# Designing And Implementation Of A Tool For Java Code Optimization

Rahma Saleem Alsawaf
Software engineering Department
College of computer sciences and mathematical
Mosul, Iraq

Dr.AsmaaYaseen Hamo
Software engineering Department
College of computer sciences and mathematical
Mosul, Iraq

***Abstract*—this research contains a full description of designing and implementation of a Java Code Optimization Tool (JCOT).This tool is introduced to the programs who's the tools user it enables him to scan his code and show the segments that may be optimized according to reducing execution time. The tool also given the user a " severity level " that helps user to decide applying the optimization or not.**

***Keywords— JCOT***

## I. Introduction

When writing Java code it can be easy to make simple mistakes that seem harmless on the surface but, as the application grows larger, it can show themselves to be slow, resource intensive processes that could use a tune-up. So it is important to optimize the code [1]. Optimization is the process of transforming a piece of code to make it more efficient (either in terms of time or space) without changing its output or side-effects. The only difference visible to the code's user should be that it runs faster and/or consumes less memory [2]. Code optimization can be divided into three distinct types, which are based on the needs of the developer: Maintainability, Size and Speed. Maintainability optimization is performed to help make code more manageable in the future. This type of optimization is usually geared toward the structure and organization of code rather than modifications to the algorithms used in the code. Size optimization, which involves making changes to code that result in a smaller executable class file. The cornerstone of size optimization is code reuse, which comes in the form of inheritance for Java classes. Speed optimization is without a doubt the most important type of optimization when it comes to Java programming. Speed optimization includes all the techniques and tricks used to speed up the execution of code. Considering the performance problems inherent in Java, speed optimization takes on an even more important role in Java than it does in other languages such as C and C++ [3].

## II. Literature Review

In (2008) Kevin Williams, etc., writes a paper that presented analysis of existing interpreter optimization techniques on the Cell BE Processor and introduced novel optimizations made possible by the architectural features of the Cell BE SPE [8].

Also in (2008) Huib van den Brink discussed the optimization techniques used in the Java HotSpot Compiler, in order to execute the program code as fast and efficient as possible [9].

In (2010) Peter Sestoft made experiments show that there is no obvious relation between the execution speeds of different software platforms, even for the very simple programs studied here: the C, C# and Java platforms are variously fastest and slowest [10].

In (2012) Pawan Nagar and Nitasha Soni presents a basic framework that allows the application programmers to recognize the constraints of application programs in instruction scheduling [11].

Also in (2012) Hiroshi Inoue and Toshio Nakatani presented a techniques to identify the instructions and objects that frequently cause cache misses without using the HPM of the processor and then showed its effectiveness in compiler optimization using two examples. The key insight is that the cache misses are often caused by pointer dereferences in hot loops in the Java programs [12].

## III. Requirements for the Java Code Optimization Tool (JCOT)

Before describing the algorithm for JCOT, the functional and non-functional requirement of it is specified as functional shown below.

### A. *Functional requirement for JCOT*

Functional requirement is a statements of services that the system should provide. How the system should react to particular inputs and how the system should behave in particular situations [5].

- It must contain one or more techniques for the optimization to make the tool work in correct way.
- It needs from the user to specify the name of the saved file to work on it and it should be free of mistakes or compilation errors.
- The tool must be able to recognize the state that fines in the file to work with it.
- The tool must work continues without error.
- The tool must show the severity for each state of the optimization to enable the user to know the speed increased if it is used.
- The tool must save the result file after optimization to enable the programmer to use it later.
- The result file should work correctly.

- The tool interface must enable the user to select from apply optimization process or not and save the file before process optimization if the user select no.

### B. *Non-Functional requirement for JCOT*

Nonfunctional requirement is Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. [5].

- When designing tool interface must follow the designing document that need the tool interface must be easy to work by the user.
- The tool must be able to work without installation of java language kit for easy used by the user.
- The tool must have graphic interface to work with it by the user.
- The tool should be compatible with Windows in browsing folder and select the specific file.
- Should display to the user the location for the code segments that contain the optimization techniques and process it.
- Should display the code after apply optimization.
- Must display the severity curve for the optimization techniques to enable the user to know the details about it.

## IV. DESIGNING THE JAVA CODE OPTIMIZATION TOOL (JCOT)

The work starts by selecting the code to work with it and segments it according to optimization techniques that embedded for designing the tool and displays the code segment to the user with ask you for accept doing the optimization process on that segment if the user accept, the tool apply the optimization process on that segment and display it with the severity curve to enable the user to know detail about that.

As mention in [4], the optimization techniques for java code is specified. The influence of every techniques on the time is calculated as shown in the table below and also the following four techniques:

TABLE I. COMPARISON OF EXECUTION TIME OF OPTIMIZATION TECHNIQUES

| Techniques name | Time before optimize the code (in millisecond) | Time after optimize the code(in millisecond) |
|---|---|---|
| Use String length to compare empty string variables | 874 | 89 |
| Avoid invoking time consuming methods in loop | 1818 | 93 |
| Avoid empty if | 67 | 59 |
| Avoid unnecessary if | 60 | 59 |
| Avoid unnecessary parentheses | 67 | 59 |
| Avoid using Message Format | 735 | 93 |
| Avoid new with string | 48 | 2 |
| Avoid null check before instance of | 4 | 3 |
| Do not create instances just to call get Class on it | 9 | 2 |
| Reduce switch density | 5 | 4 |
| Avoid new Integer to String | 3752 | 3146 |
| Avoid passing primitive integer to Integer constructor | 992 | 90 |
| Avoid passing primitive long to Long constructor | 865 | 727 |
| Avoid passing primitive char to Character constructor | 818 | 135 |
| Avoid unnecessary substring | 50 | 9 |
| Avoid equality with Boolean | 126 | 100 |
| Avoid instantiation of Boolean | 86 | 11 |
| Use single quotes when concatenating character to String | 1124 | 960 |
| Avoid multi-dimensional arrays | 22 | 21 |
| Use String instead String Buffer for constant strings | 422 | 199 |
| Avoid creating double from string | 263 | 135 |
| Always use right shift operator for division by powers of 2 | 7 | 2 |
| Use shift operators | 2 | 1 |
| Avoid using exponentiation | 283 | 5 |
| Moving Secondary Boolean Operation Outside The For Loop | 893 | 892 |
| Optimize Declarations | 2 | 1 |
| In for loops, countdown rather than up | 98 | 97 |
| Use operator=, rather than just the operator | 85 | 80 |

**Techniques 1: Loop invariant code motion**

When using absolute function inside the loop statement and don't depending on it and don't use inside the loop in another statements leading to increase the execution time , so must movie it outside the loop statement if not affect the program output [6].

Example:

Without optimization

```
class test
{
    public void method(int x, int y, int[] z)
    {
        for(int i = 0; i < z.length; i++)
        {
            z[i] = x * Math.abs(y);
        }
    }
}
```

With optimization:

```
class test
{
    public void method(int x, int y, int[] z)
    {
        int t1 = x * Math.abs(y);
```

```
            for(int i = 0; i < z.length; i++)
            {
               z[i] = t1;
            }
      }
}
```

**Techniques 2: Avoid consecutively invoking String Buffer append with string literals**

When concatenating string to another one must let to don't replay the concatenating statement and needlessness the only one of it to seep the execution time.

Usage Example:

Without optimization:

```
public class Test
{
      private void fubar()
      {
            StringBuffer buf = new StringBuffer();
            buf.append("Hello").append(" ")
               .append("World");
      }
}
```

With optimization:

```
public class Test
{
      private void fubar()
      {
            StringBuffer buf = new StringBuffer();
            buf.append("Hello World");
      }
}
```

**Techniques 3: Avoid duplication of code**

When using condition statement must let to not replay the instruction used inside it.

Usage Example:

Without optimization:

```
public class test
{
      public void method()
      {
            int x = getValue();
            if(x > 10)
            {
                  int j = i + 10;
                  int k = j * 2;
                  System.out.println(k);
            }
            else if( x < 20 )
            {
                  int j = i + 10;
                  int k = j * 2;
                  System.out.println(k);
            }
      }
}
```

With optimization:

```
public class test
{
      public void method()
      {
            int x = getValue();
            if(x > 10) || ( x < 20 )
              {
                  int j = i + 10;
            int k = j * 2;
            System.out.println(k);
              }
      }
}
```

Casting process is the operation that modify the contain of any parameters from the type define for it to any type as needed. Such that the parameter x have the return type integer and need that in type of flout to continue the other operation can use casting. So must avoid replay use it if it is unnecessary because lead to need extra time for execution the program [7].

Usage Example:

Without optimization:

```
class test
{
      public Object method()
      {
            String str = "AppPerfect";
            Object obj = (Object)str;
            return obj;
      }
```

```
}
With optimization:
class test
{
        public Object method()
        {
                String str = "AppPerfect";
        Object obj = str;
        return obj;
        }
}
```

TABLE II. COMPARISON OF EXECUTION TIME OF FOUR NEW OPTIMIZATION TECHNIQUES

| Techniques name | Time before optimize the code (in millisecond) | Time after optimize the code(in millisecond) |
|---|---|---|
| Loop invariant code motion | 2481 | 1280 |
| Avoid consecutively invoking String Buffer append with string literals | 10788 | 5522 |
| Avoid duplication of code | 367 | 61 |
| Avoid unnecessary casting | 4 | 3 |

**Algorithm for the Java Code Optimization Tool (JCOT)**

JCOT works as in the following algorithm

Step 1: Choose one project (file) written in java language to read the code inside it and use to applying optimization on it.

Step2: Repeat.

Step 2.1: Read a code segment and match it according to optimization techniques.

Step 2.2: On the screen give the both codes (before the optimization and after optimization).

Step 2.4: Wait for user to select applying optimization or not.

Step 2.5: If user selects yes change the segment and store it in the new file.

Step 2.3: show the severity curve for the code in the severity with appropriate long and color.

Step 3: until end of file

## V. RESULT

Now having the JCOT and ready to use after following the working steps and gathering the benefits from it.

Steps for working with JCOT:

- The programmer open the tool icon (without needing to install the java environment) and see the opening window for the JCOT as shown in figure (1-1).
- Clicking the enter bottom in the primary window will move the user to operational window as show in the figure (1-2).
- User should select a file to be optimized by pressing the browse bottom, the massage window appear and know the user to select the file as the same figure (1-3), after press the ok bottom in the massage window the open window appear to the user to select the file in where you store in your computer as show in the figure (1-4).
- The programmer must select the read bottom in the same window to read and segment the code then press the view bottom to view the code segment then the user must select from applying optimization via optimization bottom or not in no bottom if the user select optimization the JCOT take the code segment before optimization.
- The JCOT enable the programmer to know the severity of the techniques by show the colors indexing to that such that red color show the techniques accelerate the execution a very large margin and so on changes the color that up to yellow color that means less affect in speed of execution . As show in the figure (1-6).
- The programmer can choose out of operational window by press the back bottom and return to the primary window , in first window the programmer choose the exit to close the tool and return to desktop of the computer.

Fig. 1. Primary window for the JCOT.

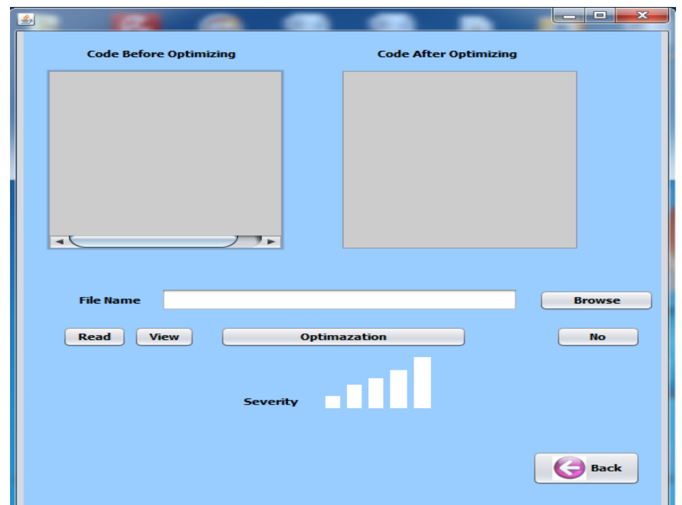

Fig. 2. Operational window for the JCOT.

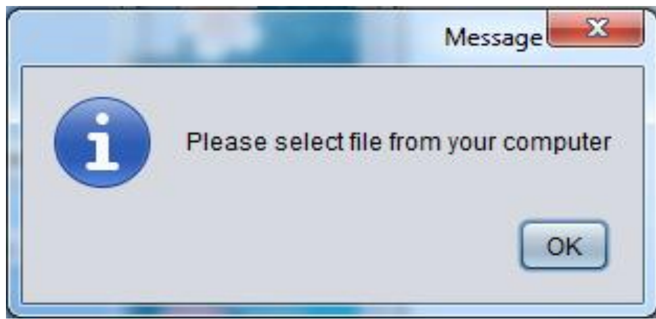


Fig. 3. Massage window.

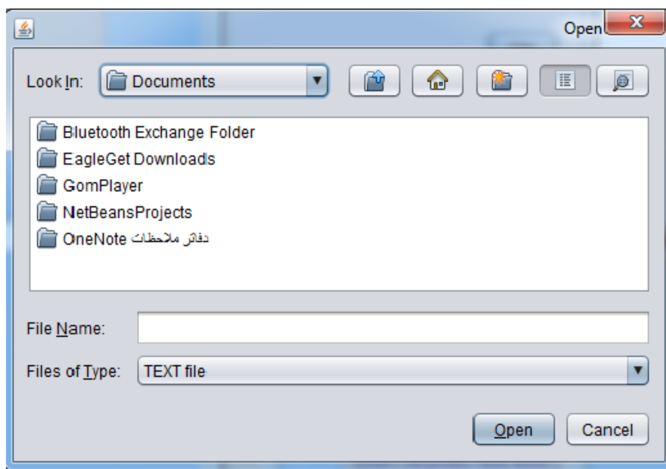


Fig. 4. Opening file window.

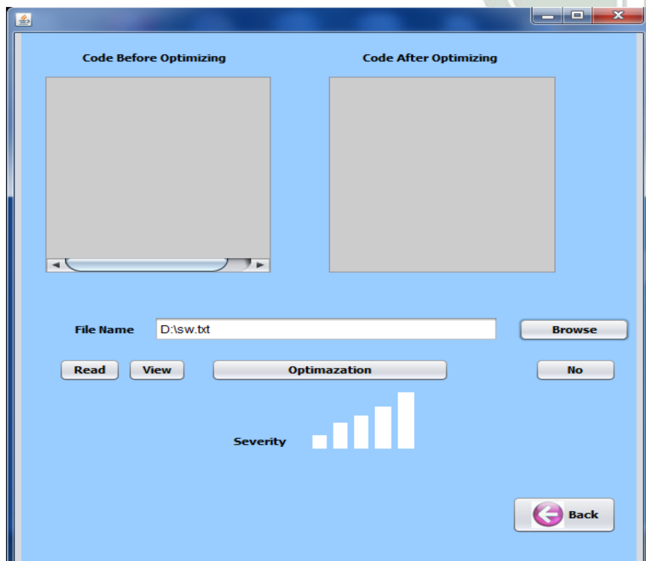


Fig. 5. Operational window for JCOT after select the file.

The programmer must select the read bottom in the same window to read and segment the code then press the view bottom to view the code segment then the user must select from applying optimization via optimization bottom or not in no bottom if the user select optimization the JCOT take the code segment before optimization.

The JCOT enable the programmer to know the severity of the techniques by show the colors indexing to that such that red color show the techniques accelerate the execution a very large margin and so on changes the color that up to yellow color that means less affect in speed of execution . As show in the figure (1-6).

The programmer can choose out of operational window by press the back bottom and return to the primary window, in first window the programmer choose the exit to close the tool and return to desktop of the computer.

## Conclusion

JCOT enables the java programmer to optimize the execution time of his/her programs. Also it gives his/her the ability to skip optimization if wonted. The severity of commends is triangulated by color and long to give the user Anne impression. JCOT follows windows for compatibility.

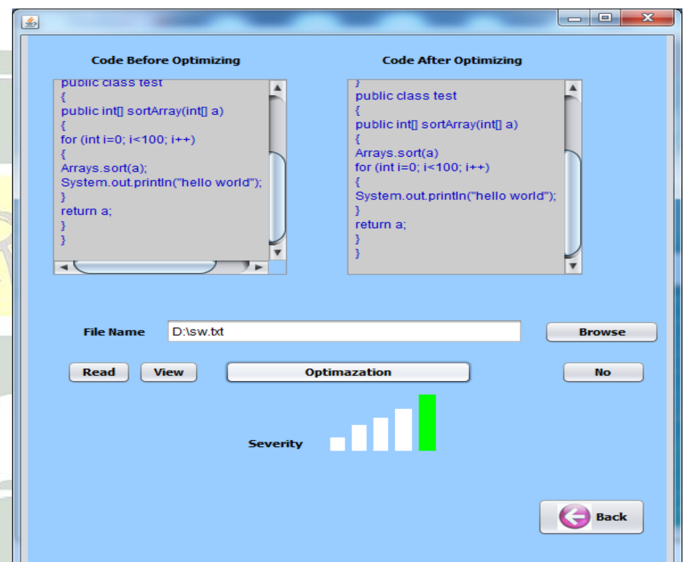


Fig. 6. Working with JCOT.

## References


[1] Michael Dorf ,(2012) , "5 Easy Java Optimization Tips " , http://www.learncomputer.com/java-optimization-tips .

[2] Maggie Johnson,(2008) ,"Code Optimization" ,Handout 20.

[3] Guihot, H. (2012). Optimizing Java Code. Pro Android Apps Performance Optimization, Springer: 1-31.

[4] Hamo Asmaa and Alsawaf Rahma,(2014)," Estimation Benefit of Java Optimization Techniques", International Journal of Enhanced Research in Science Technology & Engineering, Vol. 3 Issue 5, pp: (151-162).

[5] "Software Requirements1",2004, CS2 Software Engineering note 2.

[6] Java Performance Tunning by Jack Shirazi .

[7] http://www.javaworld.com/javaworld/jw-12-1999/jw-12-performance.html .

[8] Kevin Williams1,Albert Noll2,Andreas Gal3 and David Gregg1 ,(2008) , "Optimization Strategies for a Java Virtual Machine Interpreter on the Cell Broadband Engine"1Trinity College Dublin, Dublin, Ireland,2ETH Zurich, Zurich, Switzerland. 3University of California, Irvine, CA, USA.

[9] Huib van den Brink ,(2008), "The current and future optimizations performed by the Java HotSpotCompiler" ,Institute of Information and Computing Sciences, Utrecht UniversityP.O. Box 80.089, 3508 TB Utrecht, The Netherlands.

[10] Peter Sestoft ,(2010) ,"Numeric performance in C, C# and Java",IT University of CopenhagenDenmark,Version 0.9.1 of 2010-02-19.

[11] Tony Sintes ,(2002) ,"The String class's strange behavior explained", http://www.javaworld.com /article/2077355/core-java/don-tbe-strung-along.html.

[12] Hiroshi Inoue and Toshio Nakatani ,(2012) ,"Identifying the Sources of Cache Misses in Java Programs Without Relying on Hardware Counters ",© ACM, 2012. This is the author's version of the work.