'The best way to predict the future is to invent it'

- Alan Kay.

Original and Remake

Source of Inspiration
History

- Influenced by Lisp and Simula
- Object-oriented
- Developed by XeroxPARC 1972–1980

Smalltalk is:

- a language
- a living system
- a philosophy of coding

Distributions

- Commercial distributions
  - Cincom Smalltalk (Visual Works)
  - Several others
- Open-source distributions
  - Squeak
  - Several others

The Language
Smalltalk Object Model

- Everything is an object
- Every object is instance of a class
- Methods are public
- Attributes are protected
- Single inheritance
- Closures

Pseudo-Variables

- Constants:
  - true, false
  - nil (default value of uninitialized attributes)
- Dynamic:
  - current receiver: self and super
  - thisContext (runtime stack)
- Pseudo-Variables: cannot be assigned

“Syntax”

- := assignment
- = comparison
- == identity comparison

```
. , ( ) [ ] ^ := " ' #{ } |
```

Sending Messages

- Syntax mimic natural language
- Objects are subjects
- Verbs are messages
- Arguments are complements
- Expression ends with a period

```
album playFromTrack: 2 to: 5
```
Keyword-based Messages

- 2 raisedTo: 10.
- 2 raisedTo: 10 modulo: 1000.
- array := #(1 2 3 4 5).
- array at: 1.
- array at: 1 put: 'something'.
- array at: 1.

Sending Messages (contd)

- Any computation is performed by message passing:
  - We send messages to objects
  - Methods with same name are executed

  \(<receiver>\) method-name \(<arg>\)
  album play
  album playFromTrack: 1
  album playFromTrack: 2 to: 6

Three Kinds of Messages

- Unary: 'squeak' reversed
- Binary: 1+2
- Keyword-based:
  - Color r: 0 g: 1 b: 0

  Determine the execution order
  - Unary > Binary > Keyword-based

Chaining Messages

- Messages with same priority are executed from left to right

  2 squared reciprocal negated.
  2 squared reciprocal
  2 squared
  2 + 3 * 4 / 2
  2 + 3 * 4
  2 + 3
Cascade

• Sending multiple messages to the same object

| p |
p := Person new
name: 'Adele';
age: 62;
address: 'Earth'.

Getters and Setters

age

^age

age: newAge
age := newAge

Literal Objects

• Created at compile-time by the parser

12
3.14156
$a
'Hello World'
#show:
#(12 'abc' $b)

Block Closures

• Represent piece of code
• Delay expressions

[30 squared] value
[:x | 3 + x] value: 5

| myBlock |
myBlock := [Transcript show: 'Block!'].
...
myBlock value.
**Conditions**

7 < 12
  ifTrue: [Transcript show: '7 is less than 12'; cr].
7 < 12
  ifFalse: [Transcript show: '7 is less than 12'; cr].
7 odd
  ifTrue: [Transcript show: '7 is odd'; cr]
  ifFalse: [Transcript show: '7 is even'; cr].

**Conditions, contd.**

[0]
  ifTrue: [Transcript show: '0 is true'].
[nil]
  ifTrue: [Transcript show: 'nil is true'].
  [nil = nil] "equality"
  ifTrue: [Transcript show: 'nil is true'].
  [nil == nil] "identity"
  ifTrue: [Transcript show: 'nil is true'].
[false]
  ifTrue: [Transcript show: 'false is true']
  ifFalse: [Transcript show: 'false is false'].

**Booleans**

- Boolean is abstract
- True and False are its subclass
  - true instance of True
  - false instance of False
- or, not, and are polymorphically implemented on True and False

**Iterations - While True**

<table>
<thead>
<tr>
<th>f n</th>
</tr>
</thead>
<tbody>
<tr>
<td>f := 1. n := 4.</td>
</tr>
<tr>
<td>[ n&gt;1 ]</td>
</tr>
<tr>
<td>whileTrue: [f := f*n.</td>
</tr>
<tr>
<td>n := n-1].</td>
</tr>
<tr>
<td>f.</td>
</tr>
</tbody>
</table>
Iterations - While True

| count |
count := 0.
[count < 100]
whileTrue: [count := count + 1].
Transcript clear; show: count printString

High-Level Iterators

#(1 2 3 4) do: [:x| Transcript show:
  x printString; cr]

#(1 2 3 4) collect: [:x| x < 3]
#(1 2 3 4) select: [:x| x > 2]

Other Iteration Constructs

Transcript clear.
3 timesRepeat: ['Transcript cr; show: 'Testing!'
  Screen default ringBell]

1 to: 3 do: [:n | Transcript cr;
  show: n printString; tab;
  show: n squared printString]

Syntax on a PostCard

eByExampleWithNumber: x
  |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: (each printString); cr].
  ^ x < y
What is a class?

- All classes are subclasses of Object class
- There are two kinds of objects in the system
  - objects that can create instances of themselves (classes)
  - "normal" objects
- Classes are used in:
  - creating new instances
  - defining what the instances of a class do
  - holding class information (class variables)

Classes are Objects Too

- Classes are objects too
  - 5 class inspect.
  - String allInstances.
- A class is instance of another class, its metaclass
  - 5 class class.
- Class methods – methods executed on classes
  - Date today.

Classes

<table>
<thead>
<tr>
<th>klass p</th>
</tr>
</thead>
<tbody>
<tr>
<td>klass := Person.</td>
</tr>
<tr>
<td>p := klass new.</td>
</tr>
<tr>
<td>p class inspect.</td>
</tr>
</tbody>
</table>
Classes

Smalltalk defineClass: #NameOfClass
  superclass: #{NameOfSuperclass}
  indexedType: #none
  private: false
  instanceVariableNames: 'instVarName1 instVarName2'
  classInstanceVariableNames: ''
  imports: ''
  category: 'PostgreSQLVWCompatibility'

Meta Classes

• How are classes represented at runtime?
  - Not at all
  - As objects (class objects)
• What is the class of a class object?
  - The meta class
  - Changing the meta class will change what it means to be a class
• Control aspects of classes
  - Binding
  - Synchronisation
  - Instantiation
  - Memory (de)allocation

Meta Classes

• 1 Level System
  - All objects can be viewed as classes and all classes can be viewed as objects (as in Self). “Single–hierarchy”.
• 2 Level System
  - All Objects are instances of a Class but Classes are not accessible to programs. 2 kinds of distinct objects: objects and classes.
• 3 Level System
  - All objects are instances of a class and all classes are instances of Meta–Class. The Meta–Class is a class and is therefore an instance of itself. 2 kinds of distinct objects (objects and classes), with a distinguished class, the metaclass.

An object is an instance of...

Class
Class class
Object
Object class
Anita
Person
Metaclass
Metaclass class
Reflection

• The ability in a program to "observe" itself, and to modify itself -- keeping meta-data about program at run-time
  - Fields and methods
  - What's my class?
  - Manipulate methods, classes etc.
  - Dynamically create & load classes
• Meta Object Protocol
  - a method for accessing the details of an object system through a meta class.

Reflection

• Introspection is the ability of a program to observe and therefore reason about its own state.
• Intercession/modification is the ability of a program to modify its own execution state or alter its own interpretation or meaning. Ability to add new behaviour to program as it is executing.
• Reification is a mechanism for encoding execution state as data -- regarding something abstract as a material thing
• Structural reflection is the ability to provide reification of the program currently being executed and of its abstract data types.

Reflective system

• The Smalltalk environment is implemented in Smalltalk
• The language’s structure is defined by using Smalltalk objects
• The Smalltalk system is a "living" system with the ability to extend itself at run-time
**Introspection**

- With introspection the language is available for itself, but cannot change its own semantics
- Smalltalk has a big collection of introspective facilities
- With full introspection:
  - a program is implemented in terms of the self representations it employs
  - openness or open implementations

**Object Introspection**

```smalltalk
1 isKindOf: Integer
1 class
Object class
1 respondsTo: #+
myObject allOwners
((Smalltalk class) inheritsFrom: Object) inspect.
```

**Introspection in Smalltalk**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>subclasses</td>
<td>immediate subclasses</td>
</tr>
<tr>
<td>superclass</td>
<td>parent class</td>
</tr>
<tr>
<td>inheritsFrom:</td>
<td>test ancestry</td>
</tr>
<tr>
<td>respondsTo:</td>
<td>test for message selector</td>
</tr>
<tr>
<td>instSize</td>
<td>number of instance</td>
</tr>
<tr>
<td>instVarNames</td>
<td>variables defined in class</td>
</tr>
<tr>
<td>allInstVarNames</td>
<td>includes inherited variables</td>
</tr>
<tr>
<td>selectors</td>
<td>methods defined in class</td>
</tr>
<tr>
<td>allSelectors</td>
<td>includes inherited methods</td>
</tr>
</tbody>
</table>

**Reflection**

- With reflection, a program can change the language in which it was written
- Smalltalk is constructed in most parts by objects available and changeable at run-time
  - Editing the objects and metaobject, creating new metaclasses etc. can be done at runtime.
  - Methods can be edited, added, removed...
  - "Everything is an object" gives that everything can be edited including runtime environment, compiler/interpreter, class/object behaviour...
Dangerous?

Frequently Asked Questions

Q: Isn’t reflection dangerous?
A: Yes! You bet it is!
A: Yes, if you are not careful.
A: Yes, but you can make it safer.
A: Yes, but so is crossing the street.

Open Implementation

- Making it possible to make additions to the abstract syntax of a program
- Representing programs as data
- Making the compiler available at runtime

Why Reflection?

- Reflection brings flexibility
  - Hacking all over your program, or
  - Hacking the interpreter
- Adding new concepts without “disruption”
- Different languages offer reflection mechanisms of different power
  - None: C, C++
  - Low: Java, C# (?)
  - High: LISP, Smalltalk, Ruby, Python

Working Environment
Powerful Environment

- Incremental compilation
- Compiler accessible in any pane
- Complete environment written in itself
- Excellent debugger

Basic Tools

- Workspace
- Transcript
- Explorer
- System Browser

Workspace

- Place to execute expressions
  - Do it:
    - Execute an expression
  - Print it:
    - Execute an expression and print its result

Transcript

- Kind of std-output

Transcript show: 30.
Transcript show: 30; cr.
Transcript show: 30 factorial; cr.
30 to: 40 do:
  [:i | Transcript show: i factorial; cr].
Transcript clear.
**Inspector and Explorer**

- Getting access to the objects
- Accessing/Modifying state
- Sending messages to inspected object
  ```
  100 inspect
  100@100 corner:200@200
  ```

**System Browser**

- Browse and create classes
- Edit methods
- 5 Panes:
  - Class categories (class folders)
  - Classes
  - Method categories
  - Methods
  - Method body

**Smalltalk Drawbacks**

- Different Dialects
  - Different UIs
  - Slight difference but core the same
- Uniform language
  - Not conventional syntax
- Not mainstream

**Read more**

- Cincom Smalltalk, [www.cincomsmalltalk.com](http://www.cincomsmalltalk.com/)
- Squeak, [www.squeak.org](http://www.squeak.org)
Programming Assignment

The Castle
- Four entrances
- Consists of a number of rooms connected by doors or stairs
- Non-limited number of floors
- Entrance floor and the floors above are limited by the square shaped walls of the castle
- Labyrinth inside the castle is randomly created for every game

Rooms
- A room can have 1-6 exits -- open or locked doors, stairs or trap bars
- A corridor is a room where you cannot stop
- In the rooms you can find different items and/or creatures

Creatures
- There are adventurers exploring the castle and creatures living in the castle
- There should be a possibility to interact with the adventurer
- The creatures make their own decisions
Creatures, contd

- Hostile or friendly
- Elfs, orcs, dwarves, halflings and humans
- Not bound to rooms
- All creatures and players have physical and psychic strength

Items

- All items have a value in gold
- Some items can be combined with other items and the value is bigger than the sum of values of the two original items
- Moreover, all items have an attack value

Fighting

- Before you can see if there are any items or exits in a room you have to fight any present creatures
- There is no obligation to fight
- All fights go on until one of the fighters is dead