****** **Faculty of Arts, Computing, Engineering & Sciences**

Department of Computing

Final Year Individual Project (SEGM)

[55-6727]

2015/16

|  |  |
| --- | --- |
| **Author:** | **Jordan Cain** |
| **Date Submitted:** |  |
| **Supervisor:** | **Christopher Bates** |
| **Degree Course:** | **BSc Computer Science** |
| **Title of Project:** |  |

|  |
| --- |
| **Confidentially Required?**  **NO 🞎**  **YES 🞎** |
|
|

Title page

Contents

[1 Executive Summary 5](#_Toc441610940)

[2 Research 6](#_Toc441610941)

[2.1 Possible Optimisations 6](#_Toc441610942)

[2.1.1 For Loop Unrolling 6](#_Toc441610943)

[2.1.2 Recursion Optimisation 6](#_Toc441610944)

[2.1.3 JIT – Just In Time Compilation 6](#_Toc441610945)

[2.1.4 Appropriate Primitive Types 6](#_Toc441610946)

[2.1.5 Full Object Dereferencing - Garbage Collection 6](#_Toc441610947)

[2.1.6 Deprecated Classes 6](#_Toc441610948)

[2.1.7 Cut & Paste code 6](#_Toc441610949)

[2.1.8 Printing Exceptions to Console 6](#_Toc441610950)

[2.1.9 Run()/ Start () for Threads 6](#_Toc441610951)

[2.1.10 ArrayList Reset()/ Clear() 6](#_Toc441610952)

[2.1.11 Code In-Lining 6](#_Toc441610953)

[2.1.12 Conditional Statement Ordering 6](#_Toc441610954)

[2.1.13 Constant Folding 6](#_Toc441610955)

[2.1.14 Constant Propagation 6](#_Toc441610956)

[2.1.15 Strength Reduction 6](#_Toc441610957)

[2.1.16 Common Sub-Expression Elimination 6](#_Toc441610958)

[2.2 Language choice 6](#_Toc441610959)

[2.2.1 Python 6](#_Toc441610960)

[2.2.2 Java 6](#_Toc441610961)

[2.2.3 C++ 6](#_Toc441610962)

[2.3 Abstract Syntax Tree 6](#_Toc441610963)

[2.4 Implementation Platform 7](#_Toc441610964)

[2.4.1 Eclipse Plugin 7](#_Toc441610965)

[2.4.2 Atom Package 7](#_Toc441610966)

[2.4.3 Command Line Interface 7](#_Toc441610967)

[3 Design 8](#_Toc441610968)

[3.1 Parser 8](#_Toc441610969)

[3.2 Abstract Syntax tree 8](#_Toc441610970)

[3.3 Objects Orientation 8](#_Toc441610971)

[3.4 Tree traversal 8](#_Toc441610972)

[3.5 Interface 8](#_Toc441610973)

[3.6 Testing 8](#_Toc441610974)

[4 Implementation 9](#_Toc441610975)

[5 Bibliography 10](#_Toc441610976)

# Executive Summary

# Research

The first step in producing an application that provides Java language optimisation suggestions is to produce a set of possible optimisations.

## Possible Optimisations

### For Loop Unrolling

For loops can be unrolled to increase efficiency if the number of iterations is known, the efficiency comes from the reduction in the amount of code required to run, no new variable is require, no condition tests, incrementing and GOTO statements are required (Troy Downing 1997). An example of loop unrolling:

|  |  |
| --- | --- |
| for (int i=0; i<3; i++){  checkStatus(i);  } | checkStatus(0);  checkStatus(1);  checkStatus(2); |

### Recursion Optimisation

The JVM struggles to optimise recursion in terms of performance,

### JIT – Just In Time Compilation

### Appropriate Primitive Types

Correct data types used, long and doubles require 64bits to store, the JVM operand stack is only 32bits and thus two positions on the stack are required which eats up more memory, Longs and doubles should only be used when absolutely necessary. JAVA VIRTUAL MACHINE

### Full Object Dereferencing - Garbage Collection

Objects can only be free for garbage collection when no more references to it exist, a common mistake is people leaving an object reference hanging.

E.g: A class defining a football team holds a reference to the player objects, a player may retire and not have its reference removed from the team thus occupying memory.JAVA-THE GOOD PARTS

### Deprecated Classes

Oracle warns against the use of deprecated classes as they may be removed from the standard JRE (Java runtime environment) at any time thus reducing the longevity of applications. As most IDEs already warn users against this I will no implement this. Better, Faster, Lighter Java (book)

### Cut & Paste code

Cut and paste applications typically lead to bloated applications as not all parts of the cut code is truly required, all methods should be written from scratch. Better, Faster, Lighter Java (book)

### Printing Exceptions to Console

Not all applications will have access to a console output so relying on this to print exceptions should be avoided, instead log files can be used to write exceptions to. Hardcore Java (book)

### Run()/ Start () for Threads

When trying to start a new thread by using Run() instead of Start() no new thread is created, the Start() method is what creates the new thread and then calls Run() allowing the application to run concurrently rather than sequentially.

### ArrayList Reset()/ Clear()

When you wish to clear an ArrayList it is almost more efficient to use clear() over RemoveAll() as clear() gives you O(n) performance, while removeAll(Collection c) is worse, it gives O(n^2).

### Code In-Lining

### Conditional Statement Ordering

### Constant Folding

### Constant Propagation

### Strength Reduction

I\*5 – i+I+I+I+I

### Common Sub-Expression Elimination

(I+j) \* (i+j) – t = i+j; t\*t

## Language choice

### Python

Python is a general purpose programming language with libraries for everything from networking and file I/O to Threading and Piping, it uses mutable data types (No type casting required) and uses little “Boiler plate” code so would allow for quick development.

Having never used Python before it would require an initial process of getting to grips with the language along with the common libraries. Choosing Python would also mean that the application would either have to be ran at CommandLine or converted to JavaByte code using Jython.

### Java

Java is another general purpose programming language with a huge array of libraries, it relies on a large amount of “boiler plate” code but does run on the JVM so would allow for a very portable application assuming it only needed to run on the Command Line. Java would allow for relatively easy implementation of a plugin for an IDE such as Eclipse or NetBeans, and with my existing Java knowledge would not require a large amount of learning.

### C++

C++ is another general purpose language that is closely linked to the hardware allowing for more powerful and efficient applications (Memory management, parallelism). Whilst performance is a priority for my application (i.e a quick return time of suggestions) is necessary I will be able to achieve the required level with both Java and Python. C++ compiles to an executable file so would be widely portable but would only work in a console or using a GUI I produce.

### Language Choice

For my application I have chosen to use Python, It will allow me the ability to quickly develop my application along with the chance to learn a new language, with its strong documentation and large user base learning should be with its ability to run at the Command Line I should be able to integrate it into an Atom Plugin

## Data Structure Abstract Syntax Tree

Choosing a suitable data structure to hold the java files I will be parsing is quite straight forward, Java is written like a hierarchy, a package has many classes, which has many variables and methods etc. Therefor a tree is suitable, as the tree will be based on and storing the syntax of a language it is an Abstract Syntax Tree.

## Implementation Platform

### Eclipse Plugin

Whilst the Eclipse IDE is widely used amongst Java developers the platform is difficult to create plugins for, It would require the application to be written in Java and would reduce the portability of the application as Eclipse is a heavy weight application

### Atom Package

Atom is a relatively new open source text editor “Hackable modern day editor” growing in popularity among developers, an Atom package can make use of the Command Line thus allowing for packages to be written in almost any language (assuming the language preReqs are installed). A drawback with Atom is the requirement for the User Interface to be written using the Atom API and CoffeeScript which is not yet heavily documented, the lack of documentation can be overcome by the quantity of packages that exists that are all open source so can serve as examples.

### Command Line Interface

A simple program that is called at the Command Line and prints its output to a file or to the console would a

# Design

## Parser

## Abstract Syntax tree

## Objects Orientation

## Tree traversal

## Interface

## Testing

# Implementation

# Bibliography

BRUCE TATE, Justin Gehtland (2004). *Better, faster, lighter Java*. O'Reilly.

[online]. <https://docs.python.org/3/>

JR, Robert Simmons (2004). *Hardcore Java*. O'Reilly.

MARK, Ronald (2009). *Writing Compilers and Interpreters: A Software Engineering Approach*. 3 ed., John Wiley & Sons.

MATTHEW ARNOLD, Michael Hind, Barbara G. Ryder (2002). Online feedback-directed optimization of Java..

PAUL, Javin (2014). *Java Revisited.* [online]. <http://javarevisited.blogspot.co.uk/2014/09/common-java-multi-threading-mistakes-1-run-vs-start.html>

PAUL, Javin (2015). *Java Revisited.* [online]. <http://javarevisited.blogspot.co.uk/2015/09/how-to-reset-arraylist-in-java-clear-vs-removeAll-example.html>

[online]. <https://atom.io/docs>

TROY DOWNING, Jon Meyer (1997). *Java Virtual Machine*. 1st ed., O'Reilly.

WALDO, Jim (2010). *Java: The Good Parts*. O'Reilly.

WIKIBOOKS (2011). *Python programming*. CreateSpace.