Proposal for GSoC 2017

Optimization of Distance Between Methods in Single Java Class

P.S. I have also submitted other proposal(s), and I would prefer this one if possible. :) Thanks!

About Me

BASIC INFORMATION

Name: Liangchen Luo
Email: luolc.witty@gmail.com
GitHub Account: Luolc
Website: http://www.luolc.com

EDUCATION

Peking University, Beijing, China
Geographic Information System Candidate, expected graduation July 2019

ISSUES FIXED

* View the merged PRs on GitHub.

The following issues have been fixed (by March 31):

- **#3172,** Regression false-positive FinalLocalVariable PR#4060
- **#4003,** Indentation UTs should not use ROOT locale when they test violation/error message PR#4020
- **#3989,** UTs should not use ROOT locale when they test violation/error message PR#3993
- **#3965,** remove from Input files "Compilable with Java8" PR#3968
- **#3961,** DetailASTTest: 'checkTree' failing on deep AST tree PR#3966
- **#3896,** Test failed due to locale message settings. (with non-English locale settings) PR#3942
- **#3700,** Control Characters are not skipped with google_checks config #PR3894
- **#3731,** expand documentation on METHOD_REF token PR#3884

RELATED EXPERIENCE

- Two-year Android development experience, proficient in Java; familiar with RxJava, MVP structure and other common Android/Java libraries, and Gradle/Maven for package dependencies management.
- Followed the principles of Clean Code and TDD in daily programming.
- Teaching Assistant of course *Algorithm and Data Structure* in 2016.
- Participant in MCM (The Mathematical Contest in Modeling) in 2017.

Project

Project Name

Optimization of Distance Between Methods in Single Java Class

Project Description

Checkstyle GSoC 2017 Project Ideas#Optimization of distance between methods in single Java class

Outline

1. DSM Matrix
   As mentioned on the project description page, there is already a matrix to evaluate the distance between methods called *methods-distance*. First of all, I will learn the mechanisms of this matrix. An in-depth understanding of the penalty function is crucial for designing an appropriate and effective algorithm.

2. Data Analysis
   Before diving into the implementation of the optimization algorithm, it is necessary to analyze a considerable amount of code. With this analysis, we will try to sum up some common properties of the codes with optimal method order. This analysis can be done manually or with some data analysis tools. For instance, we can choose some code samples and evaluate each permutation of the methods, examine the optimal ones manually, and try to find the common properties. After that, automation and visualization tools can be developed, which can be used to verify more samples quicker.

   People have many different and personalized preferences of method order and code structure. But our algorithm is likely to conflict with the preferences. We will need to list the preferences and find a way to avoid the conflict in the next step.

3. Algorithm Development
   Intuitively, since the rule "declaration before first usage" is a typical dependency problem, *topological sorting* can be a useful tool to help us adjust the order of methods. After topological sorting, we can group constructors, overloaded methods, overridden methods and accessors. At last we can adjust the distant methods. There are many places in the process that require special attention. One is the recursive calls, since the topological sorting can only handle the DAG. Another is how to group without (or as little as possible) destroying the original topological order. In fact, it is also possible that after experiments, we may find that grouping before sorting is better. Of course, this will depend on the results of processing the real code.

   We can also use the *divide and conquer* strategy. The methods can be divided into independent groups (no dependencies between any two of them). Then we can find the optimal order of each sub-methods and combine the results. The combined result is likely to be an optimal solution when every sub-methods are in the optimal order.

4. Checkstyle's Check Implementation
   When the algorithm is ready, we can start the implementation of the check. It will process the code and calculate the penalty value with the optimal method order, and then compare the value with the current one. If it can find a lower penalty value, a violation should be raised. There should be arguments which can be set by users to enforce specific rules (i.e. keep overridden methods together), which correspond to the penalty values.

After implementing the check, we will apply it to Checkstyle code base and do refactoring over the code to satisfy the check. Then we will evaluate whether the reordering results are good enough. If the mentors are not happy with the code changes, the algorithm should be improved. We will have to repeat the previous steps until the refactoring is acceptable. Then the check will be enforced over the whole Checkstyle code. The reordered code with a strict structure should also be the inputs of the UTs and the distance ratio should be in UT assertions. After that, some other projects will be evaluated with the check as well.

5. Summary Article
   After the coding part is completed, a summary article will be written to describe our achievement. The article will include the explanation of the DSM matrix, the procedure of the data analysis, the details of the algorithm and the instructions of how to use this in Checkstyle. I think my experience of writing the article on mathematical model in *The Mathematical Contest in Modeling 2017* will help me in this part. The article will be posted on my blog or wherever appropriate.

Expected Timeline

Date	Work
Prior - May 4	• Get familiar with the DSM matrix (how to process the code and calculate the penalty value) and consider if there are other potential user preferences need to be introduced to the function. • Keep diving into Checkstyle's code by fixing issues.
May 4 - 30	• Analyse code samples and try to find some common properties of the ones with optimal method order. • Develop some automation and visualization tools to help the analysis. • Discuss with mentors about the implementation of the algorithm. Investigate the classic algorithms which we can refer to.
June 1 - 30	• Implement the algorithm and do validation on more samples. • Start working on the Checkstyle check. • Write a document of evaluation.
July 1 - 28	• Fix bugs of the algorithm. • Implement the Checkstyle Check and apply it to Checkstyle code base. Find a good formula to validate Checkstyle code and apply new order to the whole code base. If the reordering results are not as good as expected, the algorithm should be improved. • Write a document of evaluation.
July 29 - August 14	• Update the document of Checkstyle. • Extend the test cases if needed.
August 14 - 29	• Write a summary article throughout the project. • Buffer for unexpected delay. • Try some more algorithms if I would have extra time to use.

Extra Information

Working Time

I will be based in Beijing, China during the summer. Therefore, I will be working in GMT +8 timezone. I believe it will take about 10 weeks for me to complete the project. But before student projects will be announced in early May, I can have a head start with some early preparation.

Reason for Participation

I have always wanted to be a great developer since the first day I learnt programming. GSoC provides me a chance to make contributions to open source projects with mentorship from great developers all over the world. I believe it is really amazing. If I have the chance to participate in GSoC and work with Checkstyle, I will try my best to complete this project.

I have read Martin's *Clean Code* before and know the significance of a good coding style, and I have been using the Checkstyle Gradle plugin to improve the quality of my own code for half a year. In the past few weeks, I have fixed several issues in Checkstyle issue tracker. By now I am familiar with the contribution workflow and how to cooperate with other developers and project maintainers. Besides, since I will continue using Checkstyle in my own projects in the future, I would like to keep maintaining and improving this feature after the program ends.

I am looking forward to working on the project wich Checkstyle!