Smalltalk

Yongqi Jia [*], Jiaxu Kang [†]

April 30, 2020

1 Abstract

We are going to introduction an object-oriented programming, smalltalk. In next following sections, you will see why makes smalltalk special and what are the special aspects. And also we will provide some example code to demonstrate how to write a piece of code using smalltalk and what is the syntax of samlltalk.

2 Introduction

The Smalltalk-80 system is an object-oriented programming language and interactive programming environment. The Smalltalk-80 language includes many of the most difficult-to implement features of modern programming languages: dynamic storage allocation, full upward funargs, and call-time binding of procedure names to actual procedures based on dynamic type information, sometimes called message-passing. The interactive environment includes a full complement of programming tools: compiler, debugger, editor, window system, and so on, all written in the Smalltalk-80 language itself. A detailed overview of the system appears in [SCG 8l]. [Goldberg 83] is a technical reference for both file non-interactive programmer and the system implementer; [Goldberg 841 is a reference manual for the interactive system. [6]

3 History

3.1 Why was the language designed?

Smalltalk was created as the language underpinning the "new world" of computing exemplified by "human–computer symbiosis".[2]

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

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.[8]

"Control structures do not have special syntax in Smalltalk. They are instead implemented as messages sent to objects."[5] 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.[7] 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:
\begin{center}
    | ‘@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

In an object-oriented language like Java, each variable must be declared and, more importantly, a type must be given. In Smalltalk, the situation is different, each variable has to be declared, but no type has been specified for it. The main reason for this is that Smalltalk consists of objects and objects cannot have types.

5.2 Declaration

Smalltalk has an unusual syntax for variable declarations: variables are simply listed and enclosed in vertical bars. Also, another way to declare a variable is to use the keyword "instancevariablenames" and write out the name of the variable after that key word. For example:

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

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.[3]

5.4 Assignment Statements

The assignment statement is very simple in samlltalk. And smalltalk always has a simple variable name on the left side 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\hspace{40mm}        // 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".[3]

```
instanceVariableNames: 'd i s'
d := i asFloat.
i := d asInteger.
s := i asString.
```

5.6 Exceptions and Signals

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:[4]

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

Information is shared by combining objects that represent entities of the same kind called classes. Every object in an object-oriented programming system is a member of a single class - an instance of that class. In addition, each object contains a reference to its class as an instance. The class of the object is used as a template to determine the number of internal variables an instance will have, and holds the method table (method dictionary) corresponding to the message that all instances of the class will respond to. [1]

6.2 Class Hierarchy

Classes are arranged in a class hierarchy. Each class has a parent (or superclass) and may have children. Subclasses inherit not only the behavior of their superclasses, but also the structure of their internal variables. At the top of the hierarchy is the only class without a superclass, which is called the class object in Smalltalk. Class objects define the basic structure of all objects and the methods corresponding to the messages that each object will respond to. [1]

6.3 Method

A message is sent to an object that is an instance of a class. Search the class's method dictionary for the method corresponding to the message selector. If the method is found here, it is bound to the message, evaluated, and returned with the appropriate response. If the appropriate method is not found, the search is performed in the direct superclass of the instance's class. This process repeats

the class hierarchy until the method is found or there are no other superclasses. In the latter case, the system informs the programmer that a runtime 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

My partner and I had already tried our best to introduce this OOP language, samlltalk. There are still many people devoted to the development of computer

languages to solve problems in order to make computer languages more professional and effective. Thank you for your efforts. One more thing I want to talk about it is that the world is experiencing the COVID-19, I hope that people around the world can work together to make the world a better place.

8 References

References

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

[2] Smalltalk - wikipedia. (Accessed on 04/30/2020).

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

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

[5] Smalltalk, Mar 2020.

[6] L Peter Deutsch and Allan M Schiffman. Efficient implementation of the smalltalk-80 system. In *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 297–302, 1984.

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

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