

Assignment 1

CS409

Laurynas Sakalauskas

24th October, 2016

Overview

I have decided to implement a code smell detector with some metrics based on the ideas of many other static analyzers for writing nice looking and maintainable code. Therefore my produced analyzer can detect & collect statistics and metrics about the project mentioned below:

Code smells

1. Calculate Class Cyclomatic Complexity and limit (default: 6)
2. Calculate Weighted Method Count (WMC) and limit (default: 50)
3. Maximum number of lines per class (default: 1000)
4. Maximum number of lines per method (default: 50)
5. Maximum method name length (default: 50)
6. Maximum number of parameters in method (default: 8)
7. Maximum length of field/parameter (default: 20)
8. Maximum number of fields in the class (default: 25)
9. Maximum number of methods in the class (default: 25)
10. Minimum length of field/parameter (default: 3)
11. Minimum length of method (default: 3)

Naming Conventions

1. Boolean names must start with is (e.g. isSomething()), unless it has parameters.
2. Class must be in CamelCase.
3. Method must be in camelCase().
4. Parameters must be in camelCase.
5. Fields must be in camelCase, constants can be in UPPER_CASE.

Metrics

1. Total Code lines in project.
2. Total Classes in project.
3. Total Test/Abstract/Final classes.
4. Total Interfaces in project.
5. Total Overridden methods in project.
6. Total Methods in project.
7. Total Public/private/protected methods in project.
8. Total Methods without Javadoc
9. Total Parameters in project.
10. Total Fields in project.

Outline of design

I have used JavaParser framework's *VoidVisitorAdapter* to parse AST. Since it does not modify the project, I did not need to use any other "Visitors". Visitor pattern is being used heavily to walk through the AST and collect all the relevant statistics and warnings. There are 7 different visitor classes separated so they handle different responsibilities to make the code more maintainable, understandable and with SOLID principles in mind.

All the metrics and errors are collected in *Collector* object which holds all the data collected and can be reused throughout the project. Collector is an interface and I have added an implementation of *HashMapCollector* to simply collect the data into HashMap. In the future, if we for example want to store data to MySQL database, we can add *MysqlCollector* implementation and we would only need to change one line of code where we set the implementation to use.


I have separated different console printers to the individual classes as well, (just in case we want to skip printing some stats) namely: *ComplexityPrinter* which prints statistics about Class Cyclomatic Complexity and Weighted Method Counts. *MetricPrinter* which prints all the available metrics about the java project and finally a *WarningPrinter* which prints out all the warnings produced by running the analyzer.

With the current design it is easy to adjust the project how we want to output statistics and what else we may want analyse without modifying any collector code. Therefore, if we would like to display the results in HTML file, we can just add new Printers with some other output strategy. However, since the assignment is purely based on content, not on the style, I decided to simply output the results into console, rather than producing a nice HTML report with some graphs.

There is a *Config.java* class where all the constants are defined in one file for easy configuration change. This could be useful to adjust the limits or enable/disable some parts of the application without going and searching through the code.

Interesting parts

Well, this whole project is very interesting, since you can quickly understand the bad parts of your code and correct them. The best feeling is when your code does not throw any warnings after analyzing it. This could mean that the code follows good coding standards and in overall



can indicate that you are good programmer. Running analyzer's source code through the analyzer produces zero warning, which is great!



Results and evaluation

The analyzer application produces the relevant results for any Java project you put in. As a test cases, I have included several java projects from github that produced correct metrics and did not generate any false warnings. However, it was not clear if the assignment required any unit tests to be written, and I assumed it did not.