JAVA CODE ANALYZER

By

Siva Kiran

Nataraj

Under the Guidance of

Dr. Akbar Namin

INDEX

JAVA CODE ANALYZER

1. Introduction:

1.1 Identification:

Software engineering is the art of crafting code into software which is intended to perform a specific task. No matter how much care is taken during software engineering, we eventually let in few bugs and faults into the system we are working on. It is inevitable. Many faults go unnoticed. It is essential for the developers to track their development process and keep a check on what they are doing. And it is also essential to stick to some general rules of practices during coding the software, so that we can avoid troubles in the later stages of the development. Secure coding principles help in achieving minimizing the number of defects in the code. CERT has provided with some standards for secure java coding, which deal with issues regarding Input Validation and Data Sanitization (IDS), Declarations and Initialization (DCL), Expressions (EXP), Numeric Types and Operations(NUM), Characters and Strings (STR), Object Orientation(OBJ), Methods(MET) and few other core java related topics. The detailed explanation about all these is presented online at https://www.securecoding.cert.org/confluence/display/java/Java+Coding+Guidelines .

Most of the development teams try to follow these conventions, but they still miss out on few of the standard rules. It is so hard to avoid missing, as the code can be intricate and can easily mislead the developers into thinking that the code is perfectly alright. This can be due to bad readability of the code. And also, there is always the factor of ‘human error’. We cannot guarantee a 100 percent defect-free code, but we can try to achieve the code with minimal defects. We clearly see the need of a tool, which can analyze the java code, and figure out whether the code is following the principles or not. Java Code Analyzer is just the right tool for this purpose. We have selected few standard rules and implemented these rules. Java Code Analyzer checks the input code and verifies if it follows the secure coding principles. If the code fails to follow the standards, it gives suggestions. It is a verification tool, which makes sure that the developers are building the tool in the right way. This is a perfect unit testing code.

1.2 Historical Background

Software development goes through various stages and it is important that each stage is error, bug free.

Some stages can be completed manually while some stages do need automation, that's when we need the help of software tools to ensure the successful completion of the stage.

We have certain tools available, for example:

- SonarQube- "an open source platform for continuous inspection of code quality."
- PMD- "A static ruleset based Java source code analyzer that identifies potential problems."
- CheckStyle- "Besides some static code analysis, it can be used to show violations of a configured coding standard."
- ThreadSafe- "A static analysis tool for Java intended to find concurrency bugs."

This project's basis is to check whether the given code conforms to CERT specification. Our project takes into account at-least two properties from the CERT secure coding standards and notifies the user about where the given input program is deviating from the course.

1.3 Document Purpose, Scope and Intended Audience:

Purpose:

The sole purpose of this document is to present the idea of Java Code Analyzer and give a detailed explanation of how it is implemented. This document explains the need of such a tool, and describes how it works.

Scope:

This document describes the outline of the project. It presents the vision and scope of Java Code Analyzer tool, and gives a description of how it can be used. It also explains the approach we have taken to build it. It shows the architectural representation, use case specification, software detailed design, requirement traceability and the tools pertaining to the project.

Intended Audience:

This document is prepared with the intention to provide a detailed description to the course instructor Akbar Siami Namin, Ph.D. (http://www.myweb.ttu.edu/asiamina/) [Advanced Empirical Software Testing and Analysis (AVESTA) Research Group & Department of

Computer Science, Texas Tech University]. This document is self-consistent to explain the details to the students and any other interested parties as well.

1.4 System and Software Purpose, Scope, Intended Users

System and Software Purpose:

The purpose of this application is to provide the developers with a tool which checks if they are following the standard rules. This makes sure that the developers are not taking the wrong path. This tool is like a checkpoint to the developers. Using this, they can be sure that they are in fact going according to the standards.

Scope:

The scope of this project is analysis of java source code, and giving feedback about the code whether it is following the standard CERT rules or not. We are inspecting a part of java coding language, and verifying if the code follows these standard rules or not. Although, it can be later extended to whole java language and can implement more of the standard rules.

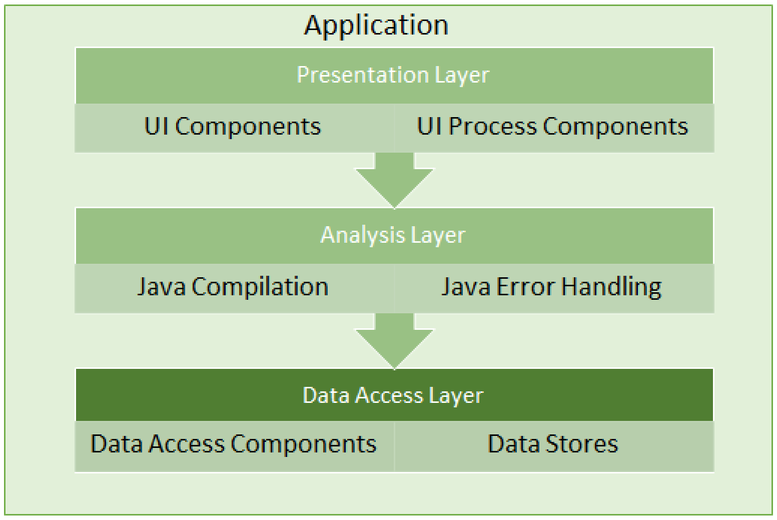
Intended Users:

This tool can be used by individual coders, software testers and software development organizations. Individual coders can constantly keep a track on their code and verify if they are following the standards or not. It will prove to be a lot of help, as it saves the strain of manual verification. Testers can constantly use this app to check if defects are caused due to digression from the standards.

2. Architectural Representation:

The Application can be broadly classified into 3 major components of architecture. Each of the component interacts with the adjacent component maintaining the flow of the system. The 3 components are:

- Presentation Layer
- Analysis Layer
- Data Access Layer



2.1 Presentation Layer:

This layer is the interface which is visible to the user. The user sends in the inputs by making use of the interface provided by the computer. We are making use of the Windows user interface here. The user can either use an IDE or use the command prompt of the windows to view and interact with the tool.

2.2 Analysis Layer:

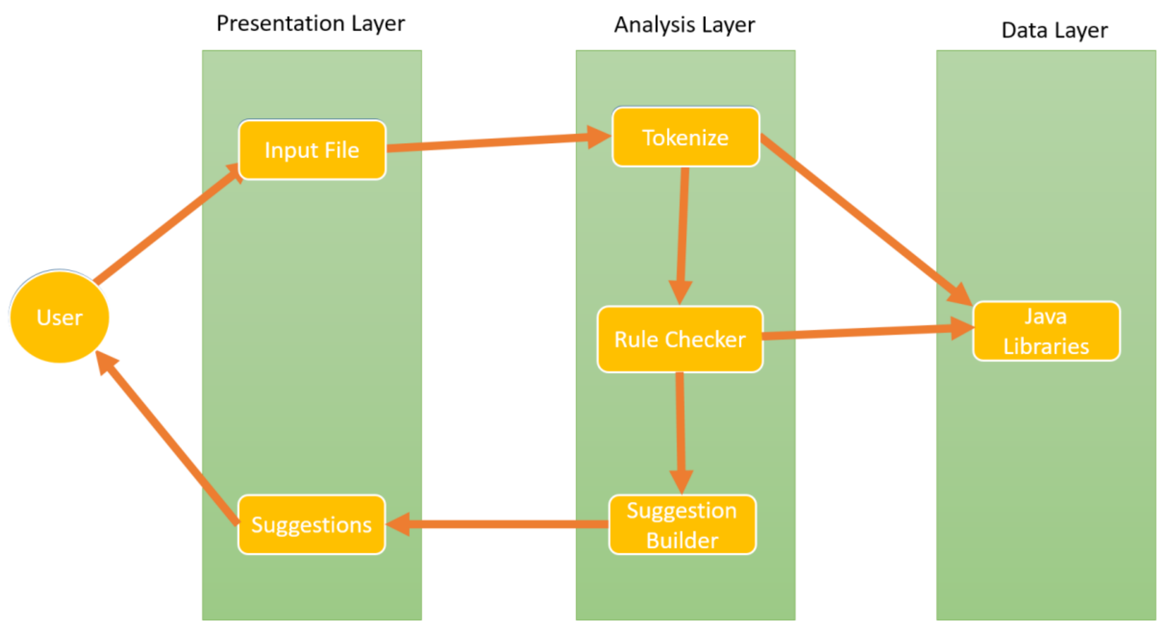
This layer is where the actual tool compilation is done. It takes in the input file that is specified by the user using the user interface. This input file is processed by the analyzer. The analyzer, on the other hand is compiled by using the local java compiler which is installed on the host system.

2.3 Data Access Layer:

In this layer, the resources requested by the tool are accessed, depending on the requirements. And also, the actual file that is being processed is accessed in this layer. On completion of required tasks, the resources suspend from access.

We have built this architecture by keeping in mind, the future of this application. We have modularized the app development, and given scope to add more features easily. This will enable the developers to add new functions without much hustle. It is very necessary, as it will give rise to opportunities for development by other developers.

2.4 Detailed Architecture:



As shown above, the user enters the input path, which is then, passed on to the Tokenizer in the Analysis layer, the Tokenizer then generates the Token file using the regular expressions.

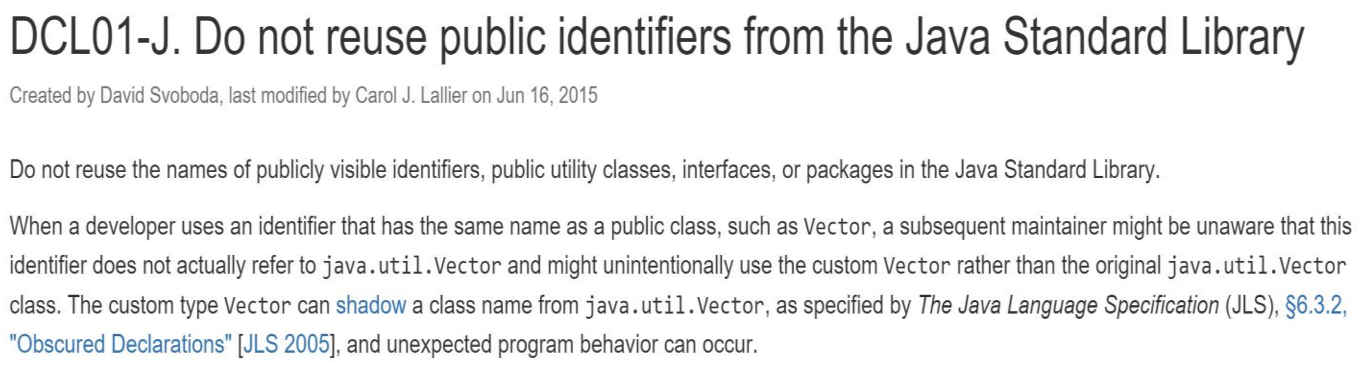
Then the Token file passes the control to the Rule Checker, which based on the logic builds the Suggestion Builder. Finally, the Suggestion Builder outputs the output in the form of suggestions.

3. Resources:

We have mentioned earlier that we are going to consider using the rules and guidelines that are specified in the CERT - Java web page. It is accessible to anyone at https://www.securecoding.cert.org/confluence/display/java/Java+Coding+Guidelines.

We have studied various rules that are mentioned by CERT and decided on working on few of the rules. Currently, we are planning on integrating at least two rules. We have found the following rules to be tangible with the tool, and can be implemented within the stipulated time. These are clearly defined in the official CERT website. Various aspects of each rule, their importance and relevance are descriptively presented in the website. Some of the tangible rules which we can implement are mentioned here.

3.1 EXP00-J. Do not ignore values returned by methods

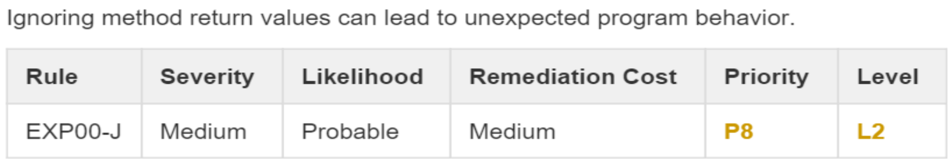


DCL01-J. Do not reuse public identifiers from the Java Standard Library

Created by David Svoboda, last modified by Carol J. Lallier on Jun 16, 2015

Do not reuse the names of publicly visible identifiers, public utility classes, interfaces, or packages in the Java Standard Library.

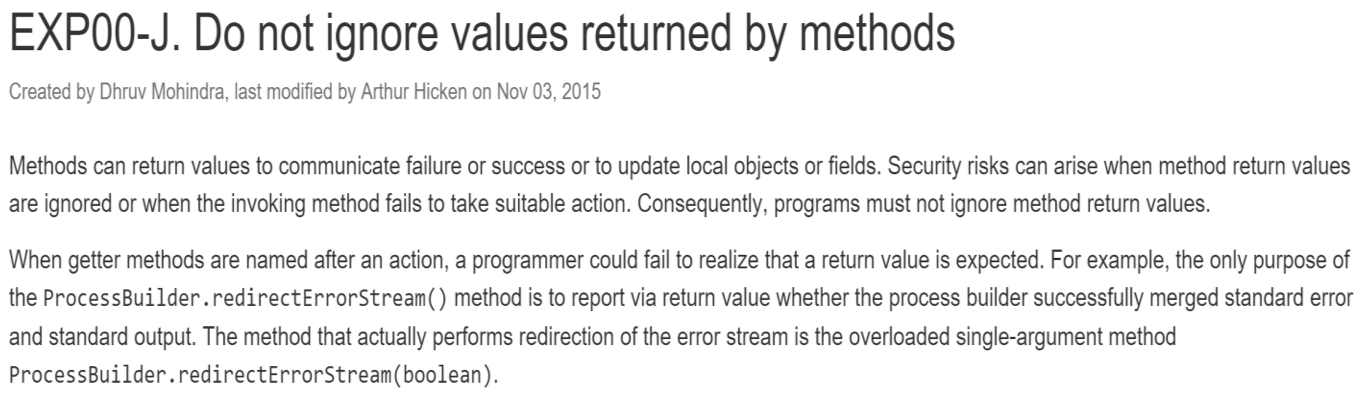
When a developer uses an identifier that has the same name as a public class, such as `Vector`, a subsequent maintainer might be unaware that this identifier does not actually refer to `java.util.Vector` and might unintentionally use the custom `Vector` rather than the original `java.util.Vector` class. The custom type `Vector` can shadow a class name from `java.util.Vector`, as specified by *The Java Language Specification* (JLS), §6.3.2, "Obscured Declarations" [JLS 2005], and unexpected program behavior can occur.



Ignoring method return values can lead to unexpected program behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP00-J	Medium	Probable	Medium	**P8**	**L2**

3.2 DCL01-J. Do not reuse public identifiers from the Java Standard Library

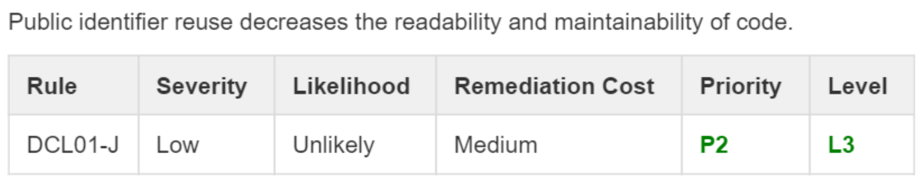


EXP00-J. Do not ignore values returned by methods

Created by Dhruv Mohindra, last modified by Arthur Hicken on Nov 03, 2015

Methods can return values to communicate failure or success or to update local objects or fields. Security risks can arise when method return values are ignored or when the invoking method fails to take suitable action. Consequently, programs must not ignore method return values.

When getter methods are named after an action, a programmer could fail to realize that a return value is expected. For example, the only purpose of the `ProcessBuilder.redirectErrorStream()` method is to report via return value whether the process builder successfully merged standard error and standard output. The method that actually performs redirection of the error stream is the overloaded single-argument method `ProcessBuilder.redirectErrorStream(boolean)`.



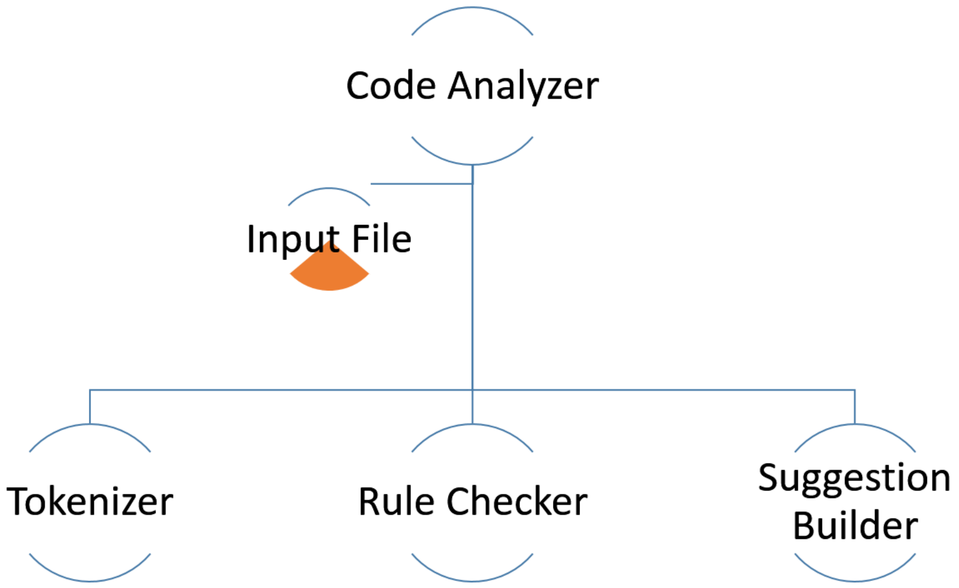
Public identifier reuse decreases the readability and maintainability of code.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL01-J	Low	Unlikely	Medium	**P2**	**L3**

4. Software Detailed Design:

We categorized the project into three broad modules. Each module carries out a specific task. The two modules are:

- Tokenizer
- Rule Checker
- Suggestion Builder



4.1 Tokenizer:

Description:

This component forms the half of the project, tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining.

For the scope of this project, we categorized the tokens into 'Function', 'Variable', 'Return', 'Method', 'Keyword'. These are present in the class 'TokenType.java'.

Input:

The tokenizer accepts a text file as an input.

Output:

Outputs a set of tokens into arrays.

Other classes of the application use these arrays. The names of these arrays in the application code are 'elements[]' and 'elementValues[]'.

4.2 Rule Checker:

Description:

The rule checker forms the other half of the project, the rule checker checks if the given program conforms to the selected CERT coding standards. The functionality of Rule Checker module is in the java class file ‘RuleChecker.java’.

Input:

The rule checker uses the arrays to perform rule checking

Output:

Rule checker just calls the ‘SuggestionBuilder.java’ methods. It performs operations on the array elements to find out if it follows rules.

4.3 Suggestion Builder:

Description:

‘SuggestionBuilder.java’ shows the functionality of this module. It has the set of suggestions, and Rule Checker calls appropriate rules.

Input:

There is no input for this. The Rule checker calls the methods in this class.

Output:

Prints what rule the code is not following, and the suggestion to correct the code. It also presents a link to learn more about the rule.

5. Requirements Traceability and Tools:

5.1 Software requirements:

Java-java 1.4 and above.

Operating system-windows 7,8,10.

5.2 Hardware Requirements:

Processor- dual core and above.

Minimum disk space-500 MB.

Recommended disk space- 1GB.

Minimum memory- 1GB.

Recommended memory- 2GB

5.3 Tools Used:

IntelliJ IDEA Community Edition 14.1.5

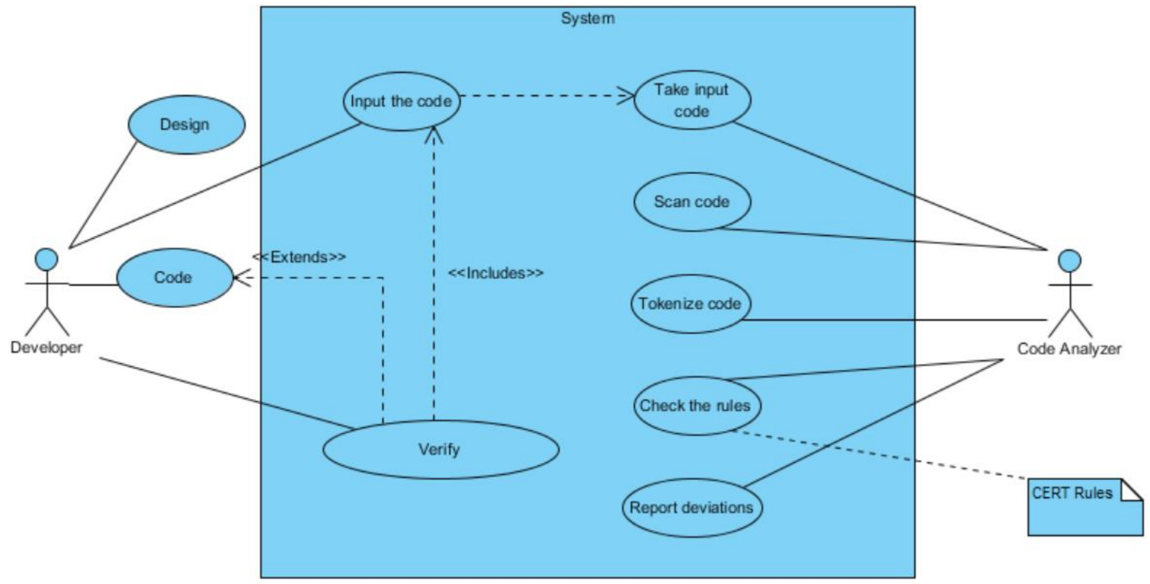
Notepad

Google Docs

MS Office

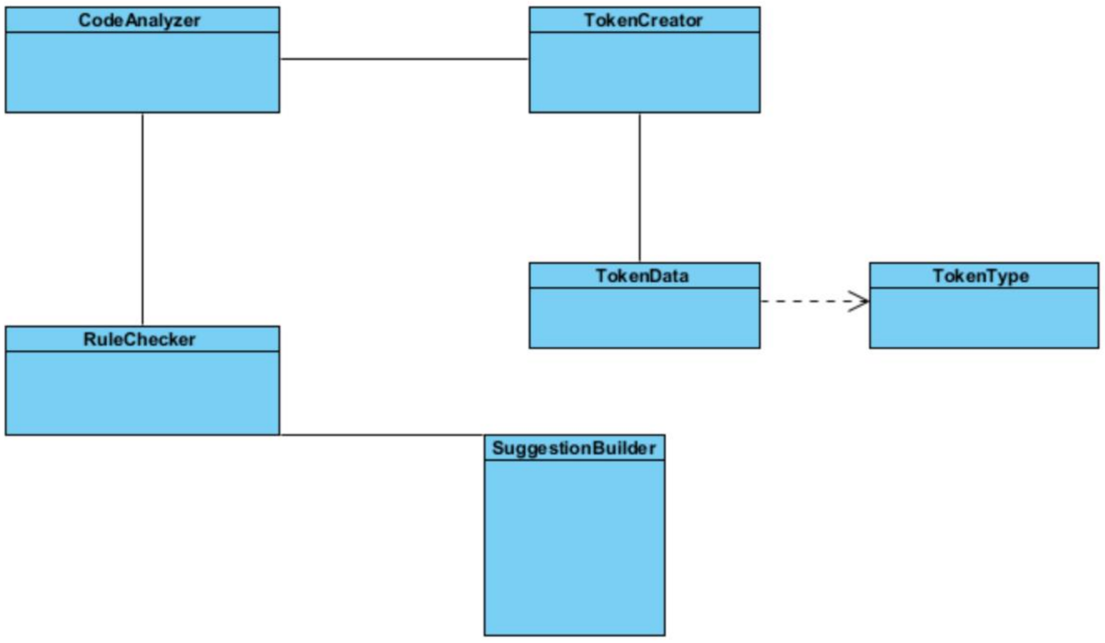
6. UML Diagrams

6.1 Use Case Diagram:



We have two actors here- The developer, who writes code, and the Code Analyzer, which analyzes the code. In the context of this application use, the developer gives the input file, which has the code, in order to verify of the code, is following secure guidelines as a part of unit testing. The code analyzer performs the actions like ‘take input code’, ‘tokenize code’, ‘check the rules’ and ‘report deviations’. The CERT rules are used to check for rules.

6.2 Class Diagram:



We have six class files. The Code Analyzer has the main() method. TokenCreator has the functionality to create tokens. CodeAnalyzer calls this. It sends the details back to the CodeAnalyzer, so that RuleChecker can use it. RuleChecker checks this data and calls SuggestionBuilder to make suggestions.

7. Application Working

We split the project into four iterations. During the first iteration, we worked on the sample which demonstrates the split of the code. We successfully showed how we are going to work on the sample code which we send to the code analyzer. During the iteration two, we worked on the actual tokenizer part, which we will discuss later. The diagram presented below gives a clear picture of the work. For Iteration 3, we successfully implemented one rule. In addition, for the final iteration, have completed another rule. The application successfully shows suggestions if there are any deviations from rules.

For the input file, we have created two deviations to check the rules. The first deviation is, we created many methods with return, but did not use them. Therefore, the rule checker identifies this and suggestion builder shows appropriate suggestion. The second deviation is, we used words like ‘Map’, which is acceptable, but for future, if the programmer imports ‘util’ package he will get errors. The rule checker identifies this and calls appropriate suggestion method.

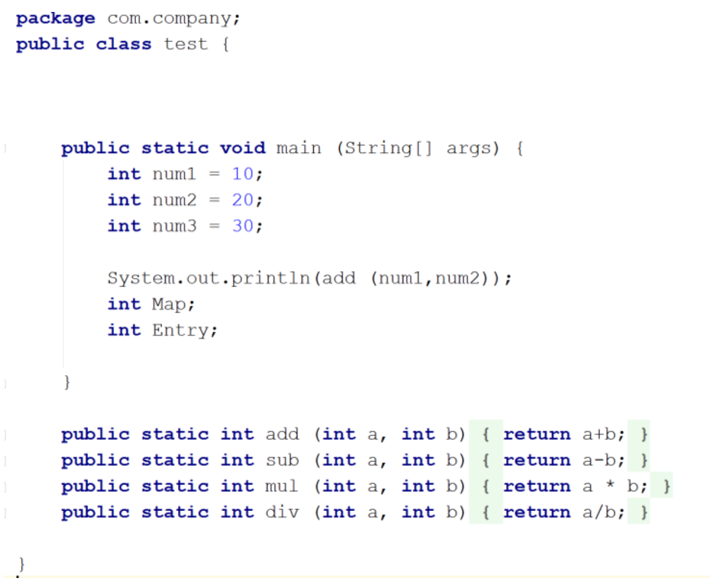
7.1 Work Progress

For the first release, we submitted a prototype, and for the second release, we successfully built the tokenizer. Using the data from the tokenizer, we made sense of the data and built rules. Suggestions are made using these rules. These rules are used and appropriate suggestions come up. These suggestions will help the users to build a secure code.

The details of the work and the progress are available on Github. We also provided the readme file, which can be used to run the project and also see the details of the project.

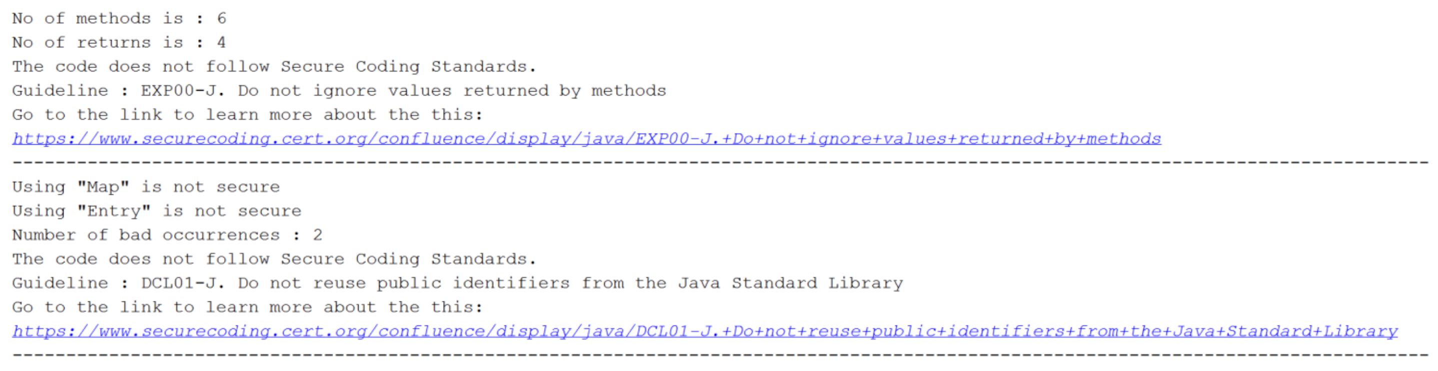
Since JAVA is a humongous language, we have focused on the Array Class.

Input: test.java (which is available in the project files)



```
package com.company;
public class test {


    public static void main (String[] args) {
        int num1 = 10;
        int num2 = 20;
        int num3 = 30;

        System.out.println(add (num1,num2));
        int Map;
        int Entry;

    }

    public static int add (int a, int b) { return a+b; }
    public static int sub (int a, int b) { return a-b; }
    public static int mul (int a, int b) { return a * b; }
    public static int div (int a, int b) { return a/b; }

}
```

Output: The output shows the suggestions for deviations. It also provides appropriate link for the guideline.



```
No of methods is : 6
No of returns is : 4
The code does not follow Secure Coding Standards.
Guideline : EXP00-J. Do not ignore values returned by methods
Go to the link to learn more about the this:
https://www.securecoding.cert.org/confluence/display/java/EXP00-J.+Do+not+ignore+values+returned+by+methods
-----------------------------------------------------------------------------------------------------------------------------------
Using "Map" is not secure
Using "Entry" is not secure
Number of bad occurrences : 2
The code does not follow Secure Coding Standards.
Guideline : DCL01-J. Do not reuse public identifiers from the Java Standard Library
Go to the link to learn more about the this:
https://www.securecoding.cert.org/confluence/display/java/DCL01-J.+Do+not+reuse+public+identifiers+from+the+Java+Standard+Library
-----------------------------------------------------------------------------------------------------------------------------------
```

7.2 Future Work:

It is possible to integrate many rules to this application, thanks to the architecture, and making use of object oriented approach. We have laid down a solid basement, and we can keep adding rules to this.

The work progress is present on Git.

Location: https://github.com/sivakiran33/Code-Analyzer.git

8. Acknowledgements:

We would like to thank our instructor Dr. Akbar Namin for giving the opportunity to work on this idea. We understood the importance of CERT guidelines from his suggestions for the project. His valuable inputs and suggestions have helped us in moving in the right direction towards the development of this project. It has been a great learning experience. Ultimately, we are able to build a solid tool, and a platform to add rules and analyze code.

9. Conclusion:

Secure coding symbolizes better software engineering. And quality software engineering results in quality products. Java code analyzer makes sure that the developers are following the secure coding principles. By making use of such a tool, the users can save a lot of time. They can focus on coding, and leave the rest to the tool, which will give feedbacks about any digressions from the rules specified by CERT. This static code analysis tool provides the developers with just the right information, and helps in building secure code.

10. References:

CERT Coding Standards:

https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java

Background Research:

https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#Java

Architectural Representation:

http://www.andromda.org/andromda-documentation/getting-started-java/application-architecture.html

Requirement Traceability:

https://docs.oracle.com/cd/E19830-01/819-4707/abqae/index.html