Smalltalk

“Everything is an object.”

Joni Saarelainen

Group 2

TIE-20306 Principles of Programming Languages
Introduction

Smalltalk is a programming language based primarily on the idea of message passing. It was designed as a small, easy to learn language compared to the alternative languages of the time. Development of Smalltalk began by in the early 1970s at Xerox PARC’s (Palo Alto Research Center Incorporated) Learning Research Group, led by Alan Kay, which makes it roughly as old as C. Smalltalk is considered one of the first object-oriented programming languages, and one of the most influential: most modern languages, such as C++ and Java have mimicked many of the concepts of Smalltalk. As an object-oriented language the center focus in its development was the concept of objects. “Everything is an object” is a common phrase in Smalltalk.

Smalltalk in itself is technically not just a programming language, but a complete operating system (of course, the operating systems in the 1970s were simpler than today). It was also the first truly Integrated Development Environment (IDE).

In this document the latest stable version (Smalltalk-80 version 2) is always considered when referring to Smalltalk as a programming language.

Features

Smalltalk is based on three main ideas: the concept of objects, sending messages to objects and receiving messages from other objects.

Objects
The concept of objects is considered Smalltalk’s most important aspect. Smalltalk is considered a “pure” object oriented language. This means that even the basic primitive types, such as integers and characters, are handled as objects. This even allows for overriding the implementation of these core features.

Messages
Following the mindset of object oriented programming, all procedures in Smalltalk are defined for objects. The key concept in handling these procedures is message passing. Messages in Smalltalk are essentially what are known as function calls in modern languages.

The third idea relates to receiving a message. Upon receiving a message from another object, the receiving object (or receiver) attempts to execute the procedure described by that message. The message itself is called the selector.
The syntax for sending a basic message is as follows:

7 fibonacci

Here, a number object (7) receives a message named fibonacci. The number fibonacci or throws an error if the procedure cannot be found.

Messages can deliver parameters, as shown in the next example:

7 raisedTo: 2

In this case, 7 receives a message named “raisedTo:”. The colon is included in the message to let the system know that there is also a parameter available: the number 2. The receiving number can use the parameter in the procedure and in this case compute the value of 7 squared - 49.

Messages can also be split into multiple parts. Consider Peter William Lount’s example from the article “A Brief Introduction to Smalltalk” (2004): a natural English phrase “Jack, pass the ball to Jill.” could be written in Smalltalk as:

jack pass: theBall to: jill

Here jack is the receiver, the combination “pass:to:” the selector (split across the expression), and finally theBall and jill are the message parameters. The message passing paradigm, the ability to split messages into parts and the dynamic typing together allow very readable expressions.

The concept of objects applies to messages as well. In Smalltalk, messages are considered objects like any others, and so they can, for example, be passed as parameters for other messages.

Reflection

As mentioned earlier, objects in Smalltalk react to incoming messages by looking for the appropriate procedure to run. This is because Smalltalk is a reflective language. Reflective languages are capable of analysing and even modifying the structure of the application itself. Databases provide a good example for some of the advantages of reflection. Assume you have a table with some columns, and you have defined a class with member variables to match the column names. Using reflection you can simply iterate through the member variables and retrieve data from the table by using the member variables’ names themselves in indexing.
Control structures are objects as well
Smalltalk does not define fixed control structures, such as if. Instead, they are implemented as messages and used by sending the messages to objects. Even numeric operators (summation, multiplication etc.) are implemented as messages between number objects:

\[
x := 5 + 1
\]

Here the number object 5 receives a message “+” with an argument: another number object with the value of 1. The message then computes the sum and as a result, \(x\) will become a new object with value 6.

Classes
Smalltalk handles everything as objects. The objects can have varying classes, very much similar to modern object languages. Smalltalk also fully supports subclasses, and in the end all classes have the same ancestor: the Object class itself.

Use today
The “official” Smalltalk language was only developed until 1980, which is when Smalltalk-80 version 2 was released. However, since the 1980s there are multiple newer Smalltalk implementations and standards that remain in active development, for example ANSI Smalltalk (considered the current Smalltalk standard from 1998), GNU Smalltalk and Squeak.

References

[] A Brief Introduction to Smalltalk, Peter William Lount, 2004. Accessible at:
http://www.smalltalk.org/articles/article_20040000_11.html


[] Smalltalk.org - http://www.smalltalk.org/main/


[] Smalltalk - Progopedia http://progopedia.com/language/smalltalk/