Google Summer of Code, 2024
Proposal for Checkstyle

Project: [Refine Google Style Guide Implementation]

Name: Mauryan Kansara

GitHub: Zopsss

LinkedIn: Mauryan Kansara

Email: mauryankansaramk@gmail.com

Time Zone: Indian Standard Time (IST), UTC +5:30

Project Size: Large

Mentors: Roman Ivanov, Vyom Yadav, Andrei Paikin

TABLE OF CONTENTS

ABOUT ME

Greetings! I'm Mauryan Kansara, currently pursuing my third year in B-Tech Computer Engineering at Indus University, Gujarat, India. Over the past three years, I've immersed myself in the world of coding, cultivating a strong proficiency in Java, Python, JavaScript, and React. My strong desire to explore and continuously learn motivates me to constantly discover new fields and technologies.

My Open Source journey began in December 2023 when I started contributing to Checkstyle. Through this experience, I've gained invaluable insights into the practical application of Java in real-world projects. I gained knowledge about mutation testing, implementing new checks in checkstyle, maven plugin usage, code coverage, test case development, etc. I also got to learn about visitor design pattern and its utilization in actual projects.

Contributing to Checkstyle has not only made me better at coding, but I also got to meet some amazing folks of the community and got to learn so much from them.

I enjoy contributing to Checkstyle because it is a beginner friendly organization, well documented and it is also very active. Maintainers are always ready to help contributors to solve issues and fix bugs.

PROJECT DETAILS

Checkstyle has its Google Java Style Guide implementation, the coverage of style guide is specified at Google's Java Style Checkstyle Coverage page. The google_checks.xml is the configuration file used to implement the google style guide rules. The current implementation is very old and outdated, we need to systematically review changes such as this commit and update our implementation according to that. There are also bugs in the implemetation which are labeled as Google Style at Checkstyle's GitHub, these bugs need to be fixed too. The project also covers fixing bugs in other checks too which are not used in the Google Style Guide Implementation.

Currently, the google style guide implementation uses the per-module testing approach, we need to refactor & transition our google style guide tests to use chapter-wise testing approach. I have explained what we mean by per-module testing approach and chapter-wise testing approach in more detail later in the proposal.

Project Deliverables:

- Review and update Google Style config to the most recent content of style guide.
- Resolve all reported issues with Google Style config.
- Resolve known issues with modules/checks.
- Refactor integration tests to be chapter-wise.

The project's main aim is to update the implementation of Google Java Style Guide to the latest version, find & fix bugs in it, refactor its testing approach and also solve issues/bugs in other checks to improve the user experience.

The project details can be also found at wiki-page.

HOW GOOGLE STYLE GUIDE IS IMPLEMENTED?

We use xml files to implement Checkstyle checks based on our requirements. Similarly, Google style guide is implemented in the google_checks.xml configuration file. It contains different modules/checks and suppressions which are used to enforce different rules of the style guide. The Configuration page contains all the information about how Checkstyle's checks can be implemented with the help of an xml file.

The style guide uses various Checkstyle's checks to ensure all the rules of style guide are being followed properly. Different checks have different functionalities, like Javadoc Comments Checks ensure all the Javadoc rules are being followed, similarly Indentation Check ensures that the code is properly indented, etc. The coverage table section of the Style Guide's Coverage page contains all the checks used to implement the style guide.

Sometimes a single rule in the style guide does not always map 1:1 with Checkstyle's check. So to overcome this issue, we use suppressions to suppress or modify some functionality or property of the Check so it can follow style guide's rule. Here are all the suppressions mentioned which are used in the implementation of the style guide: link

We use a combination of different suppressions & checks to enforce the rules of google style guide.

UPDATING STYLE GUIDE TO THE LATEST VERSION

The current implementation of google java style guide is based on 23 May 2018. After that, the new version of style guide was released on 3 Feb 2022. The project aims to update the style guide implementation based on the latest release.

This is the diff commit which shows all the parts of style guide which was updated: link

There were many documentation updates and rules updates in the latest version, below I've listed all the sections which were updated, renamed or newly introduced:

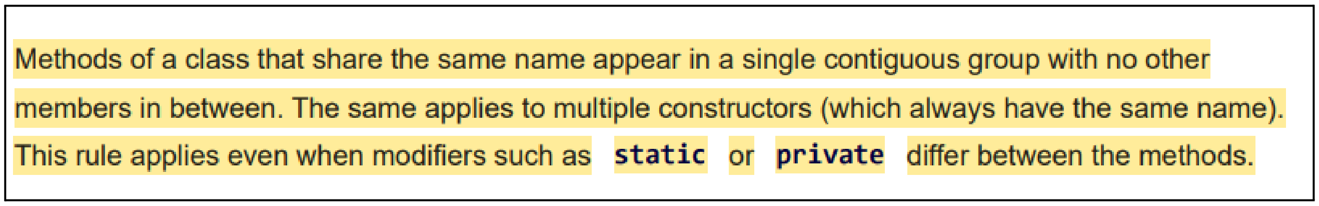
NOTE:- I've excluded small documentation updates which had nothing to do with rules or the implementation.

- 3.4.2.1 - Overloads: never split
- 4.1.1 - Use of optional braces
- 4.1.2 - Nonempty blocks: K & R style
- 4.4 - Column limit: 100
- 4.6.2 - Horizontal whitespace
- 4.8.4.3 - Presence of the default label
- 4.8.5.1 - Type-use annotations
- 4.8.5.2 - Class annotations
- 4.8.5.3 - Method and constructor annotations
- 4.8.5.4 - Field Annotations
- 4.8.5.5 - Parameter and local variable annotations
- 7.1.2 - Paragraphs
- 7.3.1 - Exception: self-explanatory members
- 7.3.4 - Non-required Javadoc

All the updated parts of the rules mentioned above are explained here:

(I've highlighted the updated parts of the rule)

3.4.2.1 - Overloads: never split


Methods of a class that share the same name appear in a single contiguous group with no other members in between. The same applies to multiple constructors (which always have the same name). This rule applies even when modifiers such as `static` or `private` differ between the methods.

It requires having overloaded methods grouped together and all overloaded constructors grouped together.

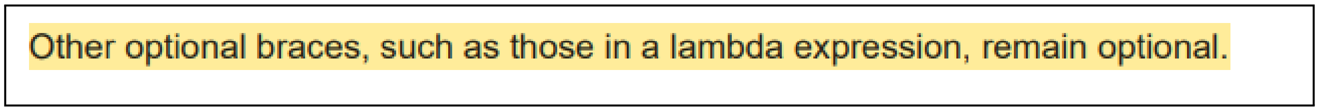
We have support for checking grouping of overloaded methods in the implementation, the OverloadMethodsDeclarationOrder module is used for this requirement.

But there is no module to check for grouping of overloaded constructors. New module needs to be created for this. I've opened an issue for adding a new module named ConstructorsDeclarationGrouping to check for grouping of overloaded constructors. This new module will be used to full-fill the requirements of this rule. I've already started working on the implementation of this new module, here its PR link: link

I've also opened an issue to add support for checking grouping of overloaded constructors in google_checks.xml, here is the link of that issue: link, the work on this issue will begin after the ConstructorsDeclarationgrouping module is implemented.

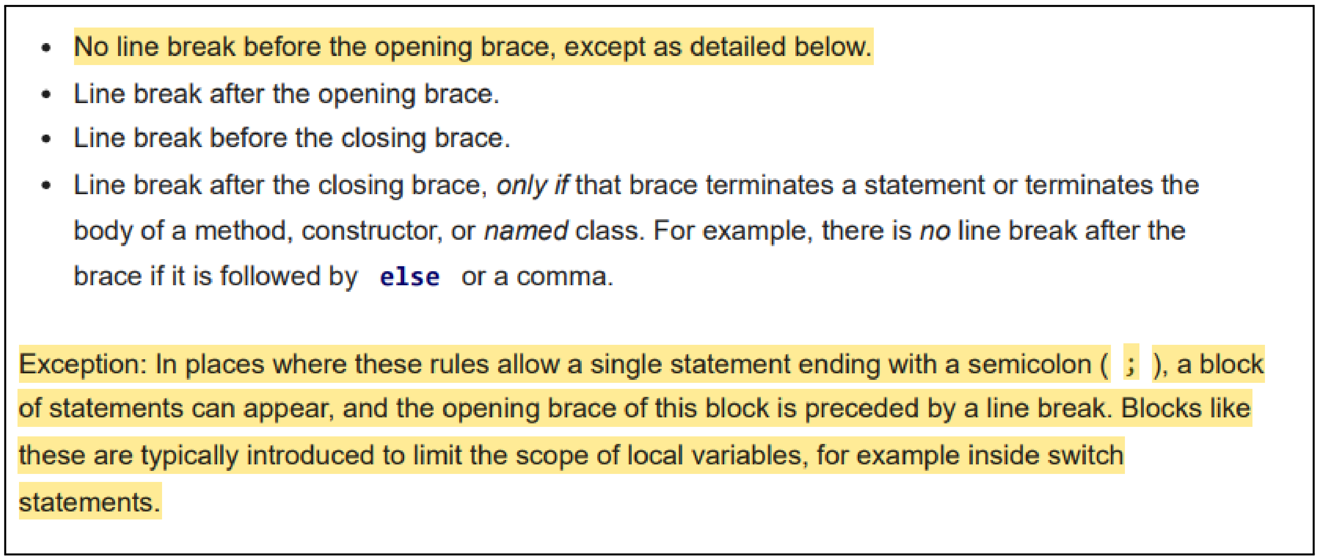
4.1.1 - Use of optional braces

This section was renamed from `4.1.1 Braces are used where optional` to `4.1.1 Use of option braces`, no rules were changed.


Other optional braces, such as those in a lambda expression, remain optional.

The implementation already has support for ignoring option braces in lambda. New test cases will be added for this rule to make sure everything is working fine and all the latest rules are being followed flawlessly.

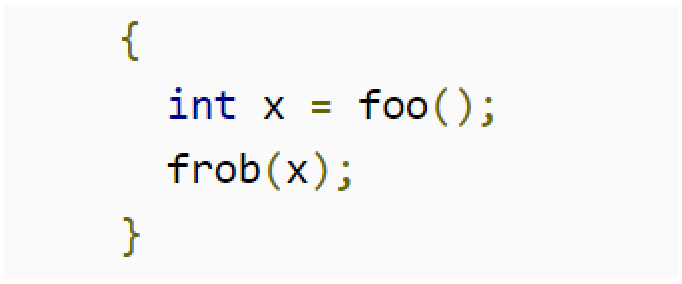
4.1.2 - Nonempty blocks: K & R style



- No line break before the opening brace, except as detailed below.
- Line break after the opening brace.
- Line break before the closing brace.
- Line break after the closing brace, *only if* that brace terminates a statement or terminates the body of a method, constructor, or *named* class. For example, there is *no* line break after the brace if it is followed by `else` or a comma.

Exception: In places where these rules allow a single statement ending with a semicolon (`;`), a block of statements can appear, and the opening brace of this block is preceded by a line break. Blocks like these are typically introduced to limit the scope of local variables, for example inside switch statements.

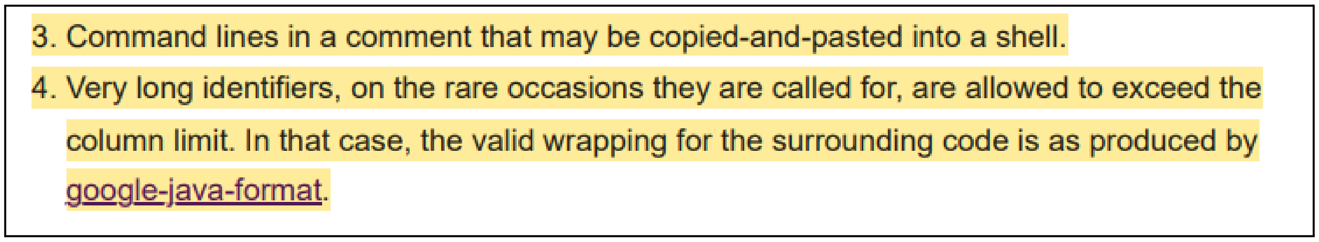
This rule is about placements of curly braces, normally the opening curly braces are not followed by a line break but the exception part says that if a block of statement appears, then its opening curly brace needs to be preceded by a line break, otherwise the . For example:-



```
    {
      int x = foo();
      frob(x);
    }
```

If this type of block occurs, the opening curly brace of this block is preceded by a line break. But I'm not so sure about this rule, so it'll need a discussion with the maintainer.

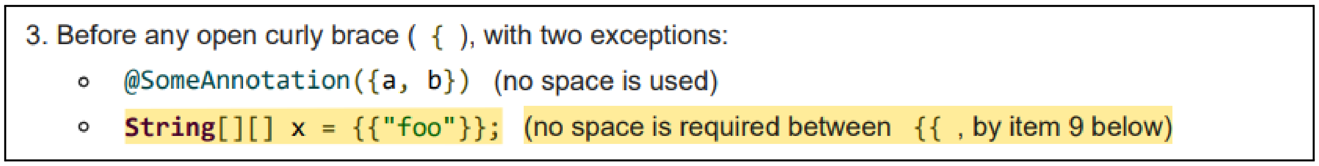
4.4 - Column limit: 100



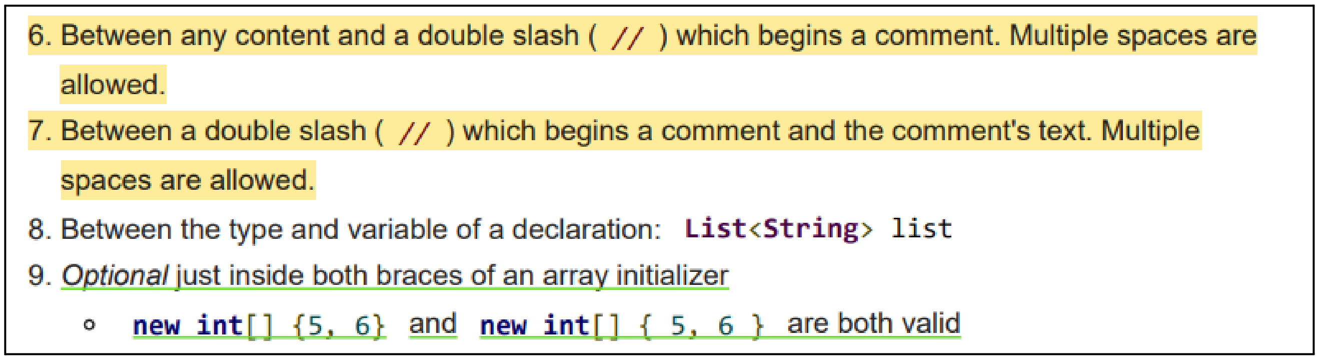
3. Command lines in a comment that may be copied-and-pasted into a shell.
4. Very long identifiers, on the rare occasions they are called for, are allowed to exceed the column limit. In that case, the valid wrapping for the surrounding code is as produced by google-java-format.

I'm not so sure about this rule, it'll need a discussion with maintainers.

4.6.2 - Horizontal whitespace



3. Before any open curly brace (`{`), with two exceptions:
   - `@SomeAnnotation({a, b})` (no space is used)
   - `String[][] x = {{"foo"}};` (no space is required between `{{` , by item 9 below)



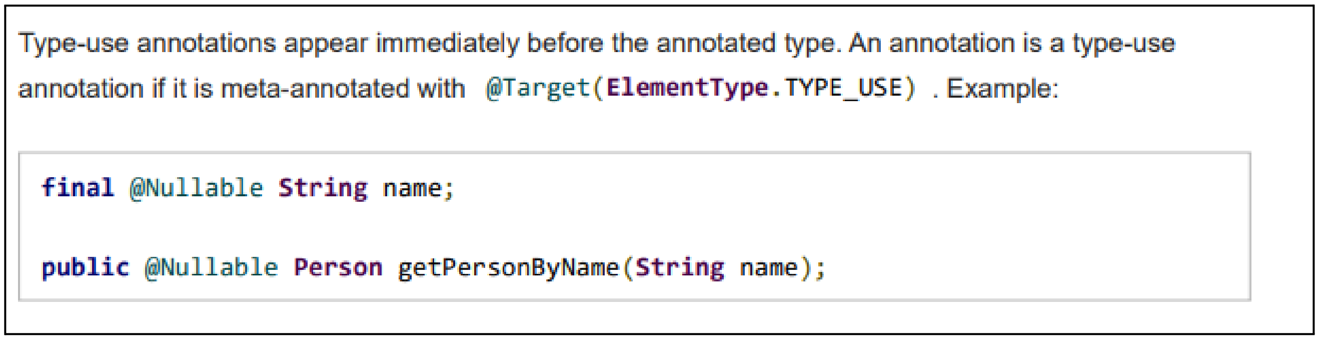
6. Between any content and a double slash (`//`) which begins a comment. Multiple spaces are allowed.
7. Between a double slash (`//`) which begins a comment and the comment's text. Multiple spaces are allowed.
8. Between the type and variable of a declaration: `List<String> list`
9. *Optional* just inside both braces of an array initializer
   - `new int[] {5, 6}` and `new int[] { 5, 6 }` are both valid

All of these updated rules are already covered in the implementation. New test cases will be added for this rule to make sure everything is working fine and all the latest rules are being followed flawlessly.

4.8.4.3 - Presence of the default label

This section was renamed from `4.8.4.3 The default case is present` to `4.8.4.3 Presence of the default label`, no rules were changed.

4.8.5.1 - Type-use annotations



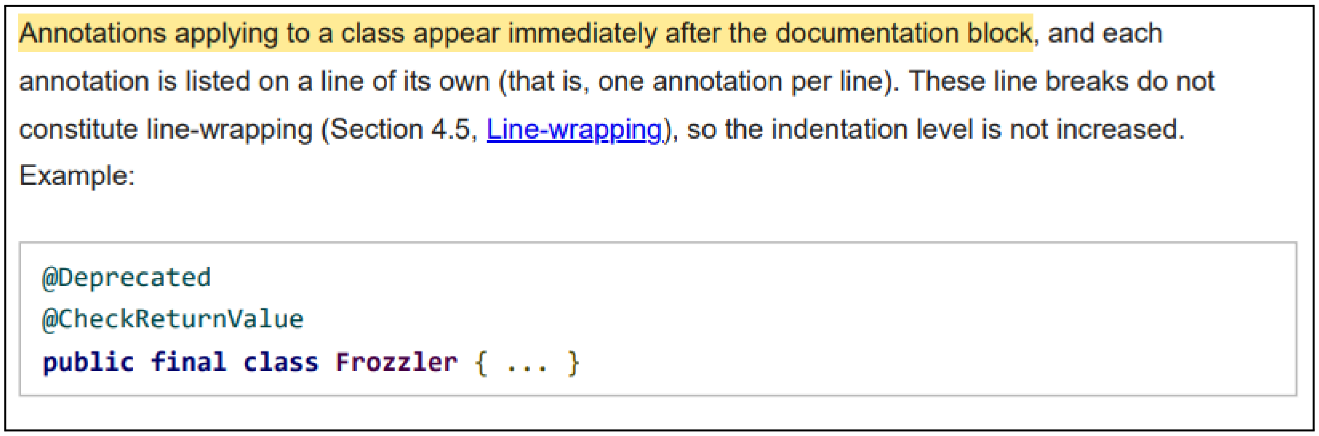
Type-use annotations appear immediately before the annotated type. An annotation is a type-use annotation if it is meta-annotated with `@Target(ElementType.TYPE_USE)` . Example:

```java
final @Nullable String name;

public @Nullable Person getPersonByName(String name);
```

This is a newly added section, it states that type-use annotation should come immediately before the annotated type. Currently in Checkstyle, there is no way to detect type-use annotations, so this will require a discussion with the maintainers.

4.8.5.2 - Class annotations

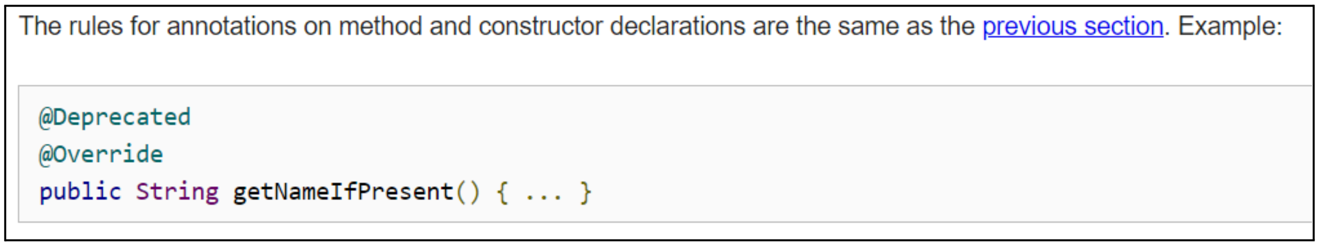


Annotations applying to a class appear immediately after the documentation block, and each annotation is listed on a line of its own (that is, one annotation per line). These line breaks do not constitute line-wrapping (Section 4.5, Line-wrapping), so the indentation level is not increased. Example:

```java
@Deprecated
@CheckReturnValue
public final class Frozzler { ... }
```

The highlighted line was newly added, it states that the class annotations should come immediately after the documentation comment, this rule is already covered in the current implementaion. The existing test cases will be updated to make sure the new requirement is being followed nicely.

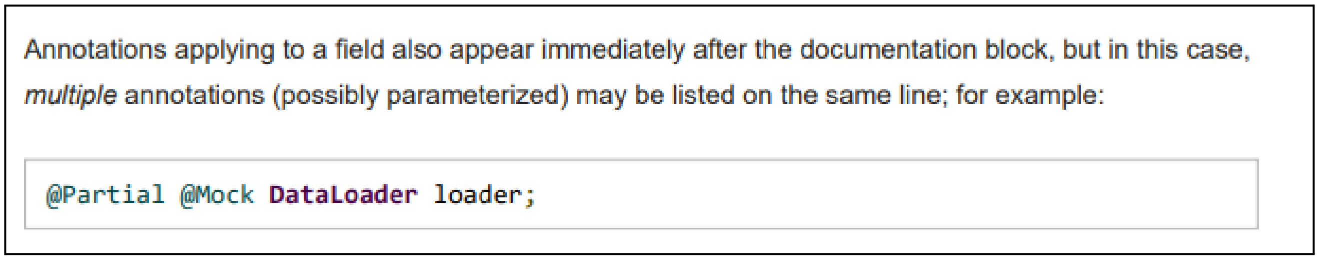
4.8.5.3 - Method and constructor annotations


The rules for annotations on method and constructor declarations are the same as the previous section. Example:

```
@Deprecated
@Override
public String getNameIfPresent() { ... }
```

This section was newly introduced, the same rules of the Class Annotations section applies here, this is also covered in the current implementation. The exception part is also covered. New test cases will be created for this section and the coverage page will be updated accordingly.

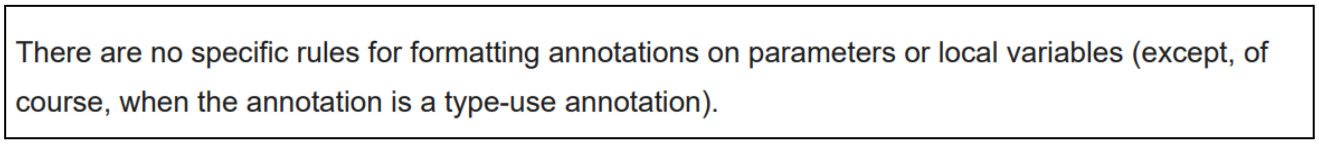
4.8.5.4 - Field Annotations


Annotations applying to a field also appear immediately after the documentation block, but in this case, *multiple* annotations (possibly parameterized) may be listed on the same line; for example:

```
@Partial @Mock DataLoader loader;
```

This section was newly introduced, all the rules of this section are already covered in the current implementation. Same as the above section, new test cases will be created for this section and the coverage page will be updated accordingly.

4.8.5.5 - Parameter and local variable annotations


There are no specific rules for formatting annotations on parameters or local variables (except, of course, when the annotation is a type-use annotation).

This section was newly added, we need to check type-use annotation's placement, but currently Checkstyle does not support to detect type-use annotation, so this will require a discussion with maintainers.

7.1.2 - Paragraphs

> One blank line—that is, a line containing only the aligned leading asterisk (`*`)—appears between paragraphs, and before the group of block tags if present. Each paragraph except the first has `<p>` immediately before the first word, with no space after it. HTML tags for other block-level elements, such as `<ul>` or `<table>` , are *not* preceded with `<p>` .

The highlighted part was newly added, it states that block-level elements like `<ul>` or `<table>` tag should not be preceded with the `<p>` tag, for example:

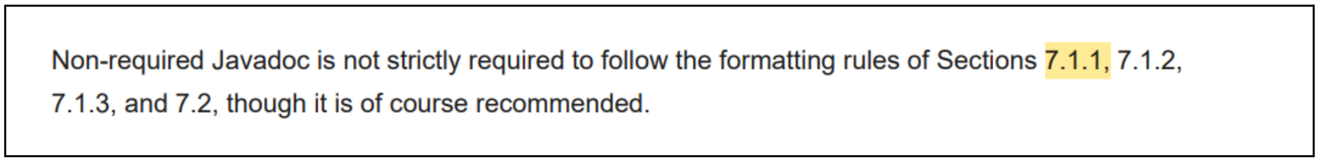
```
/**
 * <p><ul>
 *
 * </ul></p>
 */
```

This should give an error because `<ul>` tag is preceded by `<p>` tag. This new rule is not covered in the current implementation, a new module will probably need to be created to enforce this rule. This will require a discussion with the maintainers.

7.3.1 - Exception: self-explanatory members

This section was renamed from 7.3.1 - Exception: self-explanatory methods to 7.3.1 - Exception: self-explanatory members, no rules were changed.

7.3.4 - Non-required Javadoc



Non-required Javadoc is not strictly required to follow the formatting rules of Sections 7.1.1, 7.1.2, 7.1.3, and 7.2, though it is of course recommended.

A new section 7.1.1 was added in this rule. This rule states that the Non-required Javadoc does not need to strictly require to follow formatting rules of mentioned sections. I’m not so sure about this rule, it’ll need a discussion with the maintainers.

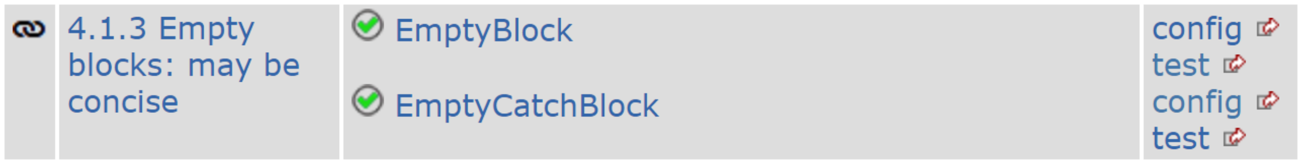
REFACTORING GOOGLE TESTS

All the tests for google style guide implementation are located in this directory.

Refactoring of google style guide tests includes transitioning from using per-module (module means a check) testing approach to use chapter-wise testing approach. I've explained per-module testing and chapter-wise testing below:

PER-MODULE TESTING :-

Sometimes, Checkstyle requires using more than one module to follow requirements of a particular section of the google style guide as the requirements & Checkstyle's check does not always map 1:1 with each other. For example, the 4.1.3 Empty blocks: may be concise section of the 4.1 Braces rule uses two modules under the hood:



🔗	4.1.3 Empty blocks: may be concise	✓ EmptyBlock ✓ EmptyCatchBlock	config test config test

It uses EmptyBlock and EmptyCatchBlock for its implementation. There are two different test files used to test this section, one for the EmptyBlock module and other for the EmptyCatchBlock module. Here is the link to the directory which contains test files of this rule: link.

Having different test files for each module that the section is using for its implementation is called per-module testing. This type of testing approach is not efficient as we're using different test files to test the different requirements of a section. It does not always guarantee that the section is tested perfectly and there is no bug in the implementation of it.

CHAPTER-WISE TESTING :-

Chapter-wise testing involves using one file to write tests for all modules a section uses instead of different test files for different modules. It includes combining different modules into one test file to test the implementation of a specific section. For Example, for the 4.1.3 Empty blocks: may be concise section mentioned above, there will be only one test file which will test all the requirements of that section instead of having multiple test files for each module it uses.

Example: Below is the code of one of my PR which shows how we can combine different modules to test a single rule:

```
    @Test
    public void testLineLengthJsniMethods() throws Exception {
        final String[] expected = {
            "14: " + getCheckMessage(LineLengthCheck.class, "maxLineLen", 100, 165),
        };

        final Configuration lineLengthConfig = getModuleConfig("LineLength");
        final DefaultConfiguration rootConfig = createRootConfig(lineLengthConfig);

        final Configuration filterConfig =
getModuleConfig("SuppressWithPlainTextCommentFilter");
        rootConfig.addChild(filterConfig);

        final String filePath = getPath("InputLineLengthJsniMethods.java");

        final Integer[] warnList = getLinesWithWarn(filePath);
        verify(rootConfig, filePath, expected, warnList);
    }
```

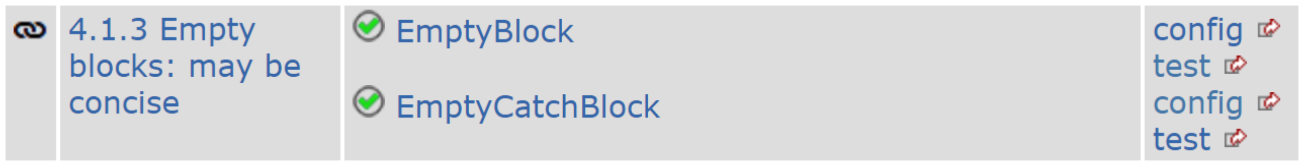
Here we're combining two different modules to test a single section:

- LineLength
- SuppressWithPlainTextCommentFilter

The PR was related to the 4.4 Column limit: 100 section of the rule 4.1 Braces.

As mentioned above, google style guide tests follows per-module testing approach and we need to refactor it to use chapter-wise testing approach. It will require combining different test files used to test different modules of a single section into one file and modify the test cases according to that and also update the coverage page documentation to add the link of the corresponding test file to each section. The coverage page will look something like this:

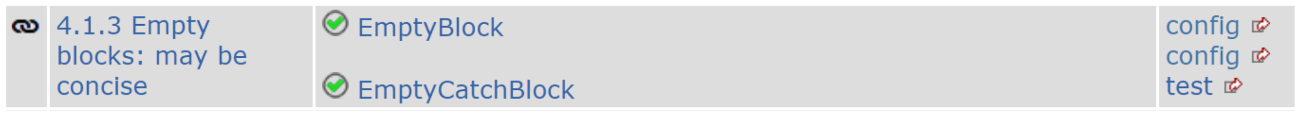
Before:



	4.1.3 Empty blocks: may be concise	✓ EmptyBlock ✓ EmptyCatchBlock	config test config test

Here, there are 2 test links which refer to the 2 test files for different modules used in this section.

After:



	4.1.3 Empty blocks: may be concise	✓ EmptyBlock ✓ EmptyCatchBlock	config config test

Here, there is only one “test” link because we combined the two different test files into one.

Following are the sections which uses more than one module for its implementation:

- 3.2 - Package statement
- 3.3.2 - No line-wrapping
- 4.1.2 - Nonempty blocks: K & R style
- 4.1.3 - Empty blocks: may be consise
- 4.5.1 - Where to break
- 4.6.2 - Horizontal whitespace
- 5.2.6 - Parameter names
- 5.2.7 - Local variable names
- 5.2.8 - Type variable names
- 7.1.1 - General form
- 7.1.2 - Paragraphs
- 7.1.3 - Block tags
- 7.3 - Where Javadoc is used
- 7.3.1 - Exception: self-explanatory methods
- 7.3.2 - Exception: overrides

These all sections will be reviewed and all their different test files will be converted into one test file for their corresponding sections.

FIXING ISSUES RELATED TO GOOGLE STYLE GUIDE

The google style guide implementation is not complete and it also contains many bugs. This project covers resolving all issues related to the style guide implementation, finding bugs, improving the coverage by reviewing the coverage page and adding support for functionalities which are not covered.

All the google style guide related issues are labeled as google style at Checkstyle's repository for easy filtration. These issues will be reviewed and resolved as a part of this project.

FIXING ISSUES RELATED TO OTHER CHECKS/MODULES

There are many checks/modules which require implementing new functionalities, features, resolving bugs and many other things. There are different labels used like new feature, bug, javadoc, miscellaneous, etc for easy filtration of the issues. This project covers reviewing & inspecting as many these types of issues as possible and solving them before the deadline.

PROJECT TIMELINE

Weeks	Start date	End date	Tasks to be completed
-	May 1	May 26	Community Bonding period
1	May 27	June 2	Adding support for ConstructorsDeclarationGrouping module in the style guide implementation. The discussion for implementing the new module is already completed, here is the issue link which contains all the information about the new module: link
2	June 3	June 6	Renaming section 4.1.1 & working on section 4.1.2
3	June 7	June 13	Working on section 4.4 & renaming 4.8.4.3
4	June 14	June 19	Working on 4.8.5.1
5	June 20	June 30	Implementing a new module for section 7.1.2 and adding support for it in the style guide implementation.
6	June 1	July 7	Renaming section 7.3.1 & working on section 7.3.4
7	July 8	July 18	Refactoring section 3.2, 3.3.2, 4.1.2, 4.1.3 & 4.6.2
8	July 19	July 28	Refactoring section 4.5.1, 5.2.6, 5.2.7, 5.2.8, 7.1.1
9	July 29	August 3	Refactoring section 7.1.2, 7.1.3, 7.3, 7.3.1, 7.3.2
10	August 4	August 11	Solving issues related to Google Style Guide Implementation.
11	August 12	August 21	Solve issues related to other checks and review them to check if they need new functionalities.
12	August 22	August 26	The project will be reviewed and final reports will be submitted to the mentors.

Different sections require different amounts of time to complete, so I have assigned different numbers of days to each section based on their requirements.

I'll be committing 35-40 hours a week to complete this project before the deadline. I might get mid-term exams during the GSoC time period, at that time I will be able to spend only 2-3 hours a day on the project but I'll make sure that no work gets delayed. The exam dates are not announced yet so I'm not so sure when they will occur.

During the community bonding period, I will discuss all the sections in which I did not have a clear idea about the requirements with maintainers, which new modules or functionalities needs to be implemented, what steps we should take to transform the style guide tests to use chapter-wise testing approach, etc. I will start sending PRs as early as possible because I'll have to cover up for my mid-term exams.

This is a long project and I'll try to finish this before the deadline but if because of some circumstances this project gets delayed, I would request maintainers to extend the timing period according to the needs.

EXPECTATIONS FROM MENTORS

This project requires implementing new modules and functionalities, in which I will need to have maintainers opinions and get help from them if I get stuck over something.

I need maintainers to take time for reviewing the new functionalities or modules and give valuable feedback so I can complete my tasks in an efficient way.

CONTRIBUTIONS SO FAR

Sr	Pull Requests	Status
	New modules & functionalities	
1	Issue #6723: new check: LeadingAsteriskAlignCheck	Reviewing
2	Issue #14726: new check: ConstructorsDeclarationGroupingCheck	Reviewing
3	Issue #14487: add support to ignore JSNI methods for LineLength check in google config	Reviewing
	Pitest Issues	
4	Issue #14019: Kill mutation for setSeverity in Checker	Merged
5	Issue#13999: Resolve pitest suppression for throwAst.getParent() of JavadocMethod	Merged
6	Issue#13999: Resolve pitest suppression for throwAst.getParent() of JavadocMethod	Merged
7	Issue #13999: Resolve pitest suppression for index += 2; of JavadocStyleCheck	Merged
	XPath Regression Testing	
8	Issue #6207: Add Xpath Regression Test for JavaNCSS	Merged
9	Issue #6207: Add Xpath Regression Test for FinalParameters #14223	Merged
	Enforce file size, Enable examples tests & Remove '//ok' comments	
10	Issue #13345: Enable examples tests for NewlineAtEndOfFile	Merged
11	Issue #11163: Enforce file size for NoEnumTrailingComma	Merged
12	Issue #13213: Remove '//ok' comments from Input files (InputNoEnumTrailingComma)	Merged

Supplemental & Minor		
13	supplemental: Issue#14273 Add JUnit test coverage for overloaded private and static methods for OverloadMethodsDeclarationOrder	Merged
14	supplemental: refactor misaligned leading asterisks in javadoc comments #14722	Merged
15	minor: corrected leading asterisks alignment in javadoc #1035	Merged
16	add new check: JavadocLeadingAsteriskAlign #848	Merged

Here all my PRs are listed: link

Issues I opened:

- new Check: ConstructorsDeclarationGrouping to check the grouping of overloaded constructors
- Update google_checks.xml to check order of overloaded constructors
- Add support to ignore JSNI methods for Google style config

WHY ME?

Good question. I’ve been actively contributing to Checkstyle since December 2023 and I’ve dedicated the last 2-3 months doing research on this project and have a clear mindset to complete this project.

I’ve mostly worked around google style guide implementation, discussed with maintainers about how we should add new checks and add support for new functionalities to improve the current implementation.

This project requires a deep understanding of how the google style guide is implemented and tested, how new checks/modules are implemented and how Checkstyle works under the hood, etc. From doing research for the last 2-3 months, I’ve gained knowledge about all these topics and I’ve also worked on implementing new modules and functionalities which are the hardest issues to work on. I’ve also opened issues about adding new modules and functionalities which are required for updating the google style guide implementation and currently planning to open more issues soon.

From my experience and knowledge, I’ll be the perfect contributor to efficiently handle this project and complete it.

FUTURE WORK

By contributing to checkstyle, I’ve gained precious experience of how Java works in real world projects, gained beneficial skills and knowledge of Continuous Integration, Mutation Testing, Code Coverage, Open-Source and community work.

After GSoC I will continue my contribution to Checkstyle, I will make quality PRs, help newcomers and solve their doubts. I’ve gained important skills by contributing to Checkstyle and I wish to continue expanding my knowledge even further.

I’ve thought of some new features ideas to add in Checkstyle, I would love to discuss them with the maintainers and implement them and improve Checkstyle user experience.

THANK YOU