****** **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 inspect each line of Java and identify in order to determine if it’s an area for optimisation.

**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.

## 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

A suitable implementation language will provide the tools to produce a successful application, a range of languages will be reviewed and the most appropriate selected.

**Python**

Python is a general purpose programming language with libraries for everything from networking and file I/O to Threading, 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**

The application will be developed using Python as it allows for rapid development and prototyping due to its lack of boiler plate code. With its ability to run on the command line it should be straight forward to integrate into an IDE or editor. Python is heavily documented and should be simple to learn in a short time frame allowing application development to start with minimum delays.

## Data Structure Abstract Syntax Tree

Choosing a suitable data structure to hold the java files 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 structure fits well, the type of tree to be used is known as an Abstract Syntax Tree (AST). An AST is a tree representation of the abstract syntax structure of code, each node of the tree is responsible for managing a construct in the code though not every detail will need to be stored, thus the abstract nature (Group 2011). ASTs are the industry standard for representing programming languages, for example both Netbeans and Eclipse, the most widely used Java editors utilise them (Foundation 2016) (NetBeans 2016).

An alternative approach to designing a custom Abstract Syntax tree would be to use an existing tool such as Antlr, Antlr is a language recognition tool that takes a set of rules (Grammar) and generates a parser that can build and walk parse trees (Parr 2016). Whilst an Application program interface (API) exists for Python to utilise Antlr it is limited to version 3.1.3 whereas the current version is 4.5.2, support is also no longer provided for the Python API.

## AST Traversal

Depth/breadth first searching

## Implementation Platform

**Eclipse/NetBeans Plugin**

Eclipse and Netbeans are the most common IDEs for Java developers, they provide comprehensive syntax guidance for developers as well as project builders (Ant) and gui builders. Developing plugins for these IDEs is complex and would limit the application implementation to a language that will run on the JVM, whilst it is possible for python to run on the JVM using Jyphon (Jyphon 2016) it would limit the python version to 2.7.

**Atom Package**

Atom is a relatively new open source text editor “A hackable text editor for the 21st century” (Team 2016) growing in popularity amongst 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 will serve as examples. With Atom and all of its packages being open sourced via Github (Kuychaco 2016) it will provide an excellent platform to make the application available to developers once completed.

**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 achieve the goal of providing suggestions to a user and would allow for the application to developed in almost any language. The Command Line interface is not user friendly and would require the developer to exit their development environment to run the application thus interrupting their workflow.

**Implementation platform choice**

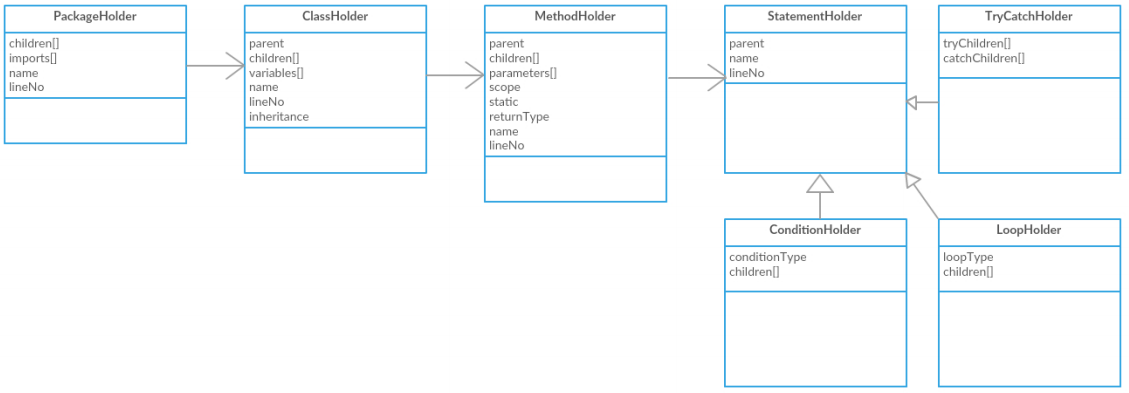
The application will be developed initially as a Command Line application, this application will then be integrated into an atom plugin. The atom plugin will have the ability to feature custom keyboard shortcuts for developers and maintain all of the developers coding inside a single editor. The application will have the ability to be indexed within all of atoms packages, making it easily accessible to Java developers.

# Design

## Abstract Syntax tree

The AST will hold the java code and make it easily accessible to the parser, each java container will have to be aware of its parent and children where applicable. The children will be implemented using Python lists which can expand and shrink as the java code is read in, parents will be implemented using object references. As Python uses Duck Typing the parent objects can be of any type thus allowing a statement to have a loop or method as its parent.

Class Diagram for Abstract Syntax Tree:

The name attributes of each class will be a string containing the actual line of Java code.

Java packages will be contained within the PackageHolder class, the application will only be designed to handle a single package so there will only ever be one packageHolder. The only special attribute a package will need is a list of imports, the name, line number and all of its children will also be stored. A PackageHolder will only have ClassHolders as children.

Classes are handled by the ClassHolder class which will need to be aware of special attributes of classes, inheritance (E.g. extends Runnable, Implements Thread) and class variables (class attributes). ClassHolder will also have to be aware of the class name, line number and all of its children. A ClassHolder will only have methods as its children although technically the variables (class attributes) are also children but will be separated into their own list of strings as they require no special operations.

All class methods will be held within MethodHolders, the MethodHolder will have to contain the parameters of a method, the scope (public/private), if the method is static(true/false) and the return type (int,String,etc). MethodHolders will also hold the method name, line number and any children, the children of a method should only be statements.

StatementHolder will hold basic statements, it is a super class which extends to three child classes, TryCatchHolder, ConditionHolder and LoopHolder. The majority of lines within a Java file will be held in statementHolders where only the parent, name and LineNo are required.

Condition, Loop and TryCatch holders all inherit from the basic statement holder, they share the simple nature of having a parent, name and line number but they also require holders for their specific children.

White space and comments within the Java are recorded as StatementHolders, there is no need to create a separate class to hold them as no operations will ever be required of them.

## Parser

To store the Java code in the abstract syntax tree each line of the Java code must be evaluated to determine the type of line. For example both a C style for loop with an iterator and a for each loop must both be detected as for loops. To evaluate each line of code Regular expression and substring matching will be applied. The use of substring matching should be limited as to avoid matching patterns in comments, for example checking if a line contains the word “package” could detect multiple packages within the Java, instead the work package followed by white space and a string, then terminating with a semicolon will be much more reliable.

To build up and test the regular expressions the website RegExr will be used (Skinner 2016).

Python contains regex libraries and will only require the import library being imported, substring matching is also build into the default string class therefore no external libraries will be required.

When the parser detects a pattern match vie Regular expression or substring match a new object will be created, if the line of code is not a simple statement the parser will have to know a new block\* has been entered. All future statements found will then have to be associated with the new block via the children attributes and their own parent attribute. To determine the end of a code block the parser will have to be aware of closing block lines “}”, once a block closer is found the parser will have to return to the parent of the new block and then continue to parse and add new children to that.

\*block referring to code encapsulated within brackets { }

Reference the paper “Understanding Source Code Evolution Using Abstract Syntax Tree Matching” (Neamtiu, Foster and Hicks 2005)

## 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.

In order for atom to call the Python part of the application it will have to make use of “Bufferd Processes” (Team 2016), the buffered process will pass a command line call to the Python code and the output will be recorded in an atom variable ready to be printed out.

## Testing

Insert test plan here Actual test plan should be moved to appendix, just talk about what testing needs to take place and a few examples.

|  |  |
| --- | --- |
| Test Description | Expected Outcome |
| Parser | |
| Detect Packages | Packages are identified and no instances of wrongful detection |
| Create PackageHolder | Package is stored with name and line number |
| Detect Imports | Imports are identified and no wrongful detections |
| Append imports to PackageHolder | Import line is stored in the imports list of PackageHolder |
| Detect Classes |  |
| Create ClassHolder |  |
| Associate Class with Package |  |
| Detect Class attributes |  |
| Detect Methods |  |
| Create MethodHolder |  |
| Associate Method with Class |  |
| Detect Method parameters |  |
| Detect Method scope |  |
| Detect if Method is static |  |
| Detect Method return type |  |
| Detect Statement |  |
| Create SattementHolder |  |
| Associate Statement with Method |  |
| Detect Try Catch blocks |  |
| Create TryCatchHolder |  |
| Detect statements and associate with try block |  |
| Detect statements and associate with catch block |  |
| Detect C style for loop |  |
| Detect foreach style for loop |  |
| Detect while loop |  |
| Detect do while loop |  |
| Create LoopHolder |  |
| Detect statements and associate with loop |  |
| Detect if statements |  |
| Detect else statements |  |
| Detect elseif statements |  |
| Detect switch statements |  |
| Create ConditionHolder |  |

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 able to read in and store a variety of Java files of varying size and complexity, the application is robust enough

# 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/>

GROUP, Object Management (2011). *Architecture-driven Modernization: bstract Syntax Tree Metamodel*. [online]. <http://www.omg.org/spec/ASTM/1.0/>

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

JYPHON, Python Foundation (2016). *Jyphon.* [online]. <http://www.jython.org/>

KUYCHACO, as-cii, joshaber, et al (2016). *Atom Github Repositry.* [online]. <https://github.com/atom/atom/>

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..

NEAMTIU, Iulian, FOSTER, Jeffrey S. and HICKS, Michael (2005). Understanding Source Code Evolution Using Abstract syntax tree matching. In: *International Conference on Software Engineering*, New York, ACM, 1-5.

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

NIEMANN, Benjamin (2010). *ANTLR Python API Documentation.* [online]. <http://www.antlr3.org/api/Python/>

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

PARR, Terence (2016). *Antlr.* [online]. <http://www.antlr.org/>

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>

SKINNER, Grant (2016). *RegExr.* [online].

TEAM, Atom (2016). *Atom Docs.* [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