# Comparation of Smalltalk and Google Language

## Chapter 1: Introducing Smalltalk

The programming language Smalltalk is an open source object oriented programming language that was designed by Alan Kay, Dan Ingalls and Adele Goldberg. Smalltalk was developed from 1970 to 1980 at the Xerox Palo Alto Research Center in California and released from 1980 to 1983. The development started with Smalltalk-71, went over Smalltalk-72, Smalltalk-74 and Smalltalk-76 to the nowadays konwn final and standard version Smalltalk-80.

In Smalltalk everything is an object. Even Strings, Integers, Booleans or a whole class is an own object. Every object can be inspected. An Integer object for example contains the self method that is comparable with the this constructor in java and an Integer object contains its value in binary, octal and hexadecimal too. In turn a String contains a list of characters. Further an Array in Smalltalk is designed to contain objects. For example this means that Integers, Strings and other types can be stored in one single Array.

Sending messages between objects is all about Smalltalk. A method can be called by sending a message. There are three types of messages you can do with Smalltalk. The first message is an unary message. An unary message contains only one word that will be sent to an object. For an example you can send the method today to the class date. The method today is the message the class date will receive. In the following illustration the message and the result of the message is shown.

```
Date today ---> 3 January 2020
```

In case of that if the class z does not contain the method, the message will be sent to the class above. If the top of all classes Object is reached and the method is still not existing, you will receive an error. But if the class Object contains the method, the method will be executed.

The second type of message is a binary message. Binary messages will be used to express logical, arithmetic and comparison opterations. One example for a binary messages is b + c and this will return the sum of b and a. A binary message can be up to 2 characters and consist of these following special characters:

```
+ / \ * ~ < > = @ % | & ? ! ,
```

The keyword message is the third type of message you can do with Smalltalk. It calls a procedure with at least two parameters. An example could be the argument "30 between: 0 and: 50". This is another example for a keyword message:

```
Array with: 1 with: 2 with: 3 with: 'last'. ---> #(1 2 3 'last')
```

As Christian Haider on the Night Of Knowledge in Lübeck 2017 said. Smalltalk is called Smalltalk, because the syntax should be that easy to read like an English sentence. This is the reason why arguments in Smalltalk end

with an dot, not with a semicolon. Additionally, Christian Haider said, Smalltalk contains only six words that are already defined at the beginning and do not need to be defined. These words are true, false, nil, super, thisContext and self. This means that everything else, even do or if loops need to be defined by the user.

## Chapter 2: Using grammer and parsers in Smalltalk

The easiest way to get a parser run is to implement the PetitParser. PetitParser is a framework developed for parsing or for writing parsers. It can be implemented in various Smalltalk platforms. In this study work every Smalltalk code is written in the Pharo IDE.

After the PetitParser is implemented, it is easy to define an own grammer in Smallltalk. PetitParser includes some terminal parsers lik #digit, #any or #word for example. Following is a grammer with its rule that it beginns with an digit and will be followed by at least one character.

```
firstGrammer := #digit asParser , #any asParser plus.
```

The first input returns false, because of corse there is no digit at 0 and the following is empty.

```
firstGrammer matches: '@'.          --> false
```

The second one is false too, because there is a character at 1, but still no digit at 0.

```
firstGrammer matches: '@x.          --> false
```

The third got a digit first of all, but got no further character ater the digit.

```
firstGrammer matches: '1'.          --> false
```

The last returns true because is has got an digit at first followd by at least one character.

```
firstGrammer matches: '9@test'.     --> true
```

For doing this simple code, its not even needed to create a class. This means that this can be done in the Pharo Playground. The Pharo Playground equals an input and output console. These grammer will now be advanced and more difficult.

The following grammer is expressing natural and expotential numbers in positiv and negativ.

```
expoNumber :=   $- asParser optional, #digit asParser plus, ($. asParser, #digit
asParser plus,
```

```
                ($e asParser / $E asParser, ($- asParser / $+ asParser) optional,
#digit asParser plus) optional) optional.
```

For better legibility there is the same code summed up in the following box.

```
naturalNumber := #digit asParser plus.
expo := ($e asParser / $E asParser), ($- asParser / $+ asParser) optional,
naturalNumber.

expoNumber := $- asParser optional, naturalNumber, ($. asParser, naturalNumber,
expo optional) optional.
```

The first two input don't match the grammer. The first one contains letter and the second one starts with two hyphen. The programm will return false.

```
expoNumber := 'asdf'            --> false
expoNumber := '--131'           --> false
```

The following inputs match the grammer, so it will return right.

```
expoNumber := '1909'            --> true
expoNumber := '-1909'           --> true
expoNumber := '-3.14E-17'       --> true
expoNumber := '3.14e+19'        --> true
```

## Chapter 3: Defining an own grammer and

In this chapter a own grammer will be defined, it will be parsed and tested. First of all its needed to create a new Subclass to PPCompositeParser. The subclass is called CoPGrammer and has the instanceVariableNames: subt for subtract, term, mul for multiplication, brackets, num for number, pro for production and var for variables.

In Smalltalk the class looks like in the following box.

```
PPCompositeParser subclass: #CoPGrammer
        instanceVariableNames: 'subt term mul var brackets num pro'
        classVariableNames: ''
        package: 'PetitParser-Tools'
```

```
brackets
^ $( asParser trim, term, $) asParser trim
```

```
mul
^ var, $* asParser trim, pro
```

```
num
^ #digit asParser plus flatten trim ==> [ :str | str asNumber ]
```

```
pro
^ mul / var
```

```
start
^ term end
```

```
subt
^ pro, $- asParser trim, term
```

```
term
^ subt / pro
```

```
var
^ num / brackets
```

```
testMul
self parse: '17*39' rule: #mul.
```

```
testNuum
self parse: '1909' rule: #num.
```

```
testSubt
self parse: '999-55' rule: #subt.
```

## Chapter 4: Comparing Smalltalk with Google Language

| What will be compared | Smalltalk | Golang |
|---|---|---|
| Release | 1980 | 2009 |
| Paradigm | Object-oriented | Multi-paradigm: concurrent, functional, imperative, object-oriented |
| Typing discipline | Strong, dynamic | Inferred, static, strong, structural |
| influenced by | Lisp, Simula, Euler, IMP, Planner, Logo, Skechpad, ARPAnet, Burroughs, B5000, cell (biology) | Alef, APL, BCPL, C, CSP, Limbo, Modula, Newsqueak, Oberon, occam, Pascal, Smalltalk |
| influenced | AppleScript, Common Lisp Object System, Dart, Dylan, Erlang, Etoys, Go, Groovy, Io, Ioke, Java, Lasso, Logtalk, Newspeak, NewtonScript, Object REXX, Objective-C, PHP 5, Python, Raku, Ruby, Scala, Scratch, Self | Crystal |
| Operating System | Cross-plattform | DragonFly BSD, FreeBSD, Linux, macOS, NetBSD, OpenBSD, Plan 9, Solaris, Windows |
| Stack Overflow Questions | 1.548 (05.01.2020) | 42.806 (05.01.2020) |

## Literature


- http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html#38
- http://esug.org/data/Old/ibm/tutorial/CHAP2.HTML#2.50
- https://2017.nook-luebeck.de/
- https://www.youtube.com/watch?v=Nsq8iRWE69Y
- https://www.lukas-renggli.ch/blog/petitparser-1?_s=eENdkyXszh_PzhrC&_k=WVHJ7bmf&_n&12
- https://www.lukas-renggli.ch/blog/petitparser-1#WritingaMoreComplicatedGrammar