Typology of programming languages

Smalltalk

Smalltalk

> *We called Smalltalk Smalltalk so that nobody would expect anything from it.*
> – Alan Kay

Principles:

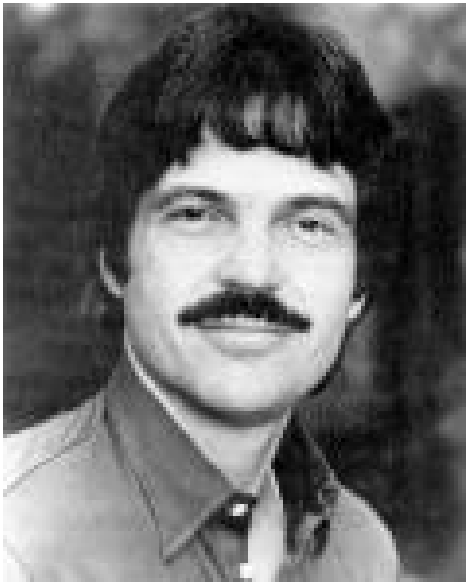
- Everything is object;
- Every object is described by its class (structure, behavior);
- Message passing is the only interface to objects.

Origin:

- A programming language that children can understand;
- To create “tomorrow’s computer”: Dynabook.

Table of Contents

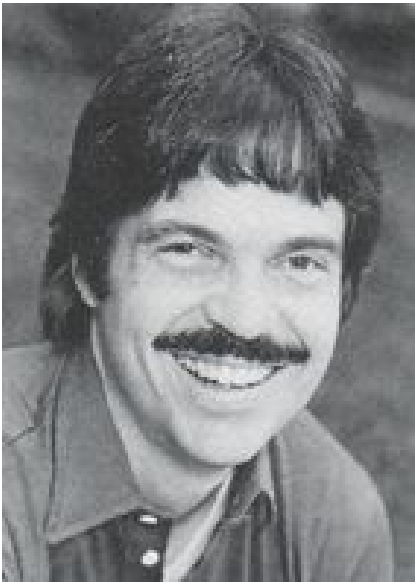
Alan Kay



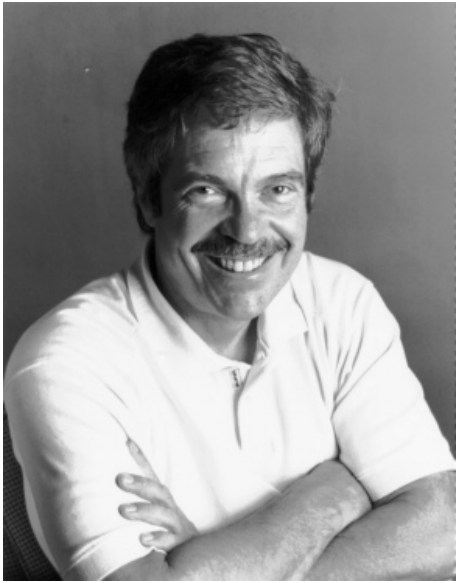
Quote

I invented the term Object-Oriented and I can tell you I did not have C++ in mind.
– A. Kay

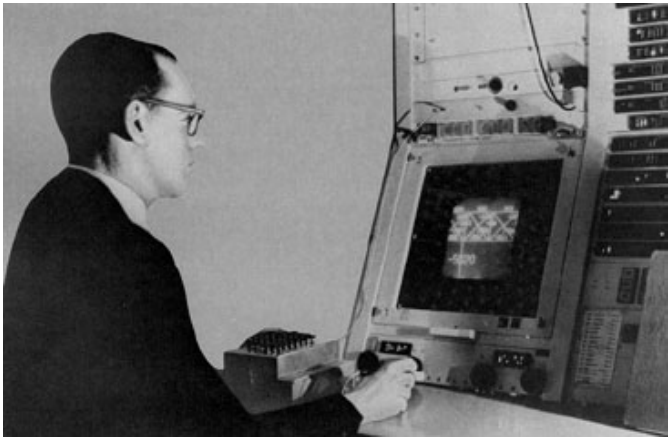
Alan Kay, 1984



Alan Kay

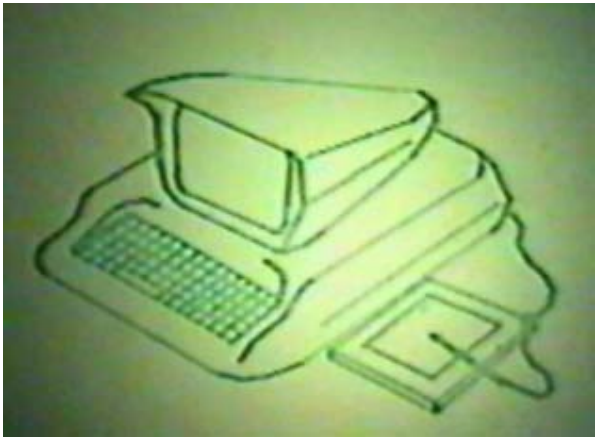


Ivan Sutherland's Sketchpad 1967

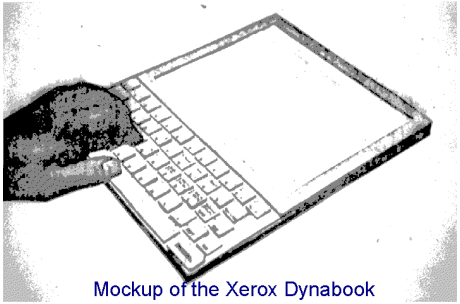


Douglas Engelbart's NLS 1974

Flex Machine 1967

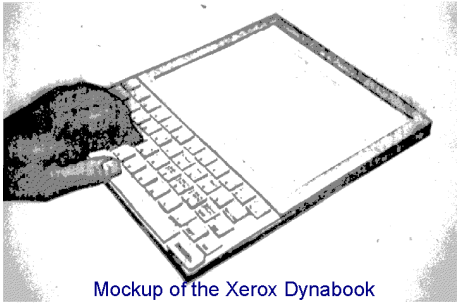


DynaBook


Mockup of the Xerox Dynabook

It would have, "enough power to outrace your senses of sight and hearing, enough capacity to store for later retrieval thousands of page-equivalents of reference material, poems, letter, recipes, records, drawings, animations, musical scores, waveforms, dynamic simulations, and anything else you would like to remember and change...".

DynaBook


Mockup of the Xerox Dynabook

It would have, "enough power to outrace your senses of sight and hearing, enough capacity to store for later retrieval thousands of page-equivalents of reference material, poems, letter, recipes, records, drawings, animations, musical scores, waveforms, dynamic simulations, and anything else you would like to remember and change...".

To put this project in context, the smallest general purpose computer in the early 1970s was about the size of a desk and the word "multimedia" meant a slide-tape presentation.

DynaBook

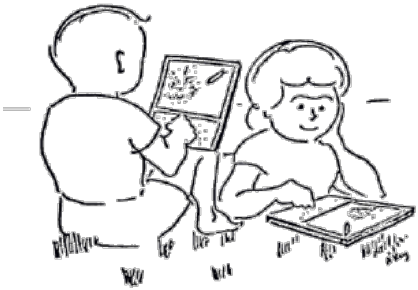


Table of Contents

Smalltalk 72

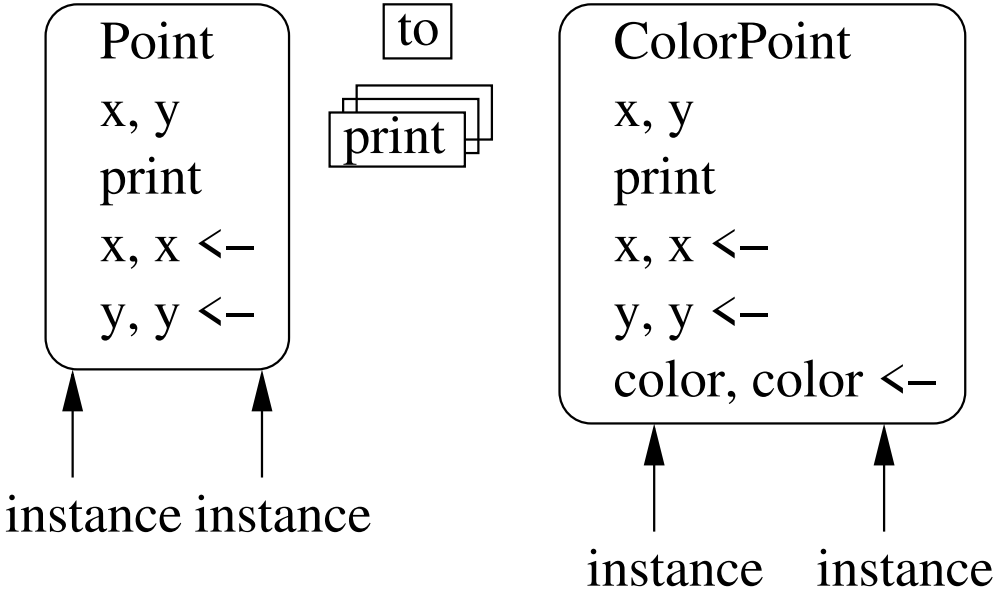
- Written in BASIC.
- Reuses the classes and instances from Simula 67.
- Adds the concept of “message”. Dynamic method lookup.

Smalltalk 72 Sample

```
to Point
  | x y
  (
   isnew => ("x <- :.
              "y <- :.)
   <) x  => ( <) <- => ("x <- : )
              ^ x)
   <) y  => ( <) <- => ("y <- : )
              ^ y)
   <) print => ("( print.
                 x print.
                 ", print.
                 y print.
                 ") print.)
  )
=> Point
```

```
center <- Point 0 0
=> (0,0)
center x <- 3
=> (3,0)
center x print
=> 3
```

Classes and Instances in Smalltalk 72



Smalltalk 72 Criticisms

- `to` is a primitive, not a method.
- A class is not an object.
- The programmer implements the method lookup.
- Method lookup is too slow.
- No inheritance.

$\Rightarrow$ Programmers were using global procedures.

But some successes:

- Pygmalion
  "Programming by examples"
  inspired Star.

Table of Contents

Smalltalk 76

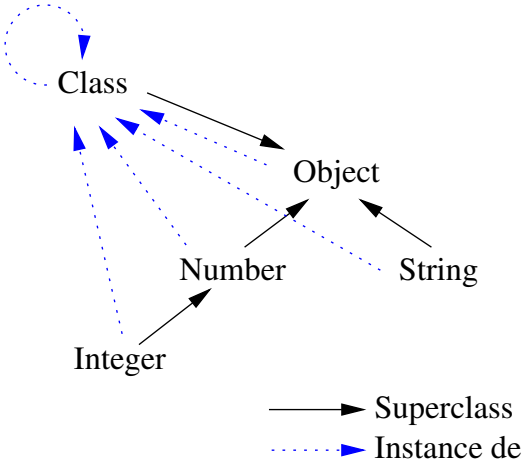
- Introduction of the `Class` class. The class of classes. Instance of itself. *Metaclass*. How to print a class, add method, instantiate etc.
- Introduction of the `Object` class. Default behavior, shared between all the objects.
- Introduction of dictionaries. Message handling is no longer handled by the programmers.

Smalltalk 76

- Introduction of inheritance.
- Removal of the `to` primitive.
  Replaced by the `new` message sent to `Class`:

```
Class new title: 'Rectangle';
fields: 'origin corner'.
```

Instantiation, inheritance in Smalltalk 76



- Objects keep a link with their generator: `is-instance-of`

Smalltalk 76 Criticism

- Significant improvement:
  - Byte-code and a virtual machine provide a 4-100 speedup.
  - *ThingLab*, constraint system experimentation.
  - *PIE*, *Personal Information Environment*.
- But:
  - A single metaclass
    hence a single behavior for classes
    (no specific constructors, etc.).

Table of Contents

Smalltalk 80

- Deep impact over computer science of the 80's.
- Most constructors take part
  (Apple, Apollo, DEC, HP, Tektronix...).
- Generalization of the metaclass concept.

Layer model -of in Smalltalk 80

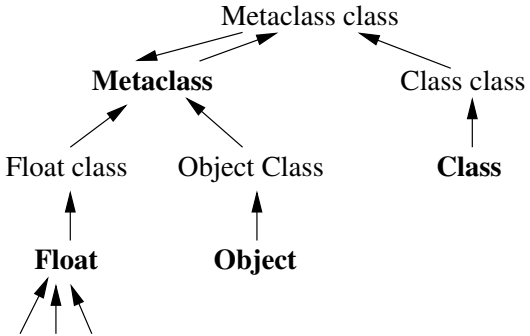
Three layer model:

Metaclass. Class behavior (instantiation, initialization, etc.).

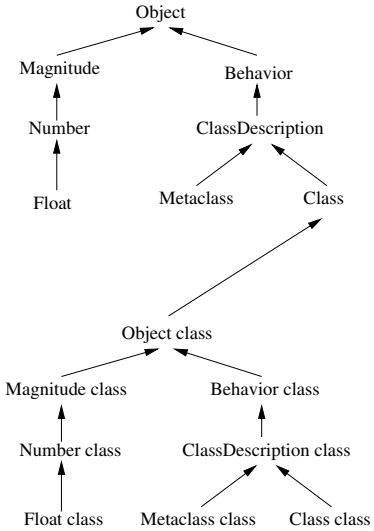
Class. Type and behavior of objects.

Instances. The objects.

Is-instance-of in Smalltalk 80



Inheritance in Smalltalk 80



The Smalltalk 80 System

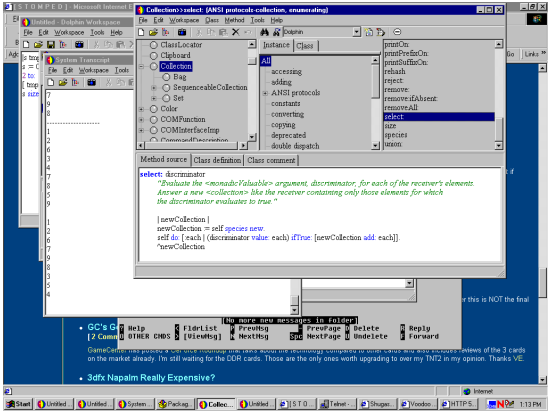
More than a language, a system where *everything* is an object, and the only control structure is message passing.

- a *virtual image*;
- a byte-code compiler;
- a virtual machine;
- more than 500 classes, 4000 methods, 15000 objects.

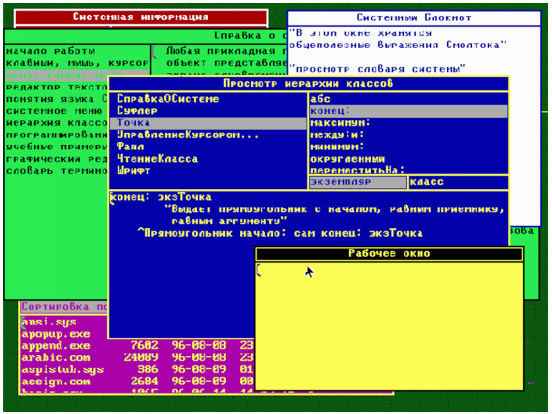
Smalltalk 80 Standard Library

- System
  `Class`, `Object`, `Number`, `Boolean`, `BlockContext` etc.
- Programming Environment
  `Model`, `View`, `Controler`, etc.
- Standard Library
  `Collection`, `Stream`, etc.
- Notable inventions
  `Bitmap`, `Mouse`, `Semaphore`, `Process`, `ProcessScheduler`

Smalltalk 80



Smalltalk 80



Booleans: Logical Operators

Boolean methods: `and:`, `or:`, `not:`.

- In the `True` class

```
and: aBlock
"Evaluate aBlock"
↑ aBlock value
```

- In the `False` class

```
and: aBlock
"Return receiver"
↑ self
```

Booleans: Control Structures

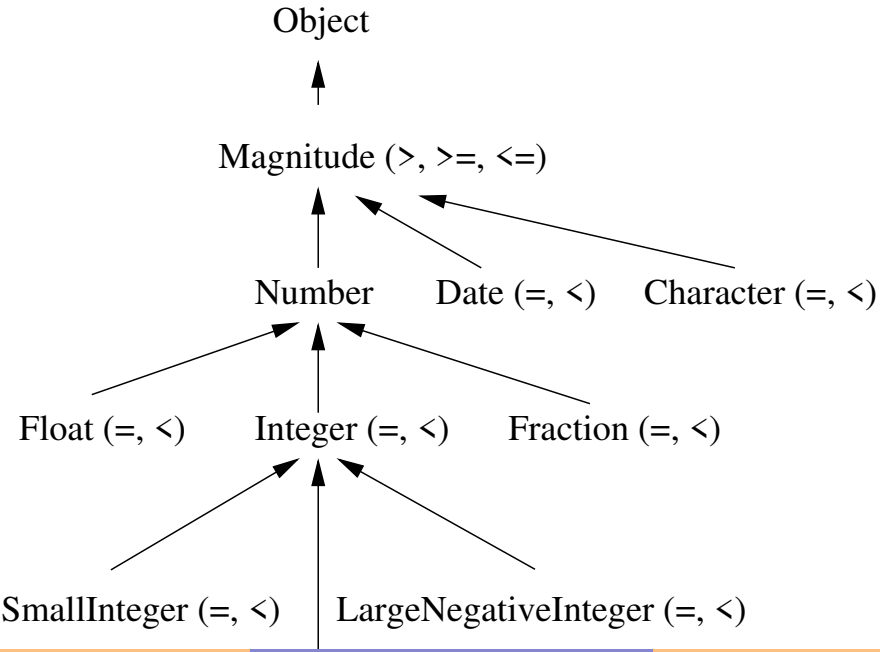
More Boolean methods:

- `ifTrue:`
- `ifFalse:`
- `ifTrue:ifFalse:`
- ... `ifFalse:ifTrue:`

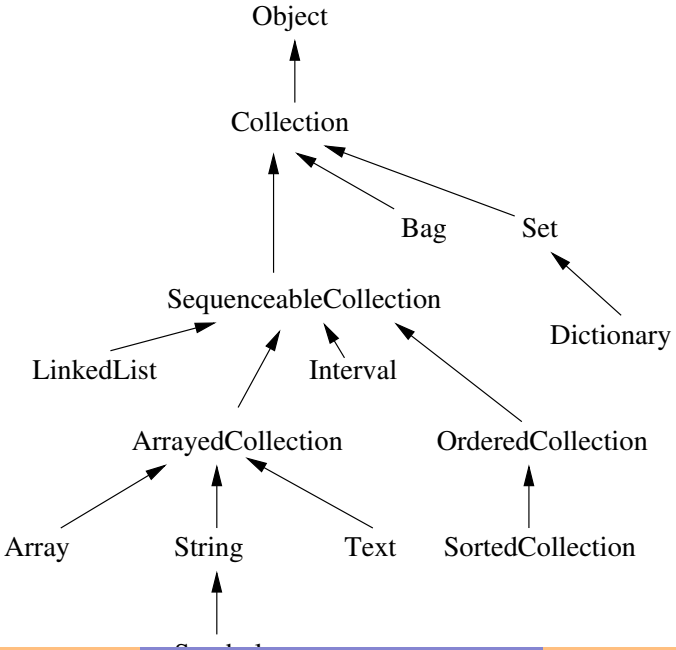
For instance, compute a minimum:

```
| a b x |
...
a <= b ifTrue:  [ x <- a ]
       ifFalse: [ x <- b ].
...
```

Integers in Smalltalk 80



Collections in Smalltalk 80



Collections in Smalltalk 80

In `LinkedList`:

```
do: aBlock
  | aLink |
  aLink <- firstLink.
  [aLink = nil]
      whileFalse:
        [aBlock value: aLink.
        aLink <- aLink nextLink]
```

The Smalltalk 80 Environment

- Everything is sorted, classified, so that the programmers can browse the system.
- Everything is object.
- The system is reflexive.
- The *inspector* to examine an object.
- Coupled to the debugger and the interpretor, a wonderful programming environment.
- Big success of Smalltalk in prototyping.

Smalltalk 80 Criticism

- Some loopholes in the semantics.
- The metaclass concept was considered too difficult.
- No typing!
- Dynamic dispatch exclusively, that's slow.
- The GC is nice, but slow too.
- The virtual image prevents collaborative development.
- No security (one can change *anything*).
- No means to produce standalone applications.
- No multiple inheritance.

Demo with Squeak

https://squeak.org

Summary