COMPARISON OF JAVA AND SMALLTALK PROGRAMMING LANGUAGES

Tomáš Bublík
ČVUT-FJFI, tomas.bublik@gmail.com

ABSTRACT:
The two famous programming languages compared from different points of view in this paper. Nowadays, Java is one of the most popular languages, while Smalltalk is one of the oldest languages. However, these languages have much in common; there are also a lot of differences among them. Some important aspects are discussed in this paper. This paper should serve also as an educational tool for the beginning Smalltalk programmers.

KEYWORDS:
Smalltalk, Java, history, comparison, characteristics

Introduction

This paper is trying to compare two, apparently different languages. But the purpose of this paper is not to give a definite answer on the question: "Which one is better?" The paper has an educational and recommendatory character. On the base of this paper, a reader should make a decision, which language is the most suitable for him/her, or which language is better for his/her project. However, Java is currently much more popular language; this paper is more engaged in Smalltalk. Therefore, Java is not analyzed so deeply.
The aim of the paper is not to describe the tiny differences in syntax, or even to analyze syntax of the languages. This was already described more detailed in the other publications. Moreover, there exists a high quality and very detailed documentation of both languages. Although that, the basics of the languages are reminded in the opening chapters. Also the basic principles of its usages are discussed. The simple examples are given during the whole paper in both languages. These examples are chosen so, that it is not difficult to understand them.
After the history summarizing, follows the brief description of the languages. In the next chapters, the attention is paid to the individual differences, and the different approaches to solving the given problems. In the conclusion, the advantages and disadvantages of both languages are summarized from the different perspectives. The result of this paper should be a guide for a programmer standing in front of the decision, which of these two languages would be more appropriate for his purposes.

Brief history

Smalltalk

The first version of Smalltalk is known as Smalltalk-71. Alan Key and Dan Ignalls were by its foundations. While creating this language, they used its key property: sending the message. They were inspired by the Simula language. The language was further developed over the version Smalltalk-72 to Smalltalk-80. In this version are: the metaclasses, the inheritance, the classes library, and the development environment. Incidentally, the IDE already looked very much like the current implementation. With the metaclasses introduction, it managed to declare the "everything is an object" concept. In 1983, the version known as Smalltalk-80 version 2 was established. It was distributed as an image of the objects and a virtual machine. The ANSI Smalltalk was standardized in 1998. It is still valid. Currently, the most two popular implementations are the open source Squeak, implementing Smalltalk-80 version 1, and the VisualWorks, implementing Smalltalk-80 version 2. The Squeak has a large community of developers, including the developers from the base community. Thanks to it, the open source GNU Smalltalk was founded. The actual project, with the modern implementation of Smalltalk, is Pharo. Thanks to the Smalltalk project, windows, icons, a mouse, and a whole graphic environment of Macintosh computers was founded.

By the mid of the 90s, Smalltalk struggled with the high memory demands, and generally with the great performance requirements. That was one of the reasons of its small popularity. Another reason, why the language was not spread, was a very high price of the commercial development environments. However, after the years of the language development, it is remarkable, that it is possible to run the first version of the language's image even in the newest edition of a virtual machine. Even though, Smalltalk is far not spread as much as Java, it managed to affect a number of languages and technologies during its existence. For example, Smalltalk laid the foundations of Ruby and Objective-C, one of the most popular languages which are used by Mac OS X and iOS operating systems.

Java

Currently, Java belongs to the most popular languages. Compared to Smalltalk, Java is a younger language. The first mentions of Java are from 1991; at that time, James Gosling, Mike Sheridan, and Patrick Naughton announced the initiation of a new language project. The original purpose of Java should was in the interactive cable TVs, but it turned out that its usage can by much wider.

In 1995, the version 1.0 was founded at Sun Microsystems. After the release, there was implemented a support for the Java applets in browsers very quickly. This helped its wilder spread. The second version of Java was released in 1998. Along with this version, it was released the version for mobile phones known as J2ME (Java 2 Mobile Edition). In 2006, Java had three editions: Java SE, JavaME and Java EE. These are editions for the standard development, the mobile applications and the enterprise applications. Sun Microsystems releases Java as the open source. Therefore, Java is one of the most used languages. The Java distribution is divided into Runtime Environment (JRE) and Standard Development Kit (SDK). To run Java applications, just the JRE is needed. In addition, the SDK contains the compiler, and the other useful utilities for the development and the debugging. From the November 13th 2006, Java is completely free under a free license. In 2010, Sun Microsystems was bought by Oracle, however, Java continues in a similar philosophy. Current version of Java is 7.0. The numbers of downloads are hundreds of millions each year.

Basic characteristics

Smalltalk

The philosophy of Smalltalk is completely different from the other programming languages. Smalltalk is pure object-oriented language, even perhaps “the most objective one”. Unlike the other programming languages, Smalltalk uses very different principle of approach to solve various problems. In spite of the language is one of the older ones, it is fully applicable even today. Smalltalk has the elegance of the functional languages while preserving its imperative, object-oriented character. Smalltalk works with the general principle that everything is an object, even a class is an object. The types, primitive in similar languages, are considered also the objects in Smalltalk. An object in Smalltalk is defined by the state (the references to the others objects), and by the ability to send and receive the messages. Each object in memory is identified by its unique pointer. A garbage collector cares about the object destroying.

Smalltalk is simultaneously an interpreted and a compiled language. Before a virtual machine interprets the bytecode, the source code is compiled into it. Smalltalk is distributed similar to Java: as an image of objects and a small virtual machine. The image of the objects is completely independent of the used operating system. Only the virtual machine is modified for the desired platforms. So as Java, Smalltalk is a platform independent language. Smalltalk has the characteristics of the operating systems; therefore, it is not necessary to run virtual machine under an operating system.

Another nice feature is that the objects can be edited at runtime. Because even the interpret of the language is written in Smalltalk, it is possible to change the properties of the language itself. A program in Smalltalk can also edit himself at runtime. Smalltalk is often referred as a fully reflective language therefore.

Smalltalk is a dynamic language. There is no type checking, and its syntax is much more similar to the conventional human thinking. With the properly following the recommended rules and the conventions for the naming, the statements of Smalltalk are similar to the human written sentences. The syntax of Smalltalk is very easy to read and easy to understand. Smalltalk is built on the three basic operations: the message sending to the object, the object specification, and the object returning as a value.

Java

Java is one of the most popular languages. It is a platform independent language which is, like Smalltalk, interpreted and also translated. The translated code is called a bytecode and is processed by a virtual machine. The virtual machine is optimized for the various platforms. Java is an object-oriented, static, strong typed imperative language. The actual version is 7.0, and its default library is quite extensive. Like Smalltalk, Java uses a garbage collector for “cleaning” the objects which are not longer referenced. On contrary to the Java inspired languages (C languages), it does not contain the pointers. On the other hand, Java contains the primitive types.

The actual version of Java supports the generic types, the auto boxing, the collections, the threading, the annotations, and many other useful features.

Briefly about the syntax of Smalltalk

However, these two languages have a different syntax; a completely different behavior of Smalltalk is very interesting in case of the simplest operations. Smalltalk allows sending three types of messages: unary, binary and keyword. Simple adding two numbers is just a message sending operation; its interpretation is as follows:

```
3 + 5
```

It means that to the object 3 is sent a message named "+" with the argument of 5.

For checking if a number belongs to an interval, the message of keyword type can be used:

```
3 between: 5 and: 10
```

This can be interpreted like: The messages "between:" and "and:" are sent to the object 3 with the parameters of 5 and 10. The control structures work in Smaltalk similarly; again, it is a simple messaging between the objects. For example, the expression

```
1 = 1
```

is a sending the message "=" to the object 1 with the parameter 1. The result is a Boolean object which also reacts to the messages. One of the many messages could be "ifTrue". The program management, after the output of the two numbers comparison, could be obtained as follows:

```
1 > 2 ifTrue: [  ]
```

The square brackets are escaping the block. Inside the block could be the statements which are executed after the condition. In Java, a similar command would look like this:

```
if (1 > 2) {
}
```

A loop is also a message sent to the block of the statements:

```
[ ] whileTrue.
```

And the block of the statements will executing until its result will be "true". Because even a block of the statements is an object in Smalltalk, the message could be used also with the block like a parameter:

```
| var |
var := 10.
[ var > 0 ] whileTrue: [ var := var - 1 ].
```

The message “whileTrue” was sent to the object “[]”with the block parameter. This block will keep executing until the result of the original block is true. The for-loop works similarly; a combination of „to: “and „do: “ messages is used instead of „whileTrue“.

Differences

Exceptions handling

At first glance, both languages have a similar processing and catching the exceptions. For this feature, in Java were created some new keywords. On the other side, Smalltalk stood at the philosophy of the messages sending, and the syntax has not changed. Unlike Java, in Smalltalk are the exceptions catched not by the virtual machine. It has several benefits. For example, it s not necessary to restrict to the „tray-catch-finally“ blocks. Many useful options are even in the standard image. For example, it is possible to use the „retryUsing“ message, and repeat the block with the different setting. Furthermore, it is even possible to write a completely new customized behavior of the exceptions processing.
The syntactical differences of both languages are very striking. Java is a classic imperative programming language, and its syntax is very close to C-familly languages. The blocks of the statements are inside the curly brackets, the method calls are nested with dots, the arguments are inside the parentheses, and a semicolon server for the end of the statement. But still, it is a machine-readable language. The Smalltalk syntax is more human-readable, and closer to the natural human language, partly similar to Lisp. For the beginning programmer, it will be easier to learn Smalltalk due to its syntax intuitives.
Whereas Smalltalk have a full support of the closures, Java is still implementing them only in form of the anonymous inner classes. In Smalltak, it is even one of the key components. All the control structures and loops are defined by the objects accepting the closures. This feature should be implemented in some of the next Java versions. This is the example of a sum of the two numbers in Smalltalk:

```
sum := [ :a :b | a + b ].
```

After the statement

```
sum value: 5 value: 5.
```

is the result 10.

The different approach of the languages is in the scope of the attributes and the methods. In spite of Java, Smalltalk does not use the nested classes. Further, Smalltalk does not use the access rights to the methods. All the methods are public. Because the objects communicate through the messages sending only, there exist no direct access to the instance, or even the class attributes, in Smalltalk. Therefore, it is a much easier hierarchy of the access permissions in Smalltalk then in Java. For this purposes, Java uses the keywords „public“, „private“ and „protected“. The class attributes are marked by the word „static“ in Java. Smalltalk has more variables types. Into the core group of the classes variables, the instance variables, and the locals variables, it adds the global and the pools variables. While the global variables are shared by all the class instances, the pools variables are shared by the subset of the system classes.
In Smalltalk, it is a different class hierarchy and the class namespaces. In the commercial distributions of Smalltalk, there are the namespaces. On the other hand, there are not in

Squeak and Phar. In the class properties, it can be specified a category, however, it has not an influence for the program behavior. It has rather the informative character. The name of the class must begin with the capital letter, and must be unique in the whole environment and the namespace. Over Java, Smalltalk have another interesting feature: the methods can be versioned. It is possible to reveal the older version at any time.

Common properties

The two ostensibly different languages have a lot of common properties. First, they have a similar philosophy of the architecture; both languages are multiplatform, and both have separated the bytecode translation and the interpretation by a virtual machine. The certain machine is adapted for the certain environment.
Next, both languages are reflexive. But, unlike Java, Smalltalk is a more reflexive language. The reflexion is even one the fundamental pillars of the language core. In Java, it is possible to work with the classes, create the instances, or read the classes information, but such a degree of the reflexion is not high as in Smalltalk is. The Smalltalk reflexion allows to monitor and change the „living“ system. The methods can be added into the classes at runtime, or even the classes can be removed at runtime. The program in Smalltalk sees himself as data which can be read and changed. For example, in case of the non-existing method exception is catched, it is possible to add the requested method into to the target object, and start to use it, without the running program interruption. These dynamic changes are possible thanks to using the observer design pattern in the Smalltalk source code.
But there exist the methods for checking the object membership to the certain class, or the capability of a method understanding. With the message „respondsTo“, it can be determined, whether the object responds to the other message. It possible to find out, what class the object belongs:

```
anObject isKindOf: String
```

For example, a method easy adding to the class:

```
MyClass addMethod: #sampleMethod body: [ 'method' ].
```

Both languages are able to work with the multiple threads. But in Smalltalk, there are little more features for the working with the threads, and generally simpler approach to this aspect. Smalltalk allows assigning a larger number of priorities. Printing the text with Transcript is the simplest example of the parallel process usage:

```
[Transcript cr; show: 'Write something'] fork.
```

In Java, the same exercise would be little more complex:

```
handler.post(new Runnable() {
           public void run() {
               System.out.println("Write something");
           }
});
```

Smalltalk and Java are the languages with a simple inheritance. The interfaces known from Java cannot be used in Smalltalk; however, both languages can use the abstract classes. Both languages have a similar functionality for the methods overriding; the method from the child overrides the method in the parent.

Collections

Both languages have an extensive support for work with the dozens of the different types of the collections, and the standard libraries of both languages have a good supply of the pre-built structures. In Smalltalk, collections are divided according to whether they are indexed, sorted, whether they have keys, etc. The basic collections in Smalltalk are: the Set, the Bag, and the ChracterSet. They are non-indexed and non-keyed with the variable sizes. The Bag collection differs from the Set in that, it can contain the same elements more than once. The collections which contain the key and the value are in Java called Maps; while in Smalltalk, these are the Dictionaries.
It is interesting that Smalltalk uses one of the collections to store the references to the global objects. This structure is called the IdentityDictionary, and the name of the variable that refers it, is Smalltalk. The whole system, with all classes and other objects, sees itself like the one big data object. With this object, it works also the virtual machine.
Java and Smalltalk can iterate over the collections. In Java, it is used the well known "foreach", or it is possible to use the special class Iterator. Again, Smalltalk implements little more functionality. To the collections can be sent the messages "to:", "reversed:", "collect" , "detect", and many others in Smalltalk. For example, the sum determination is so easy:

```
#(1 2 3 3 3 4 5) average.
```

The fast and elegant choosing, or inserting the objects into a field, is the strong side of Smalltalk. The required operations, which take several lines in Java, can be solved very quickly in Smalltalk. For example, the count the number of certain characters in the sentence: "A very common sentence with Hello World!" can be easily obtained as follows:

```
Transcript show: ('A very common sentence is Hello World!'
select: [:each | each = $e]) size
```

By using this statement, the number of "e" letters is counted, and is printed on the output. This task would be slightly more difficult in Java:

```
String sentence = "A very common sentence is Hello World!";
       int counter = 0;
       for (int i = 0; i < sentence.length(); i++) {
           if (sentence.charAt(i) == 'e')
               counter++;
       }
System.out.println("The input parameter contained " + counter
+ " letters.");
```

It should be noted that this snippet of the source code cannot be written just somewhere in Java. If it would not be a part of some program, it would be necessary to create a class with the main method inside. Therefore, the number of lines in Java would be probably higher. On contrary, this is not in Smalltalk, because it has got the Workspace and the Transcript. This is a remnant of the console command line from the days, when it was thought that Smalltalk will be also the operating system.

Performance

The application performance is probably the most controversial part in Smalltalk. It is well known that programs in this language did not belong among the fastest. Also the memory requirements were very high. However, the performance differences compared to the other languages are not so obvious; it was certainly a handicap for Smalltalk in past. Nowadays, these differences are not so significant, and moreover, some implementations are very powerful (Strongtalk). On the contrary, Java is tuned up so much that it is one of the most power languages today. Its virtual machine is highly optimized and constantly evolving. The higher memory requirements relate only to its server solutions.

Smalltalk special properties

Every element is an object in Smalltalk, even a class. The classes, whose instances are classes again, are called the metaclasses. Whenever the new class is created, also the metaclass is created automatically. All the metaclasses are the instances of the class called Metaclass. The metaclasses typically allows the creation and initialization of instances, and the initialization of the class variables. The relationship "class – metaclass" is one-by-one. There exists exactly one metaclass to each class and vice versa. The main idea of this relationship is that the instances do not have the access to the metaclass instance variables, and metaclasses do not have access to the instance variables of the instances.
As mentioned above, Smalltalk is a dynamically typed language, which means, that the types are not considered while writing a code; however, the type is controlled within the language. The object decides how to behave for its neighborhoods. In Smalltalk, the type membership is controlled at the time of deciding how to react on a message. Unlike Java, where the type is already given during the declaration. The references to the objects are transferred only by its reference, not by the value. How will an object react on the message, depends on whether the message is understood or not. In case that not, the MessageNotUndestood exception is raised. In addition, Smalltalk has the additional property: an object does not have to respond to the message, and can pretend that he understands to the all messages.

Blocks

The blocks have the special properties in Smalltalk. The block of code is enclosed in square brackets. The statements are separated by a dot. In spite of Java, the usability of the blocks in Smalltalk is much larger. A block is also considered as an object, and therefore, the messages can be sent to it. The block can be the content of a variable, the message parameter, or the element of a collection. At least, the message "value" can be send to evaluate the block and return its result value. This is another interesting feature of Smalltalk: A block in Smalltalk is not executed in the moment when the program is passing through it, but in the moment when a message is sent to it. For example, this code:

```
index := 0.
incrementBlock := [index := index + 1].
10 timesRepeat:
          [Transcript
               show: incrementBlock value;
               space]
```

will print the sequence of the consecutive numbers from 1 to 10 on the output. Even in the case that the index is incremented out of the cycle. This is because the block is performed each time the message is sent to it. This property of blocks can be widely used. In combination with closures, it is possible to write the operations, which would take several lines in Java, on two lines:

```
 (1 to: 6) inject: Set new
      into: [:ae :each | each even ifTrue: [ae add: each].ae]
```

The pair of the messages "inject" and "into" is used in this example. The even numbers are chosen from the "Set" of numbers from 1 to 6, and are inserted into to the new set. The message "inject" creates the default value, while the message "into" iterates over the result and saves it into the local variable. The output can be obtained by using "Print it" statement. Blocks can be used in multithreading. Each block can be run in a separated process by sending the "fork" or "forkAt:" message. An argument of the "forkAt:" message is the process priority level. The fact, that the block represents an object, can be uses in the communication between processes:

```
sharedQueue := SharedQueue new.
readProcess :=
          [[Transcript show: ' R' , sharedQueue next
printString] repeat] fork.
[1 to: 5
     do:
          [:index |
          Transcript show: ' W' , index printString.
          sharedQueue nextPut: index.
          Processor yield]]
          fork.
readProcess terminate
```

The two processes are executed in this program: "readProcess" like an infinite loop, and the second process like a 5-times repeated loop. Both processes are printing the letters "W" and "R" followed by the index on the output. Both processes are running concurrently, and because the second process contains a processor release command, the processes are alternating. Because the second process is in an infinite loop, it is necessary to end it with the "terminate" message. The output is the sequence:

```
W1 R1 W2 R2 W3 R3 W4 R4 W5 R5
```

Development environments

Despite of Java has not some interesting features that are standard for Smalltalk from the beginning; Java is a winner in the development environments area. This is especially due to the much larger user base and a disproportionate higher popularity of the language. The development environment for Java is more adapted to the different operating systems. Also the area of the new technologies support, or the number of plug-ins, is better for Java. Smalltalk offers a more conservative environment which is not different from those in the 80s. In addition to the standard tools, Smalltalk provides the several features that are specific to this language. The standard tools that persist are: Debugger, Workspace, Inspector, Transcript, and System Browser.
The debugger is very advanced. It is possible to run multiple debuggers simultaneously, and for example, also multi-threaded applications can be debugged. Also, a part of the source code can be debugged; just a part of the desired code can be marked and debugged. While

debugging, the calculation can be restarted, or even the code can be changed. Unlike the Java debugging tools, in Smalltalk debuggers, it is possible to step back to the previous operations. From this point of view, Smalltalk practically does not know the difference between "debugging", "code creating", and "running" modes.
The Workspace is a basic screen for the source code inserting. It is an advanced text editor. The Transcript plays the role of default (standard) output. If it is not specified otherwise, the output of running program is written as follows:

```
Transcript show: ‘ ’.
```

The System Browser is a tool for the system library accessing. Because if offers the editing, and the new elements adding, it is probably the most comprehensive tool of the entire distribution. It is composed of 5 windows, and very clearly presents the library content to the user. It is possible to browse and change classes with the attributes, the instances, the method parameters, and much more. Moreover, everything is sorted into the categories with the good comments. This tool also serves as a quick help or documentation.
The Inspector is the last describing Smalltalk tool. It serves to a detailed stepping of the current running program. The instance variables, the commands for the virtual machine, and the actual methods are clearly displayed here. This is a very useful tool which gives the opportunity to insight the actual performing operations.
There exist several other tools in Smalltalk, but their describing would be beyond this paper. An unpleasant fact is that these tools are very expensive for a commercial usage. In this respect, the development environments for Java are the winners. They are often for free: JDeveloper, Intellij Community Edition, NetBeans, Eclipse. On the other side, while comparing the computer utilization, the development environments for Java are considerably worse. Earlier, it was vice versa; the Smalltalk environments were to demanding for many computers. Nowadays, it is rather opposite. The Java development environments can take a lot of memory of a common computer, while Smalltalk does not need so much.

Languages architecture and virtual machine

Both languages have a very similar architecture of a source code processing and execution. A program in Java is translated into the bytecode which represents a sequence of commands for the virtual machine. The classes in Java are loaded and forwarded by the system of classloaders. Next, the virtual machine is divided into the heap, the stack, the method area, and the set of the registers. The information about the currently executed instruction is stored in the registers. In the stack are: the frames of current method, the references into the code pool, etc. The instances of the classes are in the heap. The method area is used to store the informations about the classes, and can be used also as a part of the heap.
The Smalltalk virtual machine consists of a stack and several special registers. The stack is organized into the stack frames. The most new frame points to the FP register, and each frame has a reference on the previous one. The next elements are the contexts. The contexts contain the local variables of the nested block, and the stack frames can keep the references on them. These contexts can be nested together. The Smalltalk virtual machine is also written in Smalltalk. Therefore, the important elements are the objects again. They holds not only the copy of the stack frame, but they have also the references on the compiled methods which contain the executable code of the block (usual it is executed when the block receives a message). The compiled methods are also an object which holds the executable code of the block or the method. A detailed description of the Smalltalk bytecode is beyond the scope of

this paper.

Program distribution

The program written in Smalltalk can be copied and executed on the other computer, only by transferring the source code. But this is only in case that the system objects were not modified. It is necessary to transfer also the whole image then. Due to the various distributions of Smalltalk, it is more confident to supply the source codes and the whole corresponding image together. In the VisualWorks implementation, a project can be exported as the file with the *.im suffix. But always, it is necessary to submit also the virtual machine corresponding to the appropriate platform. The whole image has approximately 30-50 MB. The virtual machine has about 1 MB. Therefore, today, it is not a problem to deliver the both. Java requires JRE (Java Runtime Environment) to run the programs. The JRE is a little larger than the Smalltalk image (about 90 MB). However, because Java is one of the most popular languages, it is more likely that the chosen computer will have a JRE. But today, the difference of the necessary data sizes is minor, and both languages are equal in this regard. If a program written in Java needs some other libraries, it is necessary to attach them into the JAR package. The Java programs are distributed in the JAR packages most often. Java has the advantage that the program will work with the installed JRE, because it is just one, from the Oracle. On the contrary, there are a lot of different Smalltalk implementations, and the probability, that a random code copied from the internet will work, is much lower.

Conclusion

Finally, the verdict of this paper should not be “which one is better”. This is practically impossible task, because each language is designed for a different usage. Further, each language has a completely different approach to the individual problem domains. The purpose of this paper is to provide a guide for the decision, or make a choice easier.
The Java language has developed from the evolution from the C languages, and still, it is adapting to the current needs and technologies. For the current time problem, it will be much easier to find a higher amount of the information and experiences from a huge number of developers that using it. However, Smalltalk goes the different way. It was a revolutionary language at the time of creation, and runs ahead his time. It provides the options and the tools that are not in Java. It is a visionary language which does not consider the technical limitations. The program written in Smalltalk 30 years ago can be edited, and improved even today. Likewise, it will be in the future. The program written in one of the first versions of Java will work on the newest virtual machine; however, nobody will use it. It will be too tight to the technology of the time of its creation, and the rewriting will be very difficult and expensive. There are many situations, similar to the situation, when somewhere runs a very old and slow information system, developed with several different skilled programmers, programmed, for example, in EJB 1.0, with manual database connections to the old database. Often shows up, that the most feasible way is to rewrite the whole system into a new technology. Furthermore, this is not a permanent solutions, after few years later, the similar situation occurs again. The programs in Smalltalk do not face to this faith; the syntax of Smalltalk stays practically the same (it is still the messages sending between objects). Who will learn Smalltalk, will be also able to understand the 30 years old programs. It does not have to be true in Java, because the syntax is changing. There are also a lot of frameworks and tools, and a programmer needs to know them. Today, a few programs are written in pure Java, without using only the standard libraries.
When considering the economic aspects of the decision, it will be depend on the language application. For the large systems, with many news technologies and good support, it will be better to choose Java. Java has the bigger community base. On the other hand, for the faster

development of smaller applications, Smalltalk will be better. It is obvious, that just from the several examples in this paper, in Smalltalk, the same operations are written effectively. It needs fewer lines than Java for the same operations. The programs can be written faster, can be changed during the execution, and can be more quickly adapted to the changing requirements.
Also, it should be considered that the syntax of Java is still changing, and the new elements are still added, while the syntax of Smalltalk remains the same. Only the library of Smalltalk is changed. Where the Java adds the functionality by changing the syntax, Smalltalk adds the new objects, but the syntax remains unchanged.

Acknowledgement

This work is supported from the SGS 11/166/OHK4/3T/14 grant.
I would like to show my appreciation towards assoc. prof. Vojtěch Merunka, Ph.D.for his valuable advices and comments.

LITERATURE

1. Alan Kay, Dan Ingalls, Adele Goldberg, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley Professional; 1 edition (January 11, 1989). ISBN-10: 0201136880, ISBN-13: 978-0201136883

2. Allen Wirfs-Brock, Pat Caudill, Instantiations, Inc., *A Smalltalk Virtual Machine Architectural Model*, Instantiations, Inc., 1999

3. Alex Sharp, *Smalltalk by Example: the Developer's Guide*, McGraw Hill Text; ISBN: 0079130364, 1997