Introduction to the Smalltalk Programming Language

Michael McThrow

San Jose State University
Computer Science Department
CS 152 – Programming Paradigms

November 22, 2021

Table of Contents

Table of Contents

History of Smalltalk – Xerox PARC

- Smalltalk was developed at Xerox PARC (Palo Alto Research Center) in the 1970s.
- The 1970s-era Xerox PARC was a fount of innovation in computer science, researching very important technologies such as graphical user interfaces, Ethernet, and programming languages. Aside from PARC's work on Smalltalk, other languages PARC worked on were Lisp (Interlisp), Mesa, and Cedar.

History of Smalltalk – Xerox PARC

Xerox PARC produced many luminaries of computer science, including:

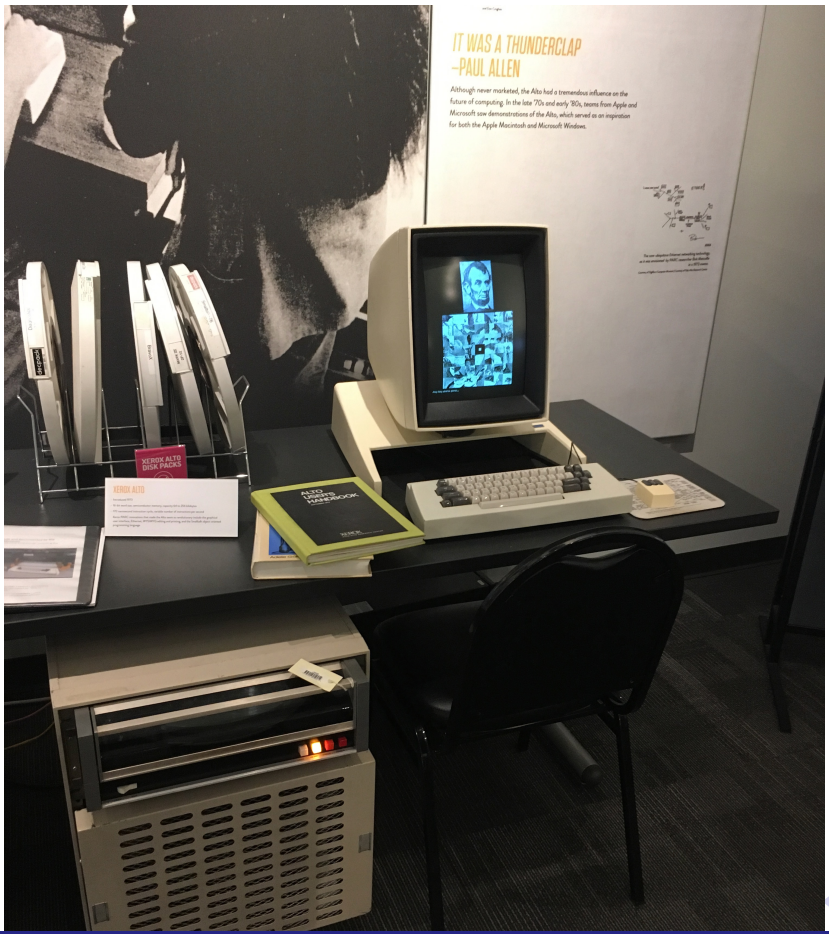
- Alan Kay – Smalltalk designer who'd later work for Apple and Disney, among other places.
- Adele Goldberg – Another major Smalltalk researcher who helped commercialize Smalltalk in the 1980s and 1990s.
- Dan Ingalls – Another major Smalltalk researcher who continues exploring GUI-driven object-oriented software development.
- Charles Simonyi – Inventor of the first word processing application: Bravo. Later moved on to Microsoft to invent Microsoft Word, which helped him become a billionaire.

History of Smalltalk – Alto

- In 1973 Xerox PARC researchers developed a machine called the Alto, which served as the vehicle for their research on Smalltalk and other projects.
- The Alto was one of the first personal computers to be manufactured, though it was internal to Xerox and thus wasn't commercially available.
- The Alto's CPU ran at 5.88MHz. The Alto had 96-512 kilobytes of memory, supported 2.5 megabyte storage devices, and had a 606x808 pixel display.
- According to the book *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*, the final cost of manufacturing an Alto was $12,000 (in 1973 dollars; $74,754 in 2021 dollars), roughly half a PARC researcher's salary at the time.

The Xerox Alto

I took this photo at the (now defunct) Seattle Living Computer Museum in 2019.



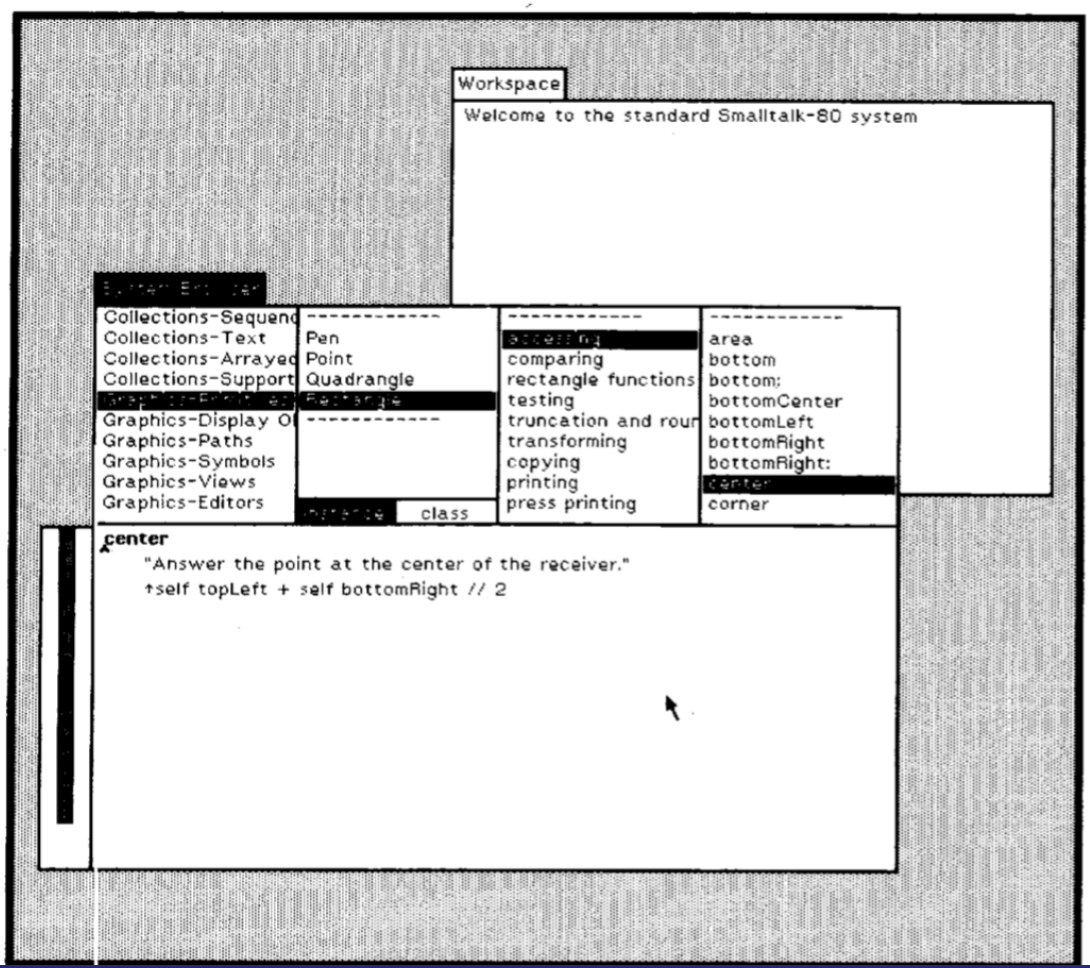
History of Smalltalk

- There were many versions of Smalltalk that were developed throughout the 1970s, such as Smalltalk-72 and Smalltalk-76.
- However, when most people talk about Smalltalk, they refer to *Smalltalk-80*.
- Smalltalk is an early example of an object-oriented programming language.
  - It is not the oldest, however; that designation goes to Simula-67, which influenced C++ and many other modern object-oriented programming languages.

Smalltalk, however, is more than just a programming language.

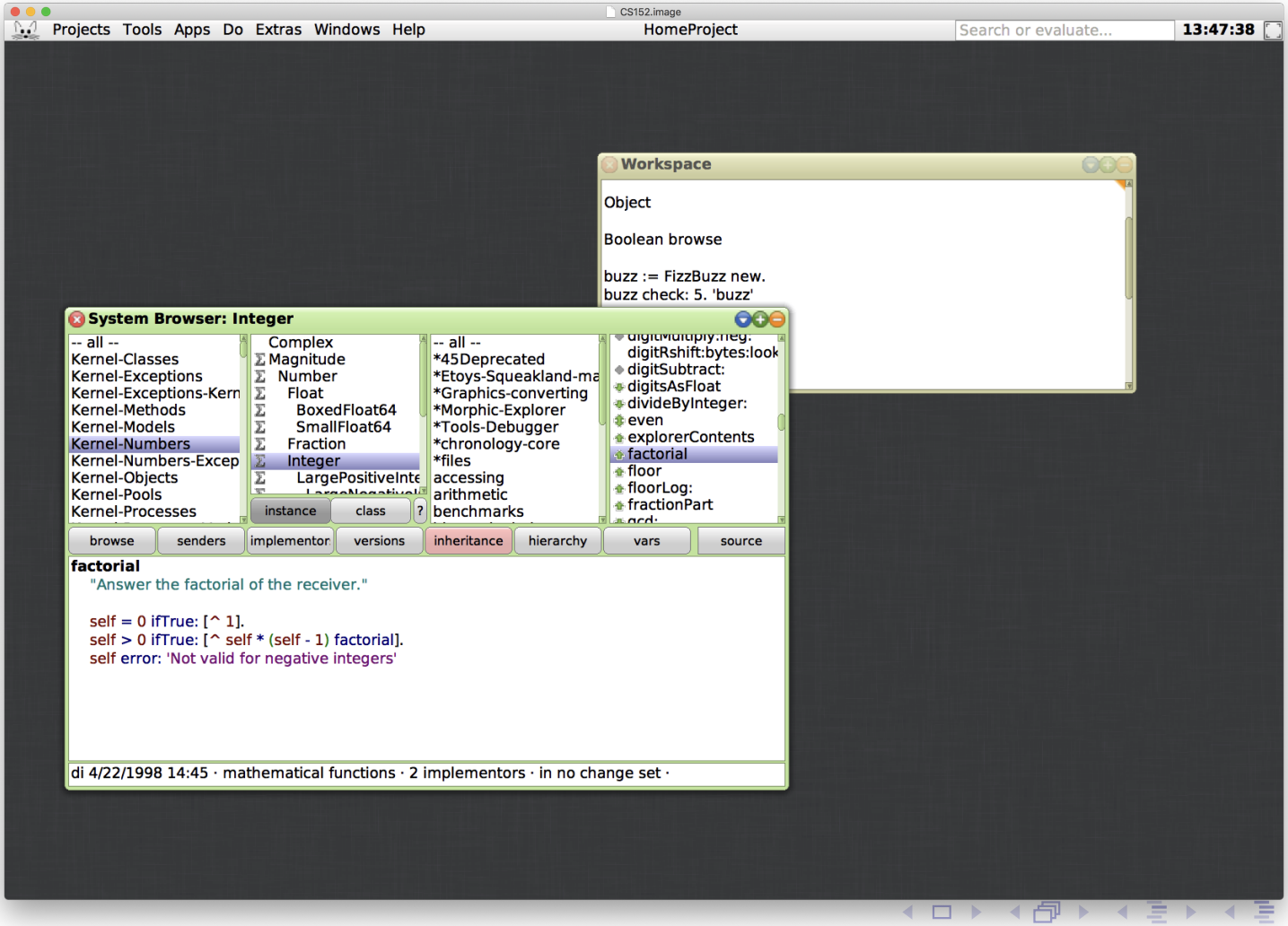
Smalltalk ran as an operating system directly on the Xerox Alto.

Smalltalk-80 Visual Environment (Goldberg and Robson p. 293)



To this day, the majority of Smalltalk implementations, such as Squeak and Pharo, run as virtual machines on top of another operating system. These virtual machines provide their own GUI environment that is separate from the host environment.

Squeak Visual Environment



In addition, in Smalltalk programs are not stored in files. Rather, the entire contents of the Smalltalk virtual machine's memory is stored on disk as an *image*.

However, not all Smalltalk implementations are self-contained like this. For example, GNU Smalltalk works like a conventional programming language.

In this class, we are going to focus less on Smalltalk in terms of the environment, and more in terms of the language itself.

An Aside: Some Apple History

- Steve Jobs and other Apple employees saw a demo of Smalltalk in 1979 at Xerox PARC. This demo, or, more specifically, Smalltalk's GUI development environment, was a direct influence on both the Apple Lisa (released in January 1983) and the Apple Macintosh (released in January 1984).
- You can see a 2017 demonstration of Smalltalk-76 from Dan Ingalls given at the Computer History Museum at `https://youtu.be/NqKyHEJe9_w`. This is the same system Steve Jobs saw.

Quote from "Steve Jobs: The Lost Interview" (1995, released in 2012)

They showed me really three things, but I was so blinded by the first one that I didn't really see the other two. One of the things they showed me was object-oriented programming, but I didn't even see that. The other thing they showed me was really a networked computer system ...; I didn't even see that. I was so blinded by the first thing they showed me, which was the graphical user interface. I thought it was the best thing I've ever seen in my life.

An Aside: Some Apple History

- When Steve Jobs left Apple in 1985, he founded a company called NeXT Computer, Inc.
- NeXT's operating system and development environment was inspired by Smalltalk. The main programming language was Objective-C, which adds Smalltalk-style object-oriented programming to C. NeXT's GUI development tools were modeled off Smalltalk's. NeXT also had support for Ethernet networking, a Xerox PARC invention.
- In December 1996 Apple purchased NeXT, and in mid-1997 Steve Jobs became interim CEO of Apple. NeXT's technology formed the basis of modern macOS. He became CEO in January 2000 and stayed at Apple until he resigned on August 24, 2011. He passed away on October 5, 2011.

Table of Contents

Introduction to Smalltalk

- Smalltalk is an object-oriented programming language.
- Smalltalk is dynamically-typed.
- Everything is an object in Smalltalk. Everything.

Unlike Java, where expressions call methods on an object (e.g., `shape.computeArea()`), in Smalltalk, we *send messages* to an object.

Unlike Java, where expressions call methods on an object (e.g., `shape.computeArea()`), in Smalltalk, we *send messages* to an object.

Example: `shape computeArea`, where `shape` is an object and `computeArea` is the message sent to the *receiver* `shape`.

An *interface* is the set of messages that the object accepts. In Smalltalk, all messages are `public` in Java terminology.

Literals in Smalltalk

- Numbers can be integers or floating-point (e.g., 3, -3, `-3.14159`)
- Octal and hexadecimal numbers are written with an `8r` or `16r` prefix (e.g., `8r31`, `16rBEEF`).
- A character has a `$` prefix (e.g., `$a`)
- Strings are delimited with single quotes (e.g., `'Hello World!'`)
  - If you use double quotes, you specify a code comment in Smalltalk. Be careful about this.
- Like Scheme, Smalltalk has symbols, which are preceded by `#` (e.g., `#hello`).

Arrays in Smalltalk

- The array `#(1 2 3)` is equivalent to `[1, 2, 3]` in Java.
- Arrays in Smalltalk may be heterogeneously typed; for example, `#(1 $b 'this is a test')` is a valid array of three elements.
- Arrays in Smalltalk may be nested; for example, `#(('one' 1) ('two' 2))` is an array containing two arrays where the first element is a string and the second element is a number. Note the lack of a # to denote the inner arrays.

Variable Names

- Private variables start with a lowercase letter.
- Shared variables start with an uppercase letter.
- These are not matters of style; these are enforced by the language environment.

Assignment

In Smalltalk, we assign a value to a variable by using the `:=` operator:

```
quantity := 19
```

- Please note that the textbook uses a left arrow ($\leftarrow$) for assignment.
- Also note that in Smalltalk variables are mutable.

Special Values in Smalltalk

- `nil` is equivalent to `null` in Java. All uninitialized variables in Smalltalk have the value `nil`.
- `true` and `false` refer to objects that represent the Boolean values true and false, respectively.

Messages

- Messages in Smalltalk are sent using a type of postfix notation.
  - e.g., `theta sin` (computes the sine of theta)
- If a method takes an argument, we use a colon to indicate the argument.
  - e.g., `list addFirst:  newComponent`
  - e.g., `HouseholdFinances spend:30.45 on:'food'`

Messages

Some messages use infix syntax

```
3 + 4
index > limit
```

Note that Smalltalk does not have operator precedence; rather, expressions are generally evaluated from left to right. For example, 3+4*5 results in 35 rather than the expected 23. We need to use parentheses to get the desired result (e.g., 3+(4*5)).

Example: Let's evaluate the expression `1.5 tan rounded`.

What happens when you send an undefined message to a Smalltalk object? For example, what happens when you run `5 wham` in Smalltalk?

In Smalltalk, undefined methods are runtime errors, not compile-time errors. This is what distinguishes dynamic object systems from static object systems like C++ and Java.

Cascading

Cascading is useful in situations where you want to send messages to the same object.

```
OrderedCollection new add: 1; add: 2; add: 3
```

Blocks

Blocks are an important, fundamental concept in Smalltalk.

Blocks

- Blocks are delimited by square brackets.
- Blocks contain sequences of expressions that are delimited by periods.
  - Periods are optional for the final expression.
  - If there is only one expression in the block, then no period is necessary.
- A block represents a *deferred* action; the block's actions are not executed immediately.

Examples of Blocks

```
[index := index + 1]

[tmp := x. x := y. y := tmp]
```

Executing a Block

To execute the sequence of expressions in a block, send the `value` message to the block.

```
[index := index + 1] value
```

Executing a Block

To execute the sequence of expressions in a block, send the `value` message to the block.

```
[index := index + 1] value
```

Remember that in Smalltalk, everything is an object.

Conditional Selection

This construct is analogous to an `if` statement in other programming languages:

```
(number \\ 2) = 0
  ifTrue: [parity := 0]
  ifFalse: [parity := 1]
```

Note that the above can be written as

```
parity := (number \\ 2) = 0 ifTrue: [0] ifFalse: [1]
```

Conditional Repetition

This construct is analogous to a `while` loop:

```
index := 1.
[index <= list size]
  whileTrue: [list at: index put: 0.
              index := index + 1]
```

There is also a `whileFalse`.

Block Arguments

Blocks can accept arguments.

```
[:array | total := total + array size]
```

Examples of do and collect Messages to Arrays

The do message is used for iterating over an array. For each element of the array, it passes it as an argument to the given block.

```
sum := 0.
#(2 3 5 7 11) do: [:prime | sum := sum + (prime * prime)]
```

Examples of `do` and `collect` Messages to Arrays

The `collect` message is analogous to `map` in Scheme.

```
#(2 3 5 7 11) collect: [:prime | prime * prime]
```

Blocks may take multiple arguments:

```
[:x :y | x + y]
```

Takeaway: Blocks in Smalltalk are very similar to anonymous functions in other languages. Blocks are used a lot in Scheme to handle deferred computations.

Implementing Methods in Smalltalk

To specify a return value in a method, we use ^.

```
totalSpentFor: reason
  (expenditures includesKey: reason)
    ifTrue: [^expenditures at: reason]
    ifFalse: [^0]
```

Note that a method does not have to return anything.

`self` in Smalltalk

`self` works similarly to Java's `self`. It can also be used in recursive functions in this very interesting manner:

```
"factorial is a method of Integer"
factorial
  self = 0 ifTrue: [^1].
  self < 0
    ifTrue: [self error: 'factorial invalid']
    ifFalse: [^self * (self - 1) factorial]
```

Temporary variables are defined inside bars.

```
spend:amount for:reason
  |previousExpenditures|
  previousExpenditures := self totalSpentFor: reason
  expenditures at: reason
               put: previousExpenditures + amount
  cashOnHand := cashOnHand - amount
```

Why haven't we discussed classes yet?

Table of Contents

Classes in Smalltalk

Believe it or not, Smalltalk does not have class files like Java. How classes and their associated methods are laid out in Smalltalk is an implementation detail.

We will cover how Squeak defines classes.

Defining Classes Using Squeak

```
Object subclass: #NameOfSubclass
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: ''
```

- In the above code, the class `NameOfSubclass` inherits from the base class `Object`.
- `instanceVariableNames`, `classVariableNames`, and `poolDictionaries` are strings containing names, where each name is delimited by a space
- `category` is a string that describes the category the class belongs in.

Difference between Instance Variables and Class Variables

- An instance variable is associated with an object.
- A class variables is shared among all instances of a class.
  - For example, suppose the class `GUIPoint` had a class variable called `Window`, which refers to the window the GUI point is drawn on. All GUIPoint objects in the system have access to the same `Window` object.

Details about Inheritance in Smalltalk

- All objects inherit from a superclass.
- The base class in Smalltalk is `Object`. All objects ultimately derive from `Object`.
- Unlike C++ and Common Lisp, Smalltalk does not support multiple inheritance; i.e., a class may only inherit one superclass in Smalltalk.

Unlike in static object systems such as C++ and Java, in Smalltalk it is possible to add methods to classes at runtime. This is very useful in systems where objects stay in the memory image for a long time but their behavior needs to be updated.

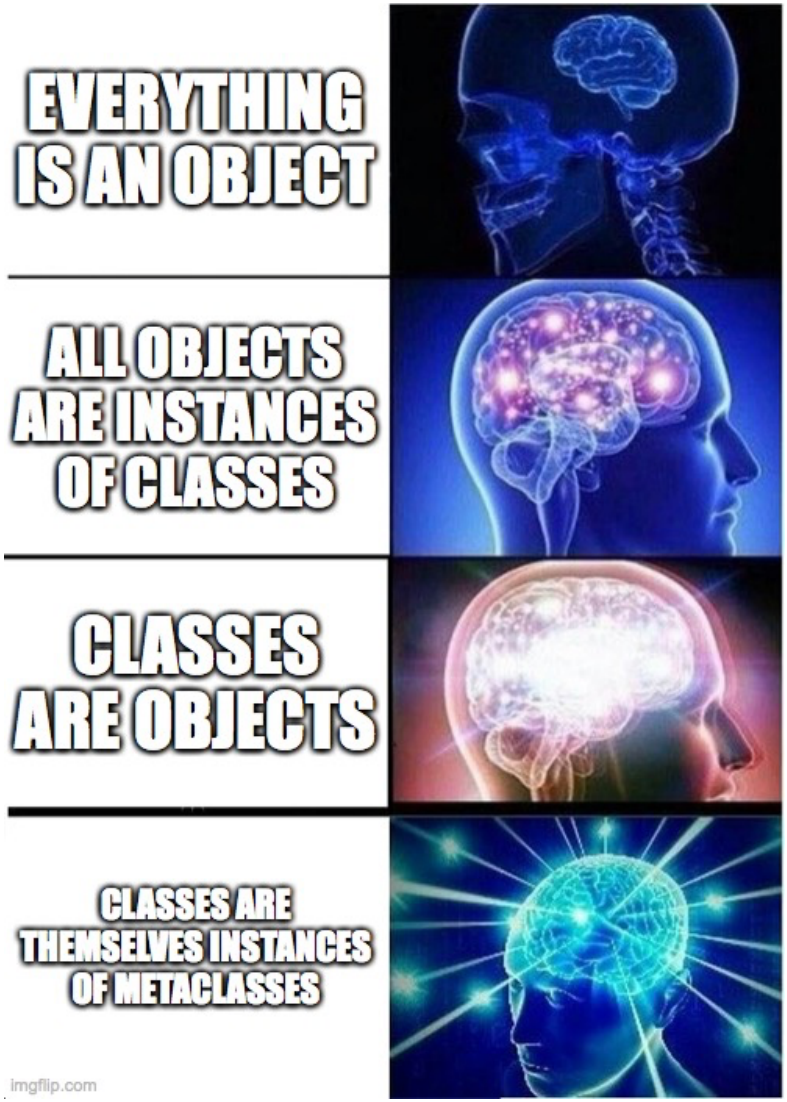
Table of Contents

- Everything is an object in Smalltalk.

- Everything is an object in Smalltalk.
- All objects are instances of classes.

- Everything is an object in Smalltalk.
- All objects are instances of classes.
- Because everything is an object, that means that classes are objects, too.

- Everything is an object in Smalltalk.
- All objects are instances of classes.
- Because everything is an object, that means that classes are objects, too.
- Because all objects are instances of classes, that means classes are themselves instances of classes.


EVERYTHING IS AN OBJECT
ALL OBJECTS ARE INSTANCES OF CLASSES
CLASSES ARE OBJECTS
CLASSES ARE THEMSELVES INSTANCES OF METACLASSES
imgflip.com

Metaclasses

- According to the textbook, "a class whose instances are themselves classes is called a metaclass" [Goldberg and Robson, p. 76].
- "Each class is an instance of its own metaclass." [p. 77].
  - "Whenever a new class is created, a new metaclass is created for it automatically" [p. 77].
- Metaclasses are instances of the class `Metaclass`.
- Metaclasses don't have their own names; you access a metaclass by sending the `class` message to the class (e.g., `Rectangle class`).

Class Methods and Instance Methods

- Class methods are equivalent to Java's `static` methods. They are associated with the metaclass.
- Instance methods are associated with instances of the class. When defining them, they are defined for the class, not the metaclass.
- There are no dedicated constructors in Smalltalk. Rather, you could create one or more class methods that call the `new` method, which is inherited from the `Object` base class.

Additional Details about Metaclasses

- If a class is inherited from a superclass, then the class's metaclass is inherited from the superclass's metaclass.
- "Class variables are accessible to both the class and its metaclass" [Goldberg and Robson, p. 84].

Table of Contents