SOEN6011- P4

Pavit Srivatsan
40155323

August 2021

1 Debugging

Debugging is the process of detecting and removing of existing and potential errors in a software code that can cause it to behave unexpectedly or crash.

1.1 Debugging the program in Eclipse IDE

A.Setting up breakpoints

Suitable breakpoints are chosen. We must be careful in choosing breakpoints to understand the program flow. There are certain cases in which breakpoints can be skipped based on the program logic.

B.Debug Configurations

A Java program can be debugged simply by right clicking on the Java editor class file from Package explorer. Select Debug As → Java Application or use the shortcut Alt + Shift + D, J instead.

C.Debug perspective

In debug perspective we can skip breakpoints, step into, step over or resume statement executions. We can modify variable values in the variables tab in debug perspective.

1.2 Advantages

- we understand program flow and logic better
- Identify errors and bugs better instead of printing variable values
- Modify program to be more efficient

2 Quality Attributes

2.1 Correctness

The values are more precise. Since the calculator is scientific in nature, correct values are mandatory. Results are accurate as possible.

2.2 Efficiency

The algorithm chosen is Divide and Conquer Algorithm which is more efficient than Iterative algorithm. The time complexity of this algorithm (O(log(n))) is better than Iterative (O(n)).

2.3 Maintainability

The function is developed from scratch and hence it requires helper functions. These functions are placed appropriately. Necessary comments are added to make it more maintainable.

2.4 Robustness

All relevant errors are added so that users are directed to enter proper inputs. It handles various errors and exception.

2.5 Usability

The program is run in a menu driven model. Menu driven model is easy to use and comprehend.

3 CheckStyle

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard [1].

3.1 Features

Checkstyle can check many aspects of your source code. It can find class design problems, method design problems. It also has the ability to check code layout and formatting issues[1].

3.2 Advantages

- portable between IDEs. If you decide to use IntelliJ later, or you have a team using a variety of IDEs, you still have a way to enforce consistency [3].

- better external tooling. It's much easier to integrate checkstyle with your external tools since it was really designed as a standalone framework. You can plug into your SCM as a pre-commit hook, or into your build tool, quite easily. Using Eclipse style convention you would need to write or locate a plugin to do the same thing [3].
- ability of creating your own rules. Eclipse defines a large set of styles, but checkstyle has more, and you can add your own custom rules [3].

3.3 Limitations

There are basically only a few limits for Checkstyle:

- Java tokens (identifiers, keywords) should be written with ASCII characters ONLY, no Unicode escape support in keywords and no Unicode support in identifiers.[1]
- To get valid violations, code have to be compilable, in other case you can get not easy to understand parse errors.[1]
- You cannot determine the type of an expression. Example: "getValue() + getValue2()".[1]
- You cannot determine the full inheritance hierarchy of type.[1]
- You cannot see the content of other files. You have content of one file only during all Checks execution. All files are processed one by one. [1]

References

[1] Checkstyle,
https://checkstyle.sourceforge.io/index.html

[2] Debugging,
https://www.eclipse.org/community/eclipse$_n ewsletter$/2017/$june/article$1.php

[3] Stackoverflow,
https://stackoverflow.com/questions/13644624/advantage-of-using-checkstyle-rather-than-using-eclipse-built-in-code-formatter