ELECTRONICS' 2004 22-24 September, Sozopol, BULGARIA

ANALYZE OF PROGRAMMING TECHNIQUES FOR IMPROVEMENT OF JAVA CODE PERFORMANCE

Ognian Nakov, Dimiter Tenev

Faculty of Computer Systems And Technologies, Technical University-Sofia, "Kliment Ohridski" 8, Postal Code 1000 Sofia, Bulgaria, phone: +359 02 9653731, e-mail: ditenev@lark.tu-sofia.bg

Keywords: java performance programming techniques

The paper describes some programming techniques for improving of Java code performance. Few programming code samples are considered, as most common to appear in a program's code. Proposal for replacement of these code samples with ones using less computer resources (CPU – central processing unit and memory) is made. Diagrams of used computer resources are shown, so benefits from proposed code samples are easy to observe. By using comparing type "a code sample versus another code sample" is easy to find bottlenecks and overhead points in a program code, and thus optimize it. An optimized program code leads to faster execution and/or less memory usage, so a programming task will require less powerful hardware. A well-written program code can significantly improve performance of the Java application.

1. INTRODUCTION

Since Sun Microsystems created Java language it becomes more and more popular. It can be used to represent server and client side of an application. A powerful feature of the Java language is its portability. An application written in Java language can be run on any operating system (which has Java Virtual Machine) without need to recompile program's code. The Java language portability is result of using Java Virtual Machine, which is middle tier between the Java program code, and the operating system. This leads to Java applications' performance fall, and thus an important disadvantage appears.

Performance of Java applications is not so fast as applications wrote on any native (for operational system) language, which is problem of the Java language's nature. As far as there is a virtual machine (Java virtual machine) between the compiled code (byte-code) and the operating system the native language will always be faster. One disposes of two approaches in relation to the cited problem - hardware and software. The first one is to compensate the slowness by using more faster, but more expensive hardware. The second one is to improve the performance of the Java code through programming techniques and specific features of the Java language.

By using good designs, following good coding practices, and avoiding bottlenecks, Java applications usually run fast enough. However, the first (and even several subsequent) versions of a program written in any language are often slower than expected, and the reasons for this lack of performance are not always clear to the developer [1]. Usually there are few ways (depending on the Java language) to complete a desired programming task. By using the appropriate programming technique, it is possible to make the program's code much efficient.

Through examining different programming techniques, and performance comparing of each other, a software developer can easy choose the appropriate ones, and thus remove many bottlenecks and overheads in a Java program. The result is faster Java code execution, less memory usage, and thus use of less powerful hardware to complete a programming task.

The aim of the article is not only to show better Java programming techniques, but also to provide clear information how exactly a code sample is better (faster, less memory consumed) than the usual used Java code.

2. ANALYSING PROGRAMING TECHNIQUES

In a program code there are many code statements, which can be analyzed. Depending on the specific program's application one statement, or another can influence on the program execution. For example in a word processing program the statements, which work with letters and String objects will influence the program's execution, because they will appear most often in the program's code.

The article describes eight programming techniques, which can optimize the Java program execution. Observed Java code samples are the most common used ones. They can be used to optimize any Java program code, because they are used in almost every program's line, and do not depend on the specific problem which a Java application solves.

The observed programming techniques are analyzed through including code statements for measuring consumed CPU time and memory directly into analyzed code. In this way any negative influence, which a profiling program can include into measurements (additional computer resources which a profiling program consumes through the measuring process) is removed. The code used to analyze a code sample is as follow:

```
long startMem = Runtime.getRuntime().freeMemory();
long startTime = System.currentTimeMillis();
// here follows analyzed code sample
long endTime = System.currentTimeMillis();
long endMem = Runtime.getRuntime().freeMemory();
```

The implemented Java methods "*Runtime.getRuntime().freeMemory();*" and "*System.currentTimeMillis();*" executes very fast (takes about 1ms) compared to the analyzed code samples (executes between 800ms and 2000ms). Thus the included measuring error is less than 0.25%.

The analyzed programming techniques follow.

2.1 Use of simple types

The simple types are ones of the most used in an application. Their use can not be omitted. The observed simple types are the integer numbers (int, byte, long). The default integer values in the Java are of 'int' type; so all numbers in calculations will be caste to 'int' type (if number fits this value). The analyzed code sample shows that for fast program execution it is better to use 'int' type, even your value fits into 'byte' or 'short' types (Fig. 1).

Tested code samples:		
for(int i=0; i<size; i++);	*for(byte i=0; i<size; i++);*	*for(long i=0; i<size; i++);*

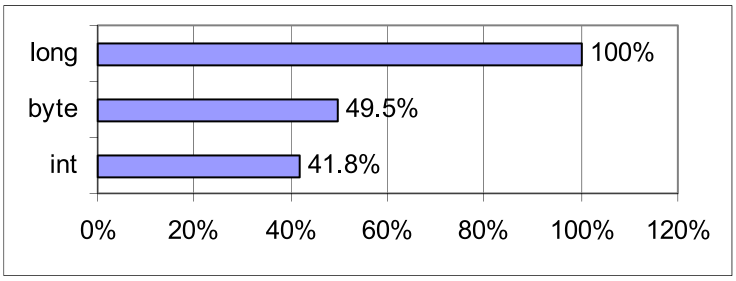


Fig. 1 - Time usage for accessing a simple type

2.2 Use of simple type wrapper classes

The Java language offers wrapper classes, which corresponds to the simple types. Wrapper classes are usually used for converting a simple type value to a string, or for formatting purposes. However the use of a wrapper class instead of its corresponding simple type leads to performance fall. The figures 2 and 3 show the memory and the time (for accessing the value) used by a simple type variable and a wrapper class instance.

Tested code samples:	

int intArray[] = new int[size];
intArray = i;
	Integer array[] = new Integer[size];
	array =new Integer(i);

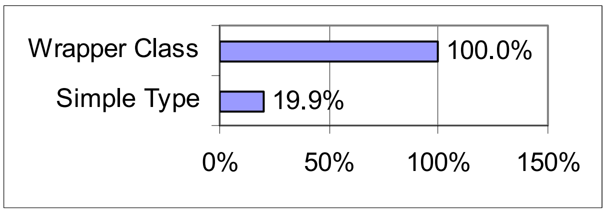


Fig. 2 – Memory Usage

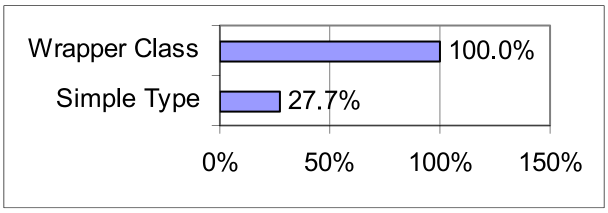


Fig. 3 – Time Usage

2.3 Use of String objects and string literals

Use of text in the Java language is capsulated in a class 'String'. There are two ways to create a string object in Java. The first one is to use the 'new' operator, and the second one is to omit 'new' operator and let compiler handle the string object creation.

By omitting 'new' operator the compiler can create an internal pool of strings. Because the String object is immutable (its value can not be changed), many references (instead of objects) can point to one and the same string into memory. The figure 4 shows the time used to create a string using 'new' operator and using literals.

Tested code samples:	
String str = new String("Hello world!");	*String str = "Hello world!";*

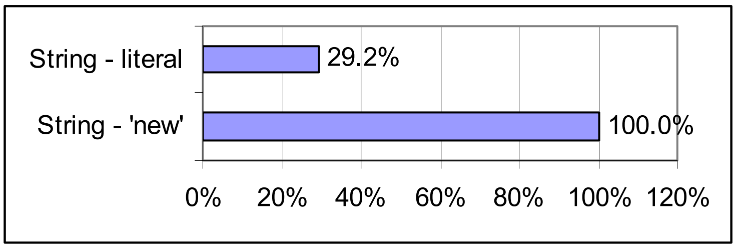


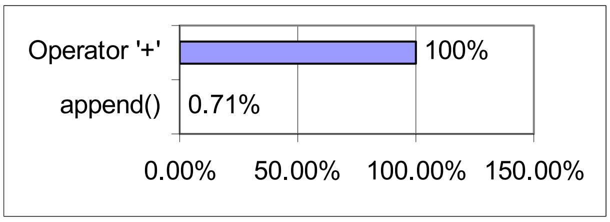
Fig. 4 – Time usage for creation of a string object through 'new' operator and literal

2.4 Text concatenation

The Java language provides '+' operator for string concatenation. This operator is easy to use, but leads to great memory leak, and CPU usage. As String is immutable object, a concatenation will lead to creation of a new String object, and garbage collecting the old ones. This disadvantage can be overcome by using the StringBuffer class and its 'append()' method. The figures 5 and 6 show the memory and time consumed to concatenate a string using "+" operator and 'append()' method.

Tested code samples:	

StringBuffer sb=new StringBuffer("Hello ");
sb.append("world!");
	String str = "Hello ";
	str = str + "world!";



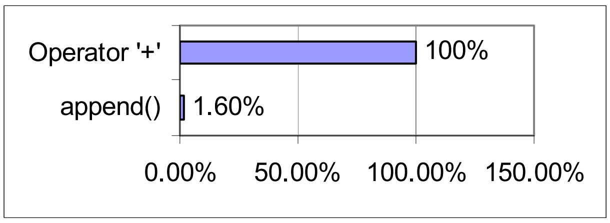


Fig. 5 – Memory Usage Fig. 6 – Time Usage

2.5 Use of arrays versus ArrayList objects

The Java language includes classes, which can store and remove objects dynamically, while array objects can not change their size. The analyzed technique includes an instance of class ArrayList and a instance of a simple array. The results show that access to elements of array object is much faster than accessing the elements of ArrayList.

Tested code samples:	
Integer val = (Integer) array[i];	*Integer val = (Integer) arraylist.get(i);*

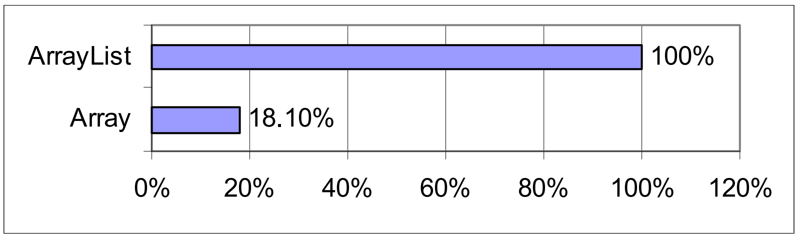


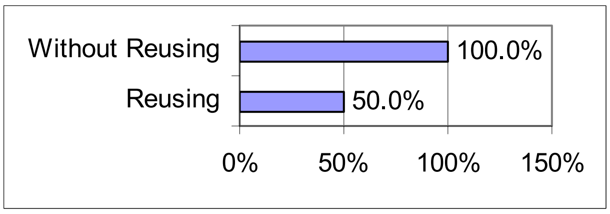
Fig. 7 - Time used to access an array/ArrayList element

2.6 Collections reusing

As strongly object-oriented language, Java encapsulates all functionality into classes. It is essential to reuse (if it is possible) any objects that we create and use in the Java programs. The object reusing can increase the program speed. The results of reusing a common Java object - ArrayList collection is shown on figures 8 and 9.

Tested code samples:	

ArrayList arrList = new ArrayList();
arrList.add(new Integer(0));
	arrList.clear(); // reuse
	arrList.add(new Integer(0));



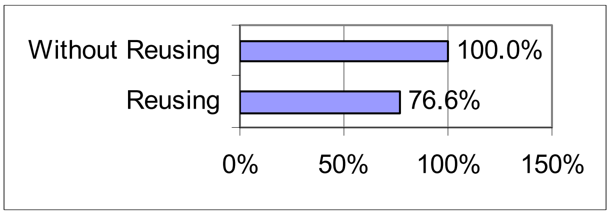


Fig. 8 – Memory Usage

Fig. 9 – Time Usage

2.7 Type casting

The use of classes and inheritance lead to type casting. Type casting takes CPU time, and thus it is important to omit it where it is possible (in loops for example). The analyze shows that casting a parent object to its underlying child is the most time consuming (considering casting a child object to its parent, and no casting) fig. 10.

Tested code samples:	

Parent parent=new Child();
((Child)parent).aChildMethod();
	Child child=new Child();
	((Parent)child).aParentMethod();

Child child=new Child(); *child.aChildMethod(); // no casting*	

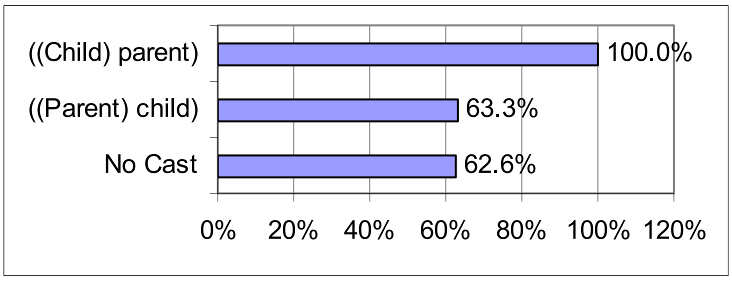


Fig. 10 – Time used for a type cast

2.8 Classes inheritance

Implementing functionality into classes leads to inheritance. The Java language presents anonymous classes. They are often used in events processing. Use of anonymous classes is comfortable (the event processing is posted where the object is created), but it consumes much more time than using a named class. The usage of anonymous classes is not appropriate decision for processing often generated events (mouse move for example) fig. 11.

Tested code samples:	

*Ainterface impl=new Ainterface() {
    public void aInterfaceMethod() {
    }
};
impl.aInterfaceMethod();*
	*AinterfaceImplementation impl=new AinterfaceImplementation();
	impl.aInterfaceMethod();*

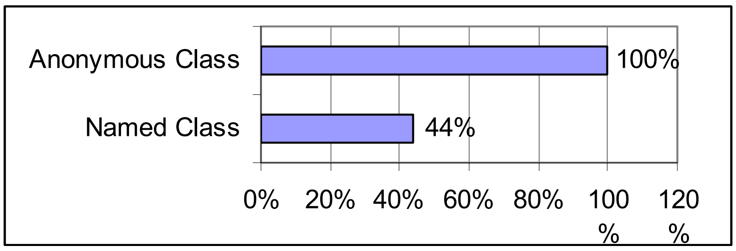


Fig. 11 – Time used to call method of named and anonymous classes

3. CONCLUSION

The performance of an application can be improved in two ways: by hardware acceleration, or by revising application's code and removing code's bottlenecks and overheads. The article proposes few programming techniques, which can increase the execution speed of the Java programs.

The analyzed programming techniques can significantly increase performance of the Java applications. As the most common used ones they can be implemented in any Java application, as simple code - replace. The software developer can observe and decide which technique to use. The estimations of CPU time and memory usage are visually presented, so it is easy to distinguish which technique to use. Usage of fast code execution leads to use of less powerful hardware.

4. REFERENCES

[1] Shirasi J., *Java Performance Tuning*, O'Reilly & Associates Inc., 2000.
[2] Eckel B., *Thinking in Java, Second Edition*, Prentice-Hall, June 2000.
[3] Horstmann C., Cornell G. *Core Java Volume I Fundamentals*, Prentice-Hall, 1999
[3] Chan P., Lee R. *The Java Class Libraries: An Annotated Reference*, Addison Wesley, September 1996