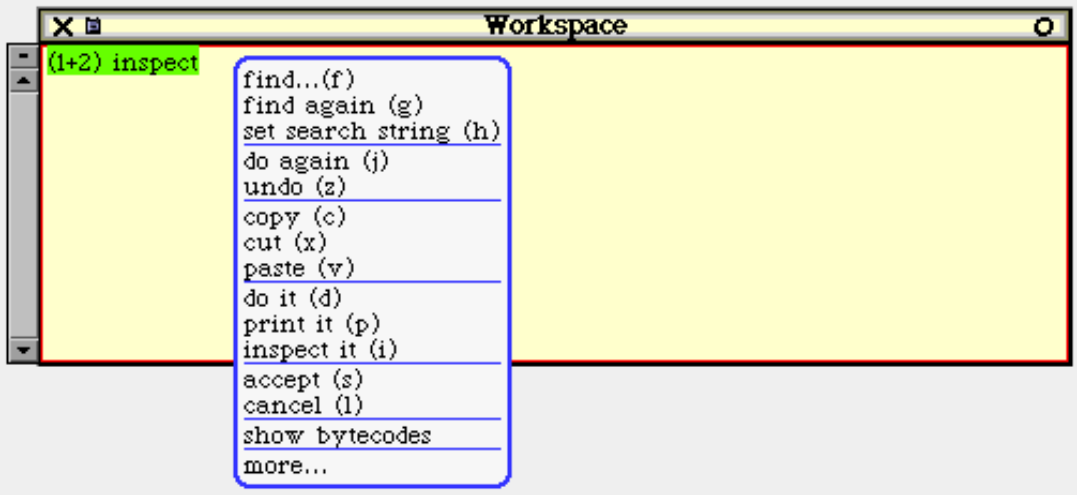


Figure 2, The Workspace

What do you can do with this menu?

- You can execute the code with **do-it**
- You can inspect the result using **inspect-it**
- Then you can print it, with **print-it**

In the following examples, I suggest you to print the results or inspect them.

2.2 Hello world in Smalltalk

We now will write our first Smalltalk code!! Open a Transcript and a Workspace, using the main menu: click in a empty area of the screen, then select the sub menu "open..." as in Figure 3.

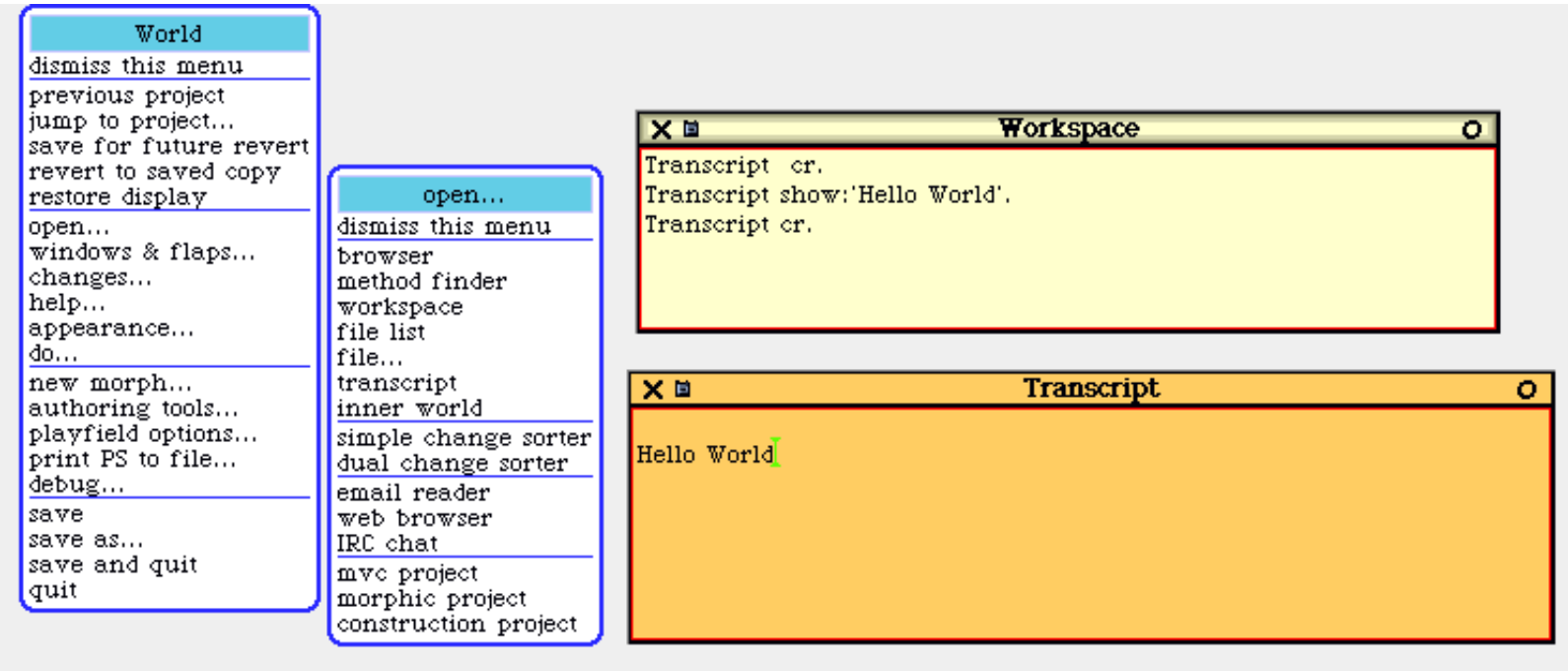


Figure 3, The First Example

Please type in a workspace:

```
Transcript cr.
Transcript show:'Hello World'.
Transcript cr.
```

Now open the menu of the workspace and select do-it or press cmd-d

> *Note:* The cmd key changes in respect of your O.S. and environment. If you are using windows or linux use alt-d. If you are using an Apple Machintosh, use Apple-d. We will refer to this key as cmd, as the Meta key on emacs.

What does this code? It simply prints a carriage return, the string "Hello World" and another carriage return.

Please observe these small things:

1. In Smalltalk, all strings are enclosed in single quote as in '**Hello World**'
2. A double quote is used for comments: **"This is a nice comment"**
3. The point **.** is used as a statement separator.
4. The **;** (semicolon) character has a special meaning, as we see beyond.

Okay, this look like when you go from Pascal to C, with all this oddities...but do not worry!

Now we will analyze our small "program" in Smalltalk in every line:

1. **Transcript** is an object used to reference to the Transcript window. For the moment, I will trust to your intuitive notion of Object.
2. **cr** is a message directed to the Transcript. You are just talking to the Transcript saying it a very small thing: "please print a carriage return"
3. **show:'Hello World'** is a parameterized message where you pass to the Transcript what actually you want print!! Even if some concept are obscured, the simplicity of the language should be sufficient to understand it.

2.3 Big Numbers

Now we will see a small thing you *cannot do* in Java or C with the standard library. Type in & evaluate:

```
2 raisedTo:128
```

---

we will obtain the right number:

```
340282366920938463463374607431768211456
```

Another small example is:

```
3 raisedTo: 32 =>  1853020188851841
```

These numbers are LargePositiveIntegers objects, and the class comment says to use:

> I represent positive integers of more than 30 bits (ie, >= 1073741824). These values are beyond the range of SmallInteger, and are encoded here as an array of 8-bit digits. [...]

So we can play with very big integers without problems!

In fact, in Smalltalk is quite easy to build dynamic and not-limited structure. For example, imagine you want build a small collection of three strings; you can do it with:

```
(OrderedCollection new) add:'first'; add:'second'; add:'thirdString'.
```

You will use often OrderedCollection objects, instead of fixed Arrays. It is so easy to use dynamic structure, because the language provide a garbage collector and an easy framework for accessing to them, as we see in the next paragraph. The semicolon ';' i used for separating more messages sent to the same object. For instance, the first Transcript example can be re-written as:

```
Transcript  cr; show:'Hello World'; cr.
```

Let's see another example:

Java Version:

```
Foo f;
// A comment
f = new Foo("AString");
```

Smalltalk Version:

```
|f|
"A comment"
f := Foo new: 'AString';
```

We declare temp variables in vertical bars, without inserting the type of the object. *Smalltalk is dynamically typed: you do not need to know the type of your object before you use it!* The assign operator in Smalltalk con be written in ':=' or, under Squeak, using the special char <-. Under Squeak this char is typed using the "_" character.

The Smalltalk expressions are generally in the form "object message". *They are evaluated from left to right!*

For example, the Java expression:

```
int i;
i=1+5*6;
```

---

will became:

```
|i|
i := 1+(5*6).
```

Smalltalk will reduce this expression in this way:

```
1+ (5*6) => 1+30 => 31
```

If we write

```
1+5*6
```

we get:

```
1+5*6 => 6*6=>36
```

This seems a bit strange, but this evaluation are done in the same way for every expression. What about method with more than one parameter?

Look at this:

Java Version:

```
Foo f;
...
f.methodWithTwoParam(1,2);
```

Will be written as:

```
|f|
...
f methodWith: 1 twoParam: 2.
```

2.4 Code Blocks

First of all, we will look a two different way of enumerating elements from a collection:

Java Version:

```
Enumeration list;
Apple elem;
list=aVector.elements();
while(list.hasMoreElements()){
  elem=(Apple) list.nextElement();
  //work wirh elem here
}
```

Smalltalk way:

```
aVector do:[:elem |
  "work with elem here"
]. "End of iteration "
```

And last, for doing small loop in Java you write:

```
for(int i=1; i<=10; i++) {
  System.out.println( Integer.toString(i) );
}
```

In Smalltalk you can write:

```
1 to:10 do:[:i|
  Transcript show: (i asString).
].
```

In this example we send the special method to:do: to the object 1. This means: 'Tell to 1 to go until 10 doing at every loop the code block I give you'. Simple, isn't it? The "code block" is enclosed in '[....]'. The to:do: sends a object (called i) in the block. It is a simple way of passing arguments. So, it is simple to do something like:

```
if(x>0) {
 x = x+1;
}else {
 x=0;
}
```

Will become:

```
(x>0) ifTrue:[ x:=x+1. ] ifFalse:[ x:=0 ].
```

We send to the result of the evaluation of '(x>0)' the message ifTrue:ifFalse.

You can have block without parameters, for example:

```
5 timesRepeat: [
 Transcript show: 'j'.
].
Transcript cr.
```

Will print five j and then a carriage-return on the Transcript.

2.5 The while loop

In Java, you write:

```
int i=5;
while(i>1) {
 System.out.println( Integert.toString(i*2) );
 i--;
}
```

In Smalltalk you can use:

```
|i|
i:=5.
[i >0] whileTrue:[
 Transcript show: ((i*2) asString) ; cr.
 i:=i-1.
].
```

You can try some more complex examples as yourself!

The Squeak IDE: going on...

For learning base usage of Squeak IDE, please refer to the read-me and to the documentation you find in the Squeak Web site or try this tutorial: http://kaka.cosc.canterbury.ac.nz/~wolfgang/cosc205/smalltalk1.html (see [IDETutorial] on References (below))

3. Main Differences between Java and Smalltalk

3.1 Java Versus Smalltalk

What are the point of contact of Smalltalk and other languages such as C and Java? Smalltalk is executed using a Virtual Machine (VM) as Java. The VM has a very complex Garbage Collector (GC) very similar to the HotSpot. And the Smalltalk code is stored in a byte-code form. But Smalltalk holds all the code in only one file (the image) and store in it all the state of the system (working Threads and so on). The image is coded in a binary-independent format (as java .class files) and can be ported across different platform. For instance, Squeak can run on Macintosh, Unix, OS/2, Ms-Dos, Windows 9x/NT/2000, and Windows CE. Even, it is easy to port the VM to other platform, because is coded in C language.

The Entire IDE, and the GUI are written in Smalltalk, mostly using the MVC (Model View Controller) design model.

> Even the Java Swing uses a MVC-like model. Squeak has even a best and powerful GUI, called Morph. See the figure Morph.

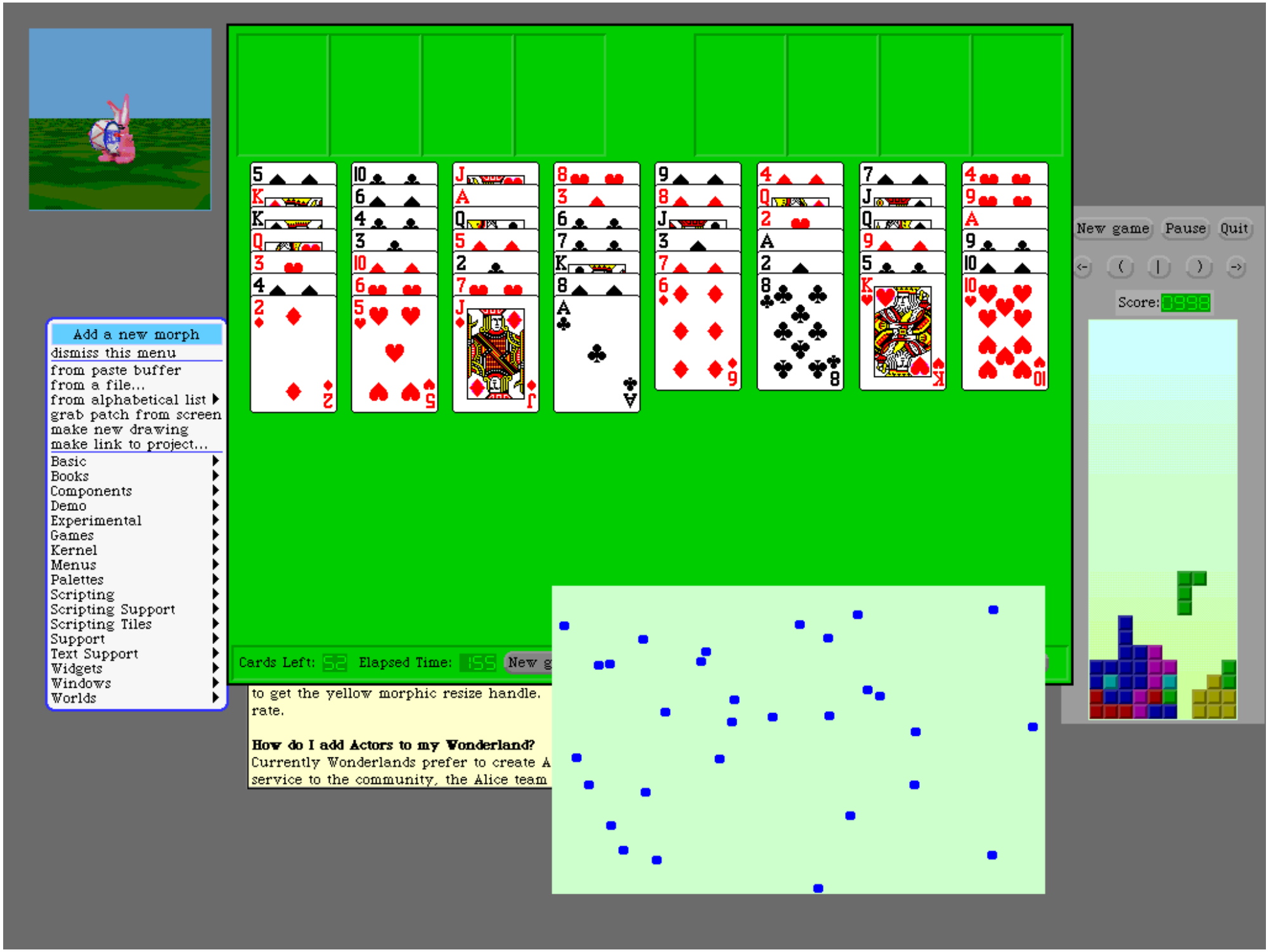
The Smalltalk IDE uses reflection for exploring itself in a very massive way. For example, writing a Proxy object is trivial!!

Every class should have a comment: like in the Javadoc, you can integrate the documentation in the code! Squeak can create even hyper-link between two line of code or two comments!

About namespace: before the 1999, the concept of **name-space** wasn't implemented in the commercial versions of Smalltalk. For instance, in Squeak all the global objects are stored in a object called **Smalltalk**. VisualWorks 5 provide namespaces, and Squeak has a similar way for isolate projects code.

The Gui, the Network and the Sound.

Exploring the big Squeak library is impossibile in this small tutorial, because of its size. But I'd like to show you only a small idea of some of the applications you can use:



Morphic: not only boring widgets!

With Squeak, and in less of 7Mb of image, you get a Email reader (Celeste), a Flash reader, a small Web browser (Scamper) a dynamic web server (PWS) and a special 3D library (3DBalloon) integrated with the Alice Authoring Tool http://www.alice.org. And even for music, you can play midi, digitalize sound and so on!

Note: The base Java 1.3 library is about 12Megabytes (uncompressed) without an ide or sound support!

> The reason is simple: the .class format need to explicit say to the Java-VM what other classes it needs, for dynamic link. This is add a lot of redundancy. For instance, if you compress the rt.jar file, you get 4.3 Megabytes (ratio 65.6%). If we try to gzip the Squeak image, we got 3.10Mb (ratio 53.7%). So the Smalltalk image has less redundancy!

3.2 Types? No thank you!!

When you program in Java, you use types. You use int-s, floats, classes and so on.

We have seen a Class can be a nice place to put a struct. In Java, you often need to do a cast from a type to another. This is very common, and no one has trouble. In Smalltalk, the Virtual Machine only understands objects. If we execute 1+2 the VM of Smalltalk search the method + in the object 1, instance of the Number class. If Smalltalk do not found the method, will throw a Exception at Runtime

> Exceptions are defined in ANSI Smalltalk. Some Smalltalk implementation may not have Exception, but can raise Errors.

The language-designers say type are userful, because the compiler can produce efficient code if it knows the type we are using (the sum of an two numbers is fast if they are integer and not float!).

Someone thinks type are userful to human-programmers, but in my own opinion, after two years of Smalltalk programming, it is not-so-true.

So, let's start to see how to program without type (and without casts...or other magic powers...)

3.3 The Squeak base library compared with Java

This is a small overview of the base library of the Squeak 2.8.

First of all, suppose you have a unordered collection, and you want to eliminate duplicates. Doing this in Java is boring, in Smalltalk is quite easy:

```
|uc unq|
uc:=OrderedCollection new.
uc add: '2' ; add: '1'.
uc add: '3'; add:'3'.
unq:=uc asSet.
```

If print unq, you get:

```
Set ('3' 1 '2' )
```

Now you can sort it simply using:

```
unq asSortedCollection
```

---

obtaining:

```
SortedCollection ('1' '2' '3' )
```

The large base library of Smalltalk can manage in this way complex data structures.

If you want collect items, try this:

```
|c s|
c:=OrderedCollection new.
c add: 1 ; add: 1.
c add: 2; add:3; add:4.
s := c select: [:i| i <=2 ]. "[*] Collect items < 2"
Transcript cr; show: (s asString) ; cr.
```

Obtaining:

```
OrderedCollection (1 1 2 )
```

First, we create a collection with five elements (1,1,2,3,4). Then the collecion c is filtered to just the elements less than two (in line [*]). The Collection classes implements also a **reject:** message, which is the logial inverse of **select:**.

3.4 The Smalltalk Code database

As you see, all in Smalltalk is written in Smalltalk! Not only the Smalltalk compiler is written in Smalltalk, but even the database holding the classes in the System is written in that language. A special dictionary, called Smalltalk, is a good starting point. Inspect it with:

Smalltalk inspect.

The Smalltalk Dictionary contains a reference to ALL the class on the system. If you want know how much classes the system has simply print:

```
Smalltalk size.
```

I have got 1193 classes on my Squeak 2.8 You can ask to the classes about their life and structure! Try for example:

```
( (Smalltalk class) inheritsFrom: Object) inspect.
```

We ask if the class of Smalltalk (SystemDictionary) is a subclass (inheritsFrom...) of the class Object, and we get true as answer.

Now open a browser with:

```
OrderedCollection browse
```

---

and click on the '?' button for getting the comment of the class. I suggest you to explore these clases: String, OrderedCollection, SortedCollection, Dictionary, Set. The Number class is also good for understanding how Smalltalk works with numbers.

Metaclass structure

There is also a dictionary for undeclared elements:

```
Undeclared inspect.
```

The Behavior class is used for describing the objects. The Hierarchy is like this

```
ProtoObject ()
        Object ()

                Behavior ('superclass' 'methodDict' 'format' )
                        ClassDescription ('instanceVariables' 'organization' )
                                Class ('subclasses' 'name' 'classPool' 'sharedPools'
'environment' 'category' )
                                        [ ... all the Metaclasses ... ]
                                Metaclass ('thisClass' )
                        Oop ()
                        Unsigned ()
```

In Smalltalk, Classes and instances are objects, and share some common properties. This is not true in Java, because static methods cannot have all these properties. The 'class' method give us the class of an object; and also classes can resppond to 'class' method. But...wait a moment, what is the class of Class? It is Metaclass:

Metaclass:

> Metaclass instances add instance-specific behavior to various class-describing objects in the system. This typically includes messages for initializing class variables and instance creation messages particular to a class. There is only one instance of a particular Metaclass, namely the class which is being described. A Metaclass shares the class variables of its instance.

Look at this table:

```
#|   Expression                    -> Output
-----------------------------------------------
1 |  (Object new) class             -> Object
2 |  Object class                   -> Object class
3 |  Object class class             -> Metaclass
4 |  Object class class class       -> Metaclass class
5 |  Object class class class class -> Metaclass
5'|  Metaclass class class          -> Metaclass

Note: 5 and 5' are the same.
```

The trick is in the 5th expression. We should define MetaMetaclass for the 5th output. For avoiding this infinite recursion, Smalltalk overlap 'Metaclass' and 'Metaclass class'.

```
[Subtle] In general, the superclass hierarchy for metaclasses parallels that for
classes. Thus,
        Integer superclass == Number, and
```

```
        Integer class superclass == Number class.
However there is a singularity at Object. Here the class hierarchy terminates,
but the metaclass hierarchy must wrap around to Class, since ALL metaclasses
are subclasses of Class. Thus,
        Object superclass == nil, and
        Object class superclass == Class.
```

3.5 Multiple-Inheritance

Smalltalk only supports single-inheritance, as Java. Java uses the interfaces for implementing a less strong interface inheritance. In respect of C++, interfaces can be thinked as *pure abstract classes*.

In Smalltalk you do not really need multiple-inheritance: you can share a set of protocols between classes. You can think protocols **as set of methods**. This give you a very powerful freedom: you can reuse code with similar meaning! Best, doing refactoring and improving code is simpler. In Java, if you want to add a method to an interface, you must implement it in all the implementing classes. In Smalltalk, you can do a smart thing: you can build a default implementation in a superclass (for instance Object) and reimplement only where it is really needed! Even if the code is not valid for every Object, it will work if correctly needed. Smalltalk will not stop you: if you know what you are doing, you will get maximum power.

But be careful: this freedom can produce very ugly code, so you must follow good programming pratice. The lack of interfaces and namespaces has been fixed in VisualWorks (see References). Squeak will provide a similar function in the future.

4. Ending Words

4.1 Going on

I have given to you only a small taste of Smalltalk. The true power of this language is under this surface, under the deep water of Object Oriented Programming.

Design Patterns and Smalltalk

About Design Patterns, a good text can be [AlpBroWoo98] Here you get a comparison of Design Patterns of the [GoF95] With this two books you can compare C++ and Smalltalk, looking how the two languages implements the same patterns.

Future of Squeak

Looking at Squeak 3.0 we see a powerful ide, for experimenting, developing and not only... it is a very fun environment! The Jitter Engine will give to Squeak a powerful VM, faster and free.

4.2 Commercial and free products list

The complete list is not-so-short.

I can give you a rouge idea:

- Visual Works 5i, before was of the ObjectShare http://www.objectshare.com now is sell by Cincom http://www.cincom.com It is a big Smalltalk with a large library and a lot of library code. Best, it executes code with a powerful JIT (Just In Time Compiler). It is probably the fastest Smalltalk on the market, in my own opinion.
- Squeak http://www.squeak.org/. Created by Alan Kay, Dan Ingalls, John Maloney and many others authors of first Smalltalk80. Squeak has multimedia support, a fantastic community, and a large base of software. See [IngKaeMal]
- Smalltalk MT
- Smalltalk/X is very powerful, and it is a viable solution for developing under X11. The last

license seems free also for commercial usage. Take a look to it.
- Dolphin Smalltalk http://www.objects-arts.com/ is a very simple Smalltalk for MS-Windows, but it has good tutorials.
- Smalltalk Agents http://www.qks.com/
- Visual Age for Smalltalk http://www.ibm.com It is a professional products, very easy to use thank to the Visual Age Ide. Last but not least, the Visual Age for Java's IDE is written in Smalltalk too!

Some of this products (as VisualWorks and Visual Age) are very big and multi platform, other are very very small and cheap (like Dolphin) so you can choose your preferred commercial product without worrying about it. If you like, try them in this order: Squeak, VisualWorks, SmalltalkX and VisualAge. You can get trial version of the latest 3 products (I have my preferred environments, but I will not say them here...email me :-)

4.3 References

1. [Grogono87] Peter Grogono, *Programming in Pascal*, Addison-Wesley,1987
2. [Squeak] Squeak Home Page: http://www.squeak.org
3. [IDETutorial] http://kaka.cosc.canterbury.ac.nz/~wolfgang/cosc205/smalltalk1.html
4. [GoF95] Gamma, Helm, Jhonson, Vlissides, *Design Patters: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995
5. [AlpBroWoo98] Alpert, Brown, Woolf, *The Design Pattern Smalltalk Companion*, Addison-Wesley,1998
6. [IngKaeMal] Ingalls, Kaehler, Maloney, Wallace, Kay, *Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself*, 199?