****** **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 Abstract 4](#_Toc443602068)

[2 Introduction 5](#_Toc443602069)

[2.1 Project Overview 5](#_Toc443602070)

[2.2 Aims and Objectives 5](#_Toc443602071)

[2.3 Limitations Remove this and detail specifics in the aims and objectives 5](#_Toc443602072)

[3 Requirements 6](#_Toc443602073)

[3.1 Initial requirements 6](#_Toc443602074)

[4 Research 7](#_Toc443602075)

[4.1 Possible Optimisations 7](#_Toc443602076)

[4.2 Chosen Optimisations 7](#_Toc443602077)

[4.3 Implementation Language choice 7](#_Toc443602078)

[4.4 Data Structure Abstract Syntax Tree 8](#_Toc443602079)

[4.5 AST Traversal 8](#_Toc443602080)

[4.6 Implementation Platform 8](#_Toc443602081)

[5 Design 9](#_Toc443602082)

[5.1 Abstract Syntax tree 9](#_Toc443602083)

[5.2 Parser 9](#_Toc443602084)

[5.3 Interface 9](#_Toc443602085)

[5.4 Testing 9](#_Toc443602086)

[6 Implementation 10](#_Toc443602087)

[6.1 Development approach 10](#_Toc443602088)

[7 Testing 10](#_Toc443602089)

[8 Evaluation 10](#_Toc443602090)

[9 Reflection 10](#_Toc443602091)

[9.1 Emerging requirements Move this to the reflection 10](#_Toc443602092)

[10 Bibliography 11](#_Toc443602093)

[11 Appendix 12](#_Toc443602094)

[11.1 Research - Optimisations 12](#_Toc443602095)

# Abstract

# Introduction

## Project Overview

Software development is an ever growing field with a diverse set of language choices, the TIOBE index shows Java as the most popular language of 2015 (TIOBE 2016) and Oracle states that Java has over 9 million developers worldwide (Oracle 2016). Java is a general purpose programming language designed to have as few dependencies as possible making it widely portable and understandable vastly popular. With an increasing number of self taught developers emerging due to programmes such as Code Academy (codacademy 2016) and Hour of Code (code.org 2016) it stands to reason that the quality of Java code may be deteriorating. Whilst numerous IDEs (Interactive Development Environment) exist that provide syntax checking for Java developers no IDE offers suggestions based on the code execution or quality of code (NetBeans 2016) (Eclipse Foundation 2016).

This means potentially less well educated developers and shortfall from IDEs, an opportunity exists to create an application that will provide suggestions. The suggestions could be based on execution efficiency or code readability as either would improve the code quality.

The project will be made up of multiple applications, an application to read, parse and store Java files, an application to scan the stored files and discover areas for optimisations and produce the suggestions, and finally a front end user interface to present the suggestions in a suitable environment.

## Aims and Objectives

The overall aim is to construct an application that can successfully parse a variety of Java files and produce a variety of helpful suggestions to a developer. IS this a prototype or the basis of an expandable application?

Objectives:

Research a variety of “optimisations” (areas of improvement) that can be applied to Java code.

Successfully read and store Java files of varying complexity.

Present helpful optimisation suggestions to developers writing Java code in a user friendly manner.

## Limitations Remove this and detail specifics in the aims and objectives

The project and deliverable is intended to prove that Java optimisations can be spotted and suggestions provided to improve code quality, only a number of optimisations will be possible in the time frame provided. As a result of the time restriction the application will be written with expandability in mind thus allowing new optimisations to be added at a later date.

# Requirements

A list of requirements is essential to ensure that the application is fit for developers and expandable for future additional optimisations. The requirements will also serve as tests to determine the success of the project and deliverable. Whilst this list of initial requirements is produced it is likely that further requirements will present themselves throughout the development process and will be documented under emerging requirements.

## **Initial requirements**

**Research possible optimisations to Java code the application can provide**

Research will take place to determine a comprehensive list of optimisations that apply to Java code, the list of optimisations will allow the application to focus on specific areas. The areas of the optimisations to be researched will not be restricted as this will allow the application to be specialised at a later date.

**The application can parse a Java file and store the contents in a suitable data structure.**

The application must ensure that it can store a Java file in a suitable data structure, the most suitable data structure will be a Tree. Each node of the tree will have to represent a different “block” of the Java code, for example a package, class, method etc will each be a node. The application will only be required to operate on one Java file at a time which can be assumed to only have one package but could have multiple classes. The different “blocks” of Java the application must handle are: Packages, Classes, Methods, Loops, Conditions, Try/Catch blocks and simple statements.

**The application can traverse the stored data structure and identify areas for optimisations.**

In order for the application to succeed it is essential that areas of the Java code can be inspected to determine if optimisations can be applied. The application should have the ability to traverse each element of Java code and be aware of its location (What method it is in,

**Optimisation suggestions should be presented to the developer through a User Interface (UI).**

The application should have an interface with developers in order to provide suggestions, the interface should be simple to use and not require multiple applications to be running, i.e Integrate the suggestion interface into the developers current Java editor.

**Execution time of the application should be kept to a minimum.**

It is unacceptable for the application to slow a developers workflow, thus the application should prioritise time efficiency. (Parallel stuff)

# Research

Before tackling the design and development of the application is important to first understand the parameters of the application, the parameters being the parsability, storage and traversal of Java code along with the final output of the application.

## Possible Optimisations

The first step in producing an application that provides Java optimisation suggestions is to produce a set of possible optimisations. All areas of optimisations will be considered during this stage in order to get a broad picture of current optimisation methods, from this broad list a small selection will be chosen to implement in the application.

A variety of different approaches to optimisation exist covering varies areas: Speed, Memory, Readability, portability, etc.

*Full list of the listed optimisations along with a more detailed explanation is available in the appendix of this report.*

## Chosen Optimisations

The chosen optimisations for the application will focus on the execution speed of Java, the two optimisations that will be implemented are For loop unrolling and Recursion prevention.

For loop unrolling is the process of taking a for loop that has a known number of iterations and replacing it with the same code repeated for the number of iterations. Removing the for loop prevents goto statements being required to jump back up to the start of the code block thus saving on execution time. Removal of the loop also removes the need for a local variable being created, that variable being tested against a condition and finally the incrementing/decrementing of the iterator. As there is so much overhead with a for loop it makes it a worthy candidate for the application to approach.

The JVM struggles to handle recursion effectively, it is often slower than iterative alternatives, this is due to the large overhead of creating a call stack.

## Implementation Language choice

It is essential to choose an appropriate implementation language,

**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.

**Final 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 straight forward. With its ability to run at the Command Line it should be feasible to integrate into an atom plugin without significant difficulties

The application will be developed in python as it has the advantage

## 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.

## AST Traversal

## 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

## Abstract Syntax tree

The AST will hold the java code and make it easily accessible to the application, for every Java code structure, E.g Package, Class, Method, Loop, Conditional, Try/Catch and simple statement will have a class to represent it. The classes will all have both parents and children with the exception of the simple statement which has no children and the package which should have no parent.

## Parser

## Interface

The application will produce a textual account of the optimisations it has found, it will be able to identify the location of the Java code the optimisation relates to along with providing replacement code when appropriate. An appropriate area for replacement code would be when a for loop can be unrolled, though this should be limited to only showing 5 iterations of the loop as not to bombard developers with data.

The interface will be presented within atom via an overlay popup which will output the suggestions, the suggestions will also be copied using atoms clipboard api to the developers clipboard giving them the ability to paste the suggestions into their code.

## Testing

Insert test plan here

A helpful aid in testing software applications is to have the ability to build a stack trace and user trace of the application, these traces show the path through the code. Any failing tests will need to be fixed and the easiest way to identify the area of code responsible for a failure will be through trace files. As part of the application a tracing utility should be constructed, the utility will produce text files detailing what methods have been called and the parameters passed to them.

# Implementation

## Development approach

Spiral

# Testing

# Evaluation

The application is successfully able to parse in a variety of Java files and store them appropriately in the abstract syntax tree,

# Reflection

## Emerging requirements Move this to the reflection

**The application should not only make suggestions but also offer replacement code where applicable.**

It is not enough to just tell a developer that an area of their code can be optimised the application should be able to provide the optimised code. The optimised code should also be easily accessible, i.e copied to the developers clipboard.

**The atom plugin should be fully portable.**

As atom is designed to be a portable editor it is essential to ensure that my application has no hard coded paths, as such I should ensure that the plugin is aware of the current file being edited and the runnable python application.

# Bibliography

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

CODACADEMY (2016). *codacademy.* [online]. Last updated 2016 February. <https://www.codecademy.com>

CODE.ORG (2016). *Hour of Code.* [online]. Last updated 2016 February. <https://hourofcode.com/>

FOUNDATION, The Eclipse (2016). *Eclipse.* [online]. <https://www.eclipse.org/org/>

[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..

NETBEANS (2016). *NetBeans.* [online]. <https://netbeans.org/>

ORACLE (2016). *Java.* [online]. <https://www.java.com/en/about/>

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>

TIOBE (2016). *TIOBE Software.* [online]. Last updated January 2016 January. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

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.

# Appendix

## Research - 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