Smalltalk

Yongqi Jia [*], Jiaxu Kang [†]

April 21, 2020

1 Abstract

2 Introduction

3 History

3.1 Why was the language designed?

The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone. Our work flows from a vision that includes a creative individual and the best computing hardware available. We have chosen to concentrate on two principle areas of research: a language of description (programming language) that serves as an interface between the models in the human mind and those in computing hardware, and a language of interaction (user interface) that matches the human communication system to that of the computer.

3.2 Who designed it?

Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Diana Merry, Scott Wallace, and others

3.3 What is its current status?

While Smalltalk is not “popular” today, it is certainly still widely used.

4 Control Structures

Smalltalk is the first computer language based entirely on the concepts of objects and messages. Therefore, in smalltalk, everything is an object, even 3, true, nil, activation records.[6]

[*]Email: yongqijia@email.arizona.edu
[†]Email: jiaxukang@email.arizona.edu

"Control structures do not have special syntax in Smalltalk. They are instead implemented as messages sent to objects."[4] control structures all handled by message sending. So that's meaning we do not need to built-in control structures. Because of this, we can combine any objects together to create control structures.[5] In terms of that, I'll introduce several objects such as, ifTrue, whileTrue and exception.
ifTrue(same as ifFalse):

```
@condition ifTrue: [— '@temps — '@.statements]
        expr ifTrue: [statements to evaluate if expr]
```

whileTrue:

```
 — '@temps —
 '@.Statements1.
['index ¡= '@stop]
   whileTrue:
[— '@blockTemps —
 '@.BlockStmts1.
'index := 'index + 1].
 '@.Statements2'
```

Exception:

```
      '@block
    on: 'exception
do: [ :'@err — — '@temps — ]
```

This is some codes that I write:

```
IfPrint

x:=0 .

x=1 ifTrue: [ Transcript cr; show: 'this will not printed.' ]

ifFalse: [ Transcript cr; show: 'this will be printed.' ]
```

Everything in Smalltalk is an object. So IfPrint is a method in a class named "P2MyIf". And I give x value first. Then giving some expression before using if function like x=1. ifTrue and ifFalse also are subclasses in known packages. We just call and use it.

5 Data Type

5.1 Smalltalk is Un-Typed

Object-Oriented language like Java, each variable must be declared, and more importantly given a type. Things are different in Smalltalk, every variable must be declared, but it is not given a type. The main reason for that is Smalltalk is consist of objects, and objects can not have a type.

5.2 Declaration

Smalltalk has an unusual syntax for variable declarations: the variables are simply listed enclosed in vertical bars. And also, there is another way to declare variables is that using the key word "instanceVariableNames" and write the variables' name after this key word. For example:

```
int x, y, z;                          // Java
instanceVariableNames: 'a b c'        "Smalltalk"
```

Why bother to declare variables at all, if there is no typing information? The primary answer is that every variable must be declared so that the compiler can set aside space for each variable. Every variable will be allocated 32 bits and can hold either a pointer to an object or a SmallInteger.

5.3 Dynamic Typing

Sometimes, the un-typed language like Smalltalk are said to be "dynamically type", since the compiler does not check the types of variables but each value has a type. Object-Oriented language like Java is said to be "statically type". At compile time, the compiler will checks to make sure every line of the program is type-correct. And at runtime, there is no need to check the type.[2]

5.4 Assignment Statements

The assignment statement is simple in Samlltalk, and Smalltalk always has a simple variable name to the left of the assignment operator. The syntax is:

variable := expression

There are some examples:

```
i := i + 1
x := 8
x := y := z := 7
```

5.5 Type Conversions

In Java, the warp class of primitive data type provides some parse method which could convert the data type. Here are some examples:

```
String a                        // String
```

int b = String.ParseInt(a)   // Int

In Smalltalk, there is only one way to convert data from one representation to another, and this is with message-sending. In Smalltalk, there is a unary message for doing each kind of conversion. The tradition is to give these methods names beginning with "as..." such as "asFloat" or "asString".[2]
instanceVariableNames: 'd i s'
d := i asFloat.
i := d asInteger.
s := i asString.

5.6 Exceptions and Signals

[3] Smalltalk provides a powerful and flexible mechanism to handle exceptions and errors in a graceful way. Errors raise a so called Signal, which can be handled by a handlerblock. Especially with respect to instance creation, deletion and garbage collection, Smalltalk exceptions are very clean and easy to use. Notice: this is an old-style (but still supported) mechanism for instance based exception handling. Signals are more lightweight in that they do not need an Exception class to be defined. I.e. you can create and handle such signals dynamically without the overhead of creating a class for it. Here is the syntax:

1. Signal access:
   - Number arithmeticSignal
   - Number divisionByZeroSignal
   - Number Object errorSignal
2. handling signals/exceptions:
   - a [...some computation ...] on:aSignalOrExceptionClass do:[... handlerBlock ...]
   - aSignalOrExceptionClass handle:[... handlerBlock ...] do:[...some computation ...]
   - aSignalOrExceptionClass catch:[...some computation ...]

Here is a example:

1. To raise an exception:
   - MyException signal.
   - MyException signal: 'With an error message'.
2. To handle an exception:

1 / 0 on: ZeroDivide do: [Transcript showln: 'Oops! Zero divide!'].
1 / 0 on: Error do: [:e — Transcript showln: 'Oops! ' , e className , '!'].

6 Subprograms

6.1 Class

The information is shared by grouping together those objects that represent the same kind of entity into what is called a class. Every object in an object–oriented programming system is a member of a single class — it is called an instance of that class. Furthermore, every object contains a reference to the class of which it is an instance. The class of an object acts as a template to determine the number of internal variables an instance will have, and holds a table of methods (a method dictionary) which corresponds to the messages to which all instances of the class will respond. [1]

6.2 Class Hierarchy

Classes are arranged in a class hierarchy. Every class has a parent class — or superclass — and may also have subclasses. A subclass inherits both the behaviour of its superclass (in terms of its method dictionary), and also the structure of its internal variables. At the top of the hierarchy is the only class without a superclass, called class Object in Smalltalk. Class Object defines the basic structure of all objects, and the methods corresponding to the messages to which every object will respond. [1]

6.3 Method

A message is sent to an object which is an instance of some class. A search is made in the class's method dictionary for the method corresponding to the message selector. If the method is found there, then it is bound to the message and evaluated, and the appropriate response returned. If the appropriate method is not found, then a search is made in the instance's class's immediate superclass. This process repeats up the class hierarchy until either the method is located or there are no further superclasses. In the latter case, the system notifies the programmer that a run–time error has occurred. [1]

6.4 Code Example

```
Object subclass: #Dog
   instanceVariableNames: 'hairColor age'
   classVariableNames: ''
   package: 'P4_feature'!

"-------------------- part1 --------------------"

!Dog methodsFor: 'initialization' stamp: 'a 4/20/2020 18:18'!
initialize
   super initialize .
   hairColor :='red'.
   age := 2 .
   ! !

"-------------------- part2 --------------------"

!Dog methodsFor: 'accessing' stamp: 'a 4/20/2020 18:19'!
hairColor
^ hairColor ! !

!Dog methodsFor: 'accessing' stamp: 'a 4/20/2020 18:21'!
hairColor: color
hairColor := color! !

!Dog methodsFor: 'accessing' stamp: 'a 4/20/2020 18:36'!
age: ages
age := ages
! !

!Dog methodsFor: 'accessing' stamp: 'a 4/20/2020 18:30'!
age
^ age! !

"-------------------- part3 --------------------"

!Dog methodsFor: 'as yet unclassified' stamp: 'a 4/20/2020 18:42'!
ageInHuman
   "this method return the age of the dog in human years"

   | diff |
   diff := 7 .
^ age * diff! !

"-------------------- part4 --------------------"

"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

Dog class
   instanceVariableNames: 'age diff'!


Dog subclass: #SmallDiff
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'P4_feature'!

"-------------------- part5 --------------------"

!SmallDiff methodsFor: 'initialization' stamp: 'a 4/20/2020 18:49'!
initialize
   super initialize.
   hairColor := 'golden' .! !

"-------------------- part6 --------------------"

!SmallDiff methodsFor: 'as yet unclassified' stamp: 'a 4/20/2020 18:50'!
ageInHuman
   "this method return the age of the dog in human years"

   | diff |
   diff := 6 .
^ age * diff! !
```

"part1" is to create a dog class which has two variables, hairColor and age. And give this class was created in the package which name is "P4_feature".

"part2" is to initialize this class with two initialized information.

"part3" is the getter and setter methods for this class.

"part4" is a method for this class.

"part5" is to create a subclass of dog class.

"part6" using the keyword "super" to initialize this subclass.

7 Summary

8 References

References

[1] module01.pdf. (Accessed on 04/20/2020).

[2] Smalltalk: A white paper overview. (Accessed on 04/13/2020).

[3] Smalltalk/x basic classes - exceptions. (Accessed on 04/13/2020).

[4] Smalltalk, Mar 2020.

[5] Adele Goldberg and L Peter Deutsch. Smalltalk. *Encyclopedia of Software Engineering*, 2002.

[6] Fred Rivard. Smalltalk: a reflective language. In *Proceedings of REFLECTION*, volume 96, pages 21–38, 1996.