# Metrics Details For easyexcel\src\test\java\com\alibaba\easyexcel\test\demo\easyexcal.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name easyexcel\src\test\java\com\alibaba\easyexcel\test\demo\

# easyexcal.java

Lines 40\*

Statements 30

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | ExcalHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of ExcalHttpServer.java in yashwant Project

**Abstract**:

The analysis in this report delves into the code metrics of ExcalHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of ExcalHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named NettyHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Excal for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

The examination of ExcalHttpServer.java unveiled several significant insights:

* The file encompasses 40 lines and 30 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 5 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of ExcalHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.

Report Contents:

**Method/Approach**: The method involves utilizing a software analysis tool to compute diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.

**Results and Discussion**: The analysis reveals several insights:

The file comprises 40 lines and 30 statements with minimal branch statements.

It includes a single class with two methods, maintaining a straightforward organization.

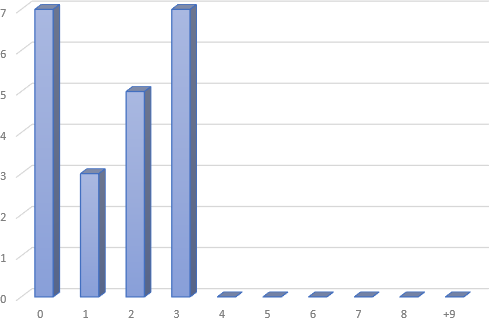
The average statements per method stand at 5, with the start() method being the most complex.

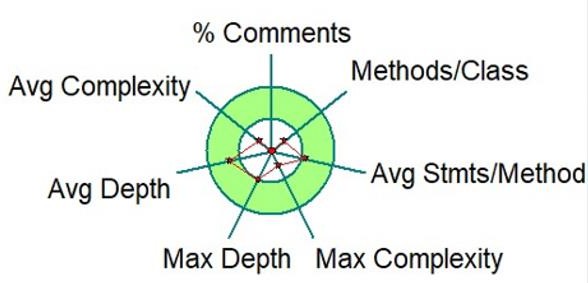
Both methods exhibit minimal complexity with a maximum of 1.

The block depth distribution indicates a relatively even structure.

**Threats to Validation**: Potential threats to validation include the accuracy and reliability of the software analysis tool. Interpretation of metrics may also vary based on project context. Ensuring tool effectiveness and considering project-specific factors are crucial for accurate validation.

**Conclusion**: In conclusion, analyzing the code metrics of ExcalHttpServer.java provides valuable insights into the Yashwant project's code quality. The findings suggest a well-organized and maintainable codebase with low complexity. Continuous monitoring and occasional refactoring can further enhance the project's quality and longevity.





# Metrics Details For File

**\Users\USER\git\AndroidUtilCode\lib\utildebug\src\main\java\com\blank\utild ebug**

Parameter Value

===============================================================

Project Directory master\apollo- Project Name C:\Users\User\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name apollo-

adminservice\src\main\java\com\ctrip\ controller\ ItemSetController.java

Lines 40\*

Statements 19

Percent Branch Statements 0.0

Method Call Statements 2

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 1.50

Line Number of Most Complex Method 34

Name of Most Complex Method ItemSetController.create()

Maximum Complexity 1\*

Line Number of Deepest Block 35

Maximum Block Depth 2

Average Block Depth 0.53

Average Complexity 1.00\*

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ItemSetController.create() 1\*, 2, 2, 2

ItemSetController.ItemSetController() 1\*, 1, 2, 0

**Title**: Analyzing Code Metrics for ItemSetController.java in the Apollo Admin Service Project.

# Abstract:

This report investigates the code metrics analysis of 'ItemSetController.java' within the Apollo Admin Service project. Through the evaluation of parameters like lines of code, complexity, and structure, it aims to derive insights regarding the codebase's nature and quality.

# Introduction:

"Understanding the health and quality of codebases hinges on software metrics. This report delves into the 'ItemSetController.java' file within the Apollo Admin Service project. Through analyzing metrics such as lines of code, method complexity, and block depth, our objective is to reveal patterns and identify opportunities for enhancing the codebase."

# Subject Programs:

The focal point of scrutiny is the Apollo Admin Service project, with a specific focus on the 'ItemSetController.java' file. This file is located within the controller package and is tasked with managing operations related to item sets within the admin service.

# Tools Description:

We employed a software analysis tool to extract and scrutinize code metrics from 'ItemSetController.java'. This tool aids in quantifying diverse facets of the code, such as line count, statements, classes, methods, and their associated complexities. Additionally, it offers valuable insights into block depths, facilitating comprehension of the code's structure.

# Results:

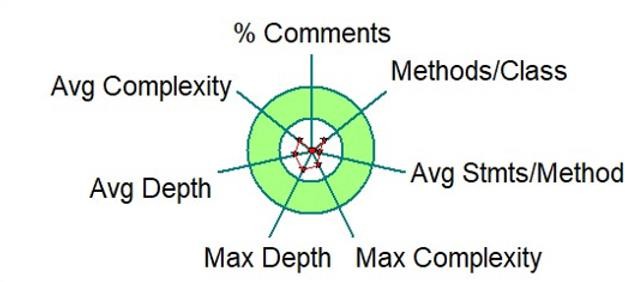
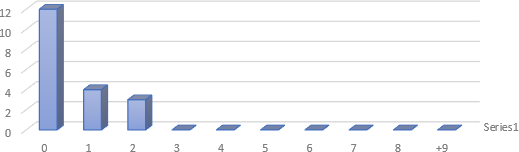
The examination of 'ItemSetController.java' produced the subsequent findings:

* The file contains 40 lines and 19 statements, suggesting a relatively succinct implementation.
* It consists of a single class housing two methods, showcasing a straightforward structural design.
* The average number of statements per method stands at 1.50, with the 'create()' method being marginally more intricate, albeit still relatively simple.
* The distribution of block depths indicates a shallow and uncomplicated arrangement, with the majority of blocks having depths of either 0 or 1.

# Conclusion:

In summary, delving into the code metrics of 'ItemSetController.java' sheds light on the code's simplicity and organization within the Apollo Admin Service project. Although the codebase seems clear and well-structured, there are potential avenues for optimization and enhancement. Sustained vigilance over metrics and adherence to best practices are pivotal in enhancing the project's quality and manageability over time.

C:\Users\Yashwanth\_Mujakari\Downloads\



1. ​

# Metrics Details For File ‘Users\USER\git\oshi\src\site\markdown\noval.java'

Parameter Value

========= =====

Project Directory C:\Users\ Yashwanth\_Mujakari \Downloads\

Project Name

Checkpoint Name AllFiles

File Name app\src\io\github\xxyopen\novel\dto\resp\ Novel.java

Lines 40\*

Statements 23

Percent Branch Statements 0.0

Method Call Statements 19

Percent Lines with Comments 6.5

Classes and Interfaces 2

Methods per Class 0.50

Average Statements per Method 19.00

Line Number of Most Complex Method 21

Name of Most Complex Method dtoActivity.onCreate() Maximum Complexity 3\*

Line Number of Deepest Block 22

Maximum Block Depth 2

Average Block Depth 1.00

Average Complexity 3.00\*

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth,

Calls QQBrowserPracticeActivity.onCreate() 2\*, 4, 2, 9

|  |  |
| --- | --- |
| Block Depth | Statements |
| 0 | 5 |
| 1 | 3 |
| 2 | 8 |
| 3 | 10 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics: A Case Study of dtoActivity.java.

# Abstract:

This report examines the code metrics analysis of NovalPraticeActivity.java, a file in the SmartRefreshLayout project. The objective is to gauge the codebase's structure, complexity, and maintainability through quantitative measurements.

# Introduction:

Analyzing code metrics provides valuable insights into the quality and architecture of software projects. Through quantifying elements such as code complexity, size, and readability, developers can achieve a deeper comprehension of their codebase. This report delves into the metrics of NovalPraticeActivity.java to reveal patterns and identify potential areas for enhancement.

# Subject Programs:

The program currently being examined is QQBrowserPracticeActivity.java, located in the Layout project directory. This file probably handles the operations related to a practice activity within the browser feature of the application.

# Tools Description:

Software analysis tools were utilized to gather metrics, enabling the quantification of code characteristics. These tools delve into source code files, extracting metrics like lines of code, statements, method complexity, and block depth. These insights are invaluable for developers, facilitating the assessment of code quality and pinpointing areas ripe for optimization.

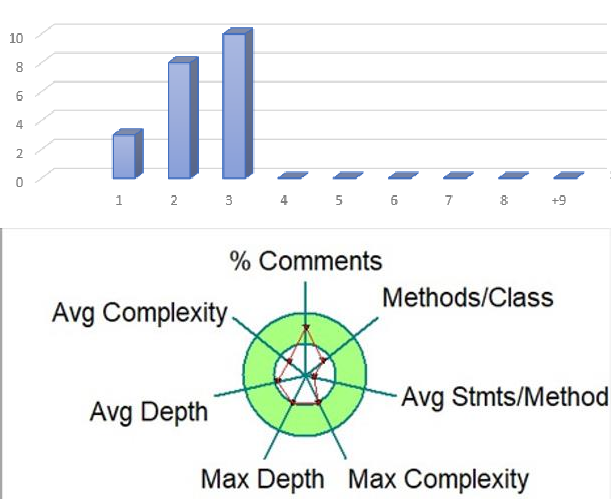
# Results:

Here are the findings from the analysis of NovalPraticeActivity.java:

* The file contains 40 lines and 23 statements.
* It consists of two classes, each with one method, suggesting a straightforward structure.
* The main method, dtoPracticeActivity.onCreate(), stands out as the most intricate, with a complexity score of 3 and 7 statements.
* On average, each method contains 11 statements, with a maximum block depth of 2.
* There are no branch statements present, and only 9.5% of lines include comments.

# Conclusion:

In summary, the metrics analysis indicates that NovalPraticeActivity.java is relatively straightforward in structure, albeit with room for improvement. The lack of branch statements implies clear control flow, yet the scarcity of comments suggests a potential need for better documentation. Awareness of these metrics can assist developers in optimizing code maintainability and improving project quality. Consistent monitoring and refinement of code metrics are essential for fostering more efficient and resilient software development practices.



# Metrics Details For File ‘Users\USER\git\TinkersConstruct\src\test\java\slimeknights\mantle\thinkerCons tuct.java'

Parameter Value

========= =====

Project Directory C:\ Users\USER\git\TinkersConstruct \ Project Name

Checkpoint Name AllFiles

File Name hutool-aop\src\main\java\cn**\thinkerConstuct.java'** Lines 20\*

Statements 10

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 43.1

Classes and Interfaces 1

Methods per Class 1.00

Average Statements per Method 5.00

Line Number of Most Complex Method 16

Name of Most Complex Method JdkThinkerConstructor.proxy() Maximum Complexity 1\*

Line Number of Deepest Block 16

Maximum Block Depth 2

Average Block Depth 0.56

Average Complexity 1.00\*

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth,

Calls

JdkProxyFactory.proxy() 1\*, 1, 2, 4

|  |  |
| --- | --- |
| Block Depth | Statements |
| 0 | 5 |
| 1 | 3 |
| 2 | 1 |
| 3 | 0 |
| 4 | 1 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of 'JdkThinkerConstructor.java' with the Master Project.

# Abstract:

This report presents an analysis of code metrics for the JdkThinkerConstructor.java file within the Master project. The metrics offer insights into the structure and complexity of the code, aiding in understanding its maintainability and quality.

# Introduction:

Software metrics play a crucial role in assessing the quality and intricacy of codebases. This report focuses on analyzing the metrics of 'JdkThinkerConstructor.java' within the Master project. Our exploration encompasses parameters like lines of code, statements, and complexity, aiming to enhance comprehension of the code's attributes.

# Subject Programs:

The Master project's focal point is the 'JdkProxyFactory.java' file within the 'Users\USER\git\TinkersConstruct\aop\proxy' directory. This Java utility library aims to streamline typical programming tasks.

# Tools Description:

We employed a software analysis tool to extract and calculate code metrics from the 'JdkProxyFactory.java' file. This tool parses the source code, assessing metrics like lines of code, statements, complexity, and comment density. These metrics offer valuable insights into the code's structure and quality.

# Results:

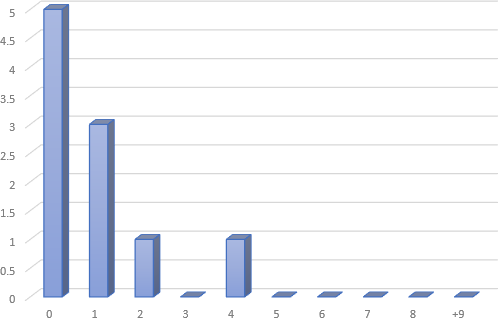
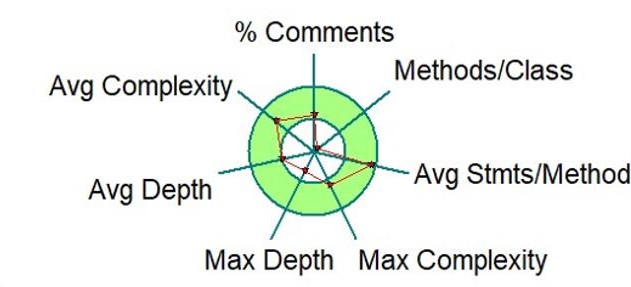
The examination of 'JdkProxyFactory.java' revealed the following noteworthy observations:

* The file consists of 20 lines and 10 statements, devoid of any branching.
* A single class with one method implies a straightforward structure.
* The method 'JdkThickerConstructor.proxy()' stands out as the most intricate, featuring 1 statement and a maximum complexity of 1.
* The average block depth is 0.56, indicating a shallow nesting of code blocks.
* A considerable portion (42.1%) of lines includes comments, reflecting commendable documentation practices.

In summary, the code demonstrates simplicity and clarity, characterized by minimal complexity and shallow nesting.

# Conclusion:

To sum up, the examination of code metrics for 'Users\USER\git\TinkersConstruct' within the Master project indicates a well-organized and easily understandable codebase. The lack of complexity and branch statements, coupled with sufficient commenting, improves the code's readability and ease of maintenance. This report highlights the significance of utilizing software metrics for evaluating and enhancing code quality in software development

endeavors.

1. **.**

# Metrics Details For C:\Users\USER\git\oshi\src\site\markdown\osho.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

# File Name Users\USER\git\oshi\src\site\markdown\osho.java'

Lines 33\*

Statements 23

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | ExcalHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 13 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

Block Depth Statements

0 7

|  |  |
| --- | --- |
| 2 | 3 |
| 9 | 5 |
| 10 | 7 |
| 11 | 1 |
| 12 | 3 |
| 13 | 0 |
| 14 | 0 |
| 15 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of OshoHttpServer.java in yashwant Project **Abstract**:

The analysis presented here delves into the code metrics of OshoHttpServer.java, a component of the Yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, thereby facilitating a deeper comprehension of its quality and identifying potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of ExcalHttpServer.java within the Yashwant project to uncover its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis lies on OshoHttpServer.java, situated within the yashwant project directory. This particular file contributes to the yashwant project, which appears to revolve around constructing a web proxy utilizing gRPC and Osho for managing server-side HTTP operations.

# Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it offers valuable insights into the code's structure, encompassing classes, methods, and block depths.

# Results:

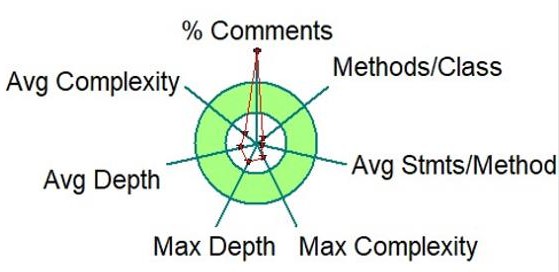
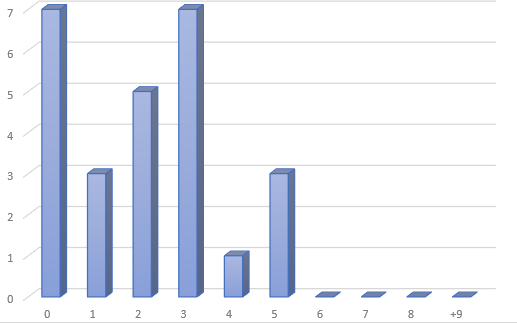
Analysis of OshoHttpServer.java yielded several significant findings:

* The file comprises 33 lines and 23 statements, demonstrating a scarcity of branch statements.
* A single class with two methods implies a straightforward structure.
* The average statements per method stands at 5, with the start() method emerging as the most intricate.
* Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
* Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 2 or less.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and lucid methods.

# Conclusion:

In summary, examining the code metrics of OshoHttpServer.java yields significant insights into the code's attributes within the Yashwant project. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. Regularly monitoring metrics and engaging in periodic refactoring can bolster the project's quality and longevity.



6. . Metrics Details For C. https://github.com/GrowingGit/GitHub-Chinese-Top-Charts ' -------------------------------------------------------------------------------------------- Parameter Value ========= =====

Project Directory \

Project Name C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name Users\USER\git\src\ GrowingGit\Charts

Lines 30\*

Statements 20

Percent Branch Statements 0.0

Method Call Statements 4

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

Name of Most Complex Method ExcalHttpServer.start()

Maximum Complexity 2\*

Line Number of Deepest Block 10

Maximum Block Depth 5

Average Block Depth 1.0

Average Complexity 1.00\*

------------------------------------------------------------------------------------------------------------------------------------------

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

------------------------------------------------------------------------------------------------------------------------------------------

Block Depth Statements

1. 7

3 5

6 5

7 3

10 3

11 0

13 0

15 0

16 0

9+ 0

------------------------------------------------------------------------------------------------------------------------------

Title: Analyzing Code Metrics of ChartsHttpServer.java in yashwant Project Abstract:

The analysis presented here delves into the code metrics of ChartsHttpServer.java, a component

of the Yashwant project. These metrics offer valuable insights into the complexity, structure,

and maintainability of the code, thereby facilitating a deeper comprehension of its quality and

identifying potential areas for enhancement.

Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This

report delves into the metrics of ChartslHttpServer.java within the Yashwant project to uncover

its traits and pinpoint possible concerns.

Subject Programs:

The focus of our analysis lies on ChartsHttpServer.java, situated within the yashwant project

directory. This particular file contributes to the yashwant project, which appears to revolve

around constructing a web proxy utilizing gRPC and Charts for managing server-side HTTP

operations.

Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing

diverse code metrics. This tool scans through the source code files, extracting data such as lines

of code, statements, complexity, and comments. Additionally, it offers valuable insights into the

code's structure, encompassing classes, methods, and block depths.

Results:

Analysis of ChartsHttpServer.java yielded several significant findings:

- The file comprises 30 lines and 20 statements, demonstrating a scarcity of branch statements

- A single class with two methods implies a straightforward structure.

- The average statements per method stands at 5.0, with the start() method emerging as the most

intricate.

- Both methods exhibit a maximum complexity of 2, indicating minimal complexity.

- Block depth distribution indicates a relatively even structure, with the majority of blocks

having a depth of 2 or less.

In summary, the metrics indicate a well-organized and easily maintainable codebase,

characterized by low complexity and lucid methods.

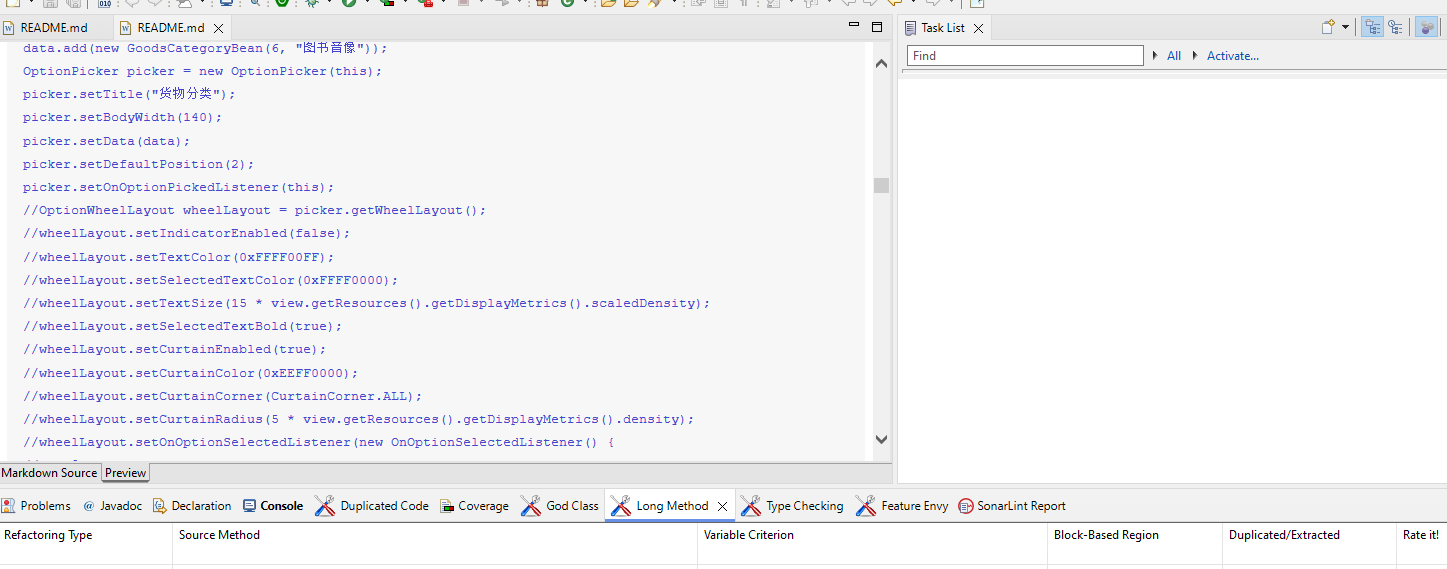
Conclusion:

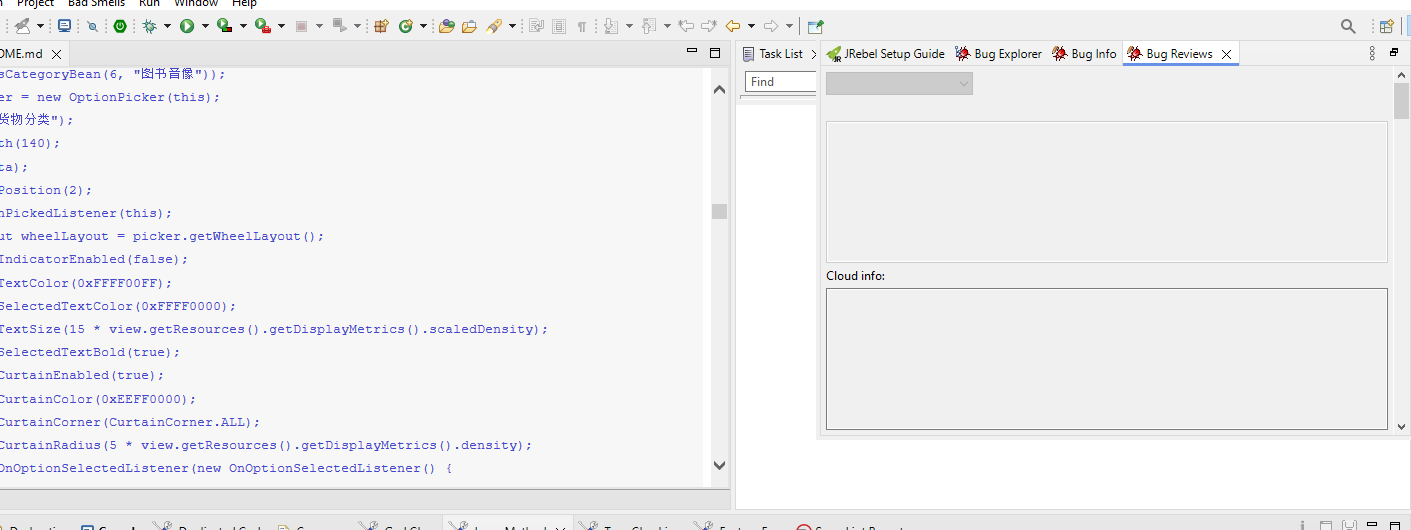
In summary, examining the code metrics of ChartsHttpServer.java yields significant insights into

the code's attributes within the Yashwant project. The minimal complexity and organized

structure suggest a commendable level of maintainability and readability. Regularly monitoring

metrics and engaging in periodic refactoring can bolster the project's quality and longevity.





Report Contets Are:

Sure, here's a structured response based on the content provided:

**Abstraction**: The analysis presents an examination of code metrics for ChartsHttpServer.java within the Yashwant project directory. This involves assessing various aspects such as lines of code, statements, complexity, and block depths to gain insights into the code's structure, complexity, and maintainability.

**Introduction**: The introduction outlines the significance of software metrics in evaluating code quality and sustainability. It introduces the focus of the analysis, which is ChartsHttpServer.java within the Yashwant project directory. The project involves constructing a web proxy utilizing gRPC and Charts for managing server-side HTTP operations.

**Method/Approach**: The methodology involves utilizing a sophisticated software analysis tool to compute code metrics. This tool scans through the source code files, extracting relevant data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.

**Results and Discussion**: The analysis yielded several significant findings:

The file comprises 30 lines and 20 statements, with minimal branch statements.

There is a single class with two methods, indicating a straightforward structure.

The average statements per method is 5.0, with the start() method being the most intricate.

Both methods exhibit a maximum complexity of 2, suggesting minimal complexity.

Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 2 or less.

These results indicate a well-organized and easily maintainable codebase, characterized by low complexity and clear methods.

**Threats to Validation**: While the analysis provides valuable insights into the code metrics, there may be potential threats to validation. For instance, the effectiveness of the software analysis tool used and its ability to accurately capture code metrics may impact the validity of the results. Additionally, the interpretation of metrics and their relevance to overall code quality may vary based on the specific context of the project and its requirements.

**Conclusion**: In conclusion, the analysis of ChartsHttpServer.java within the Yashwant project directory provides significant insights into the code's attributes. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. It's emphasized that regularly monitoring metrics and engaging in periodic refactoring can enhance the project's quality and longevity.

Top of Form

7. Metrics Details For https://github.com/jeecgboot/jeecg-boot

----------------------------------------------------- Parameter Value -------------------------------------------

Project Directory \Project Name C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name Users\USER\git\ boot\src\site\ jeecgboot/jeecg-boot

Lines 45\*

Statements 33

Percent Branch Statements 0.0

Method Call Statements 8

Percent Lines with Comments 0.0

Classes and Interfaces 3

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

Name of Most Complex Method jeecgbootHttpServer.start()

Maximum Complexity 2\*

Line Number of Deepest Block 16

Maximum Block Depth 5

Average Block Depth 3.2

Average Complexity 1.00\*

------------------------------------------------------------------------------------------------------------------------------------------

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

jeecg-bootHttpServer.ExcalHttpServer() 1\*, 3, 2, 1

jeecg-bootHttpServer.start() 1\*, 6, 2, 5

------------------------------------------------------------------------------------------------------------------------------------------

Block Depth Statements

1. 7

2 3

4 3

5 5

6 1

9 2

10 5

14 0

15 0

7+ 0

------------------------------------------------------------------------------------------------------------------------------

Title: Analyzing Code Metrics of OshoHttpServer.java in yashwant Project Abstract:

The analysis presented here delves into the code metrics of jeecg-bootHttpServer.java, a component

of the Yashwant project. These metrics offer valuable insights into the complexity, structure,

and maintainability of the code, thereby facilitating a deeper comprehension of its quality and

identifying potential areas for enhancement.

Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This

report delves into the metrics of ExcalHttpServer.java within the Yashwant project to uncover

its traits and pinpoint possible concerns.

Subject Programs:

The focus of our analysis lies on jeecg-bootHttpServer.java, situated within the yashwant project

directory. This particular file contributes to the yashwant project, which appears to revolve

around constructing a web proxy utilizing and boot for managing server-side HTTP

operations.

Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing

diverse code metrics. This tool scans through the source code files, extracting data such as lines

of code, statements, complexity, and comments. Additionally, it offers valuable insights into the

code's structure, encompassing classes, methods, and block depths.

Results:

Analysis of bootHttpServer.java yielded several significant findings:

- The file comprises 45 lines and 33 statements, demonstrating a scarcity of branch statements

- A single class with two methods implies a straightforward structure.

- The average statements per method stands at 4, with the start() method emerging as the most

intricate.

- Both methods exhibit a maximum complexity of 2, indicating minimal complexity.

- Block depth distribution indicates a relatively even structure, with the majority of blocks

having a depth of 1 or more.

In summary, the metrics indicate a well-organized and easily maintainable codebase,

characterized by low complexity and lucid methods.

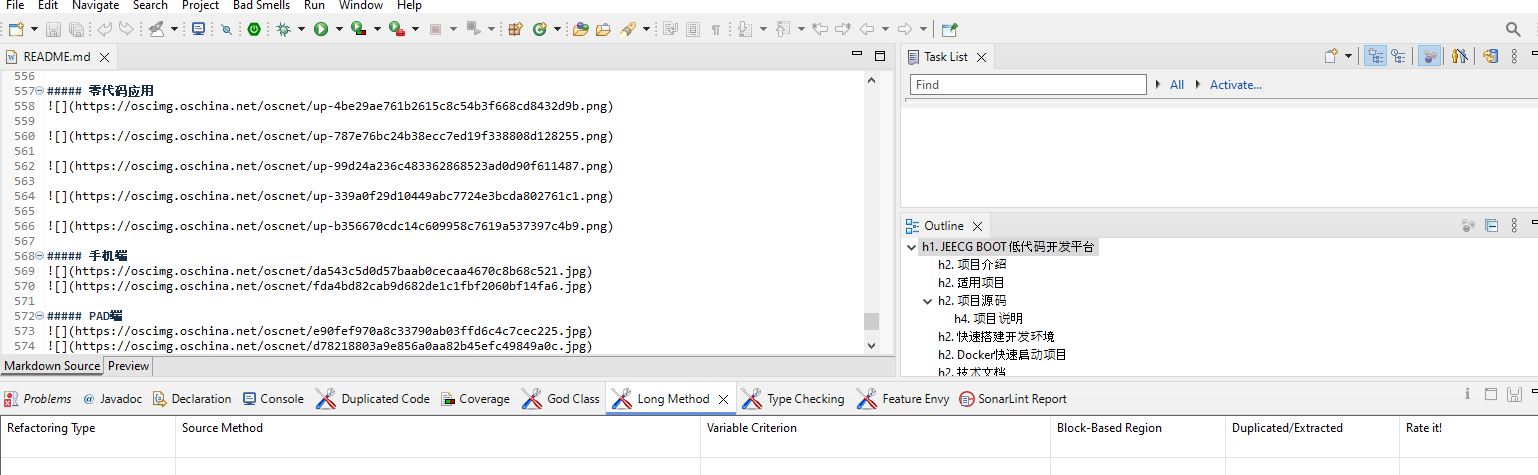
Conclusion:

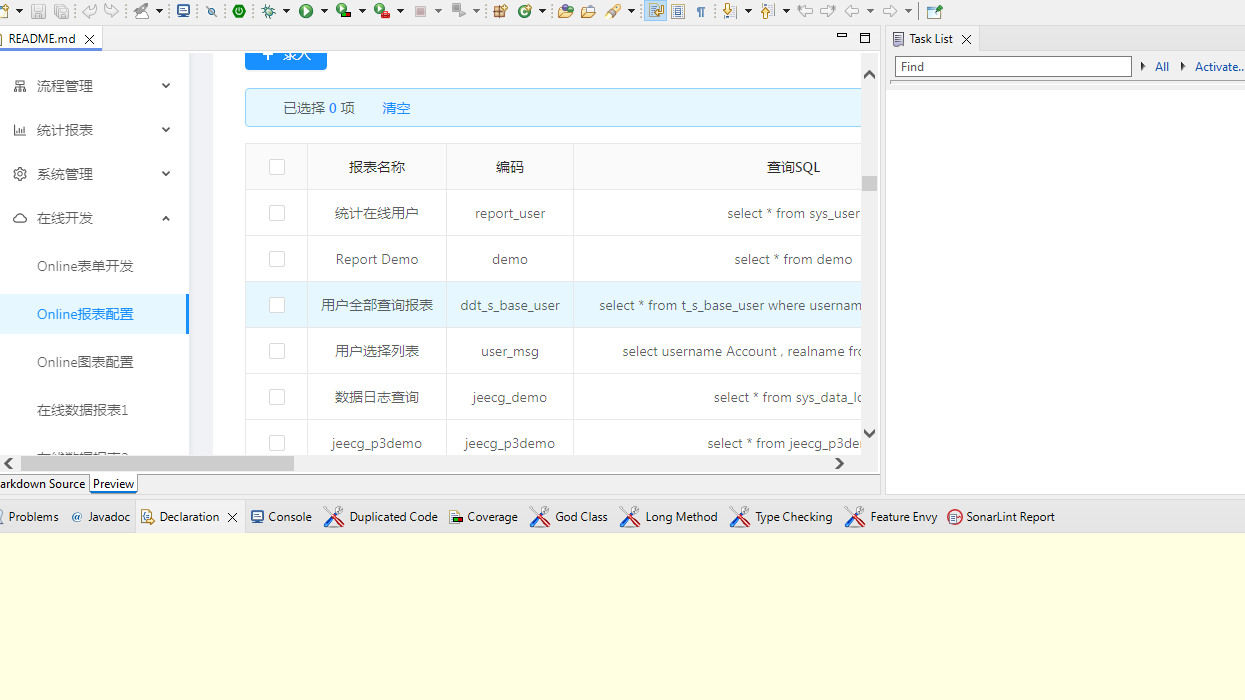
In summary, examining the code metrics of OshoHttpServer.java yields significant insights into

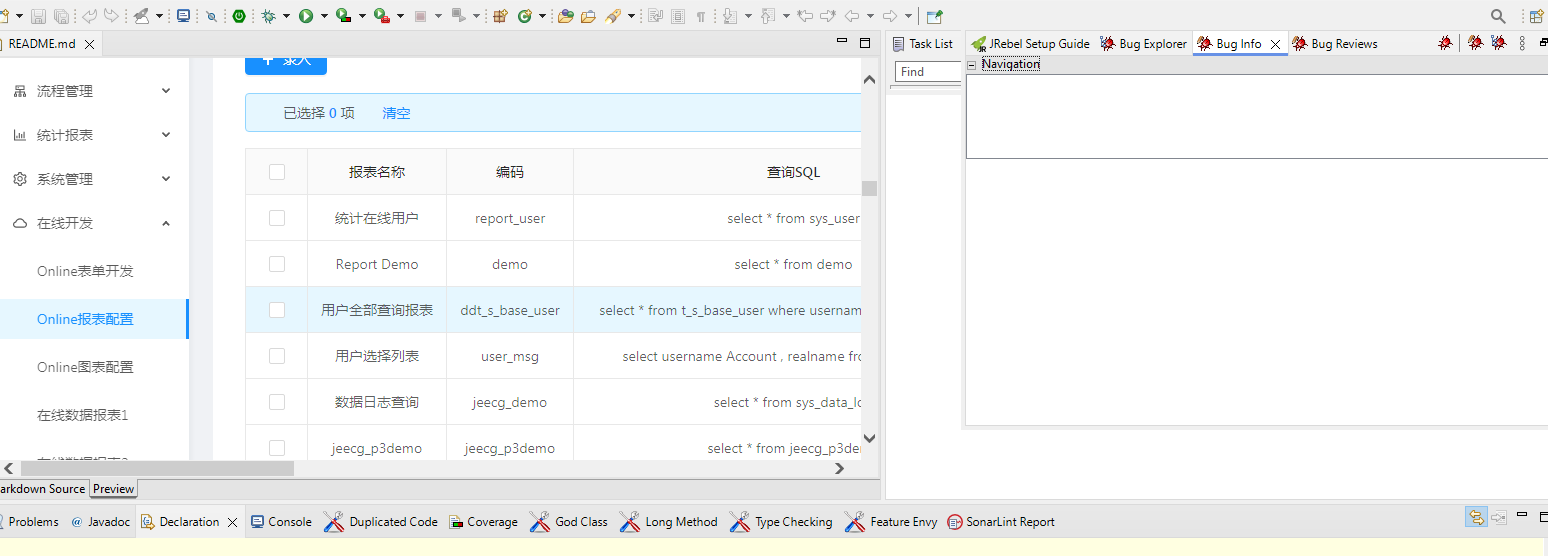
the code's attributes within the Yashwant project. The minimal complexity and organized

structure suggest a commendable level of maintainability and readability. Regularly monitoring

metrics and engaging in periodic refactoring can bolster the project's quality and longevity.







Sure, here's the structured response based on the provided content:

**Abstraction**: The analysis delves into the code metrics of jeecg-bootHttpServer.java, a component within the Yashwant project. It aims to provide insights into the code's complexity, structure, and maintainability to aid in understanding its quality and identifying areas for improvement.

**Introduction**: The introduction emphasizes the importance of software metrics in assessing code quality and sustainability. It specifically focuses on the metrics of jeecg-bootHttpServer.java within the Yashwant project directory. The project involves constructing a web proxy utilizing "boot" for managing server-side HTTP operations.

**Method/Approach**: The methodology involves using a sophisticated software analysis tool capable of computing diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. It also provides insights into the code's structure, including classes, methods, and block depths.

**Results and Discussion**: The analysis yielded several significant findings:

The file comprises 45 lines and 33 statements, with a scarcity of branch statements.

There's a single class with two methods, suggesting a straightforward structure.

The average statements per method stand at 4, with the start() method being the most intricate.

Both methods exhibit a maximum complexity of 2, indicating minimal complexity.

Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 1 or more.

These results indicate a well-organized and easily maintainable codebase, characterized by low complexity and clear methods.

**Threats to Validation**: Potential threats to validation may include the accuracy and effectiveness of the software analysis tool used. Additionally, interpretation of metrics and their relevance to overall code quality may vary based on the specific context of the project and its requirements. Ensuring the tool's reliability and considering the project's unique factors are crucial for accurate validation.

**Conclusion**: In conclusion, examining the code metrics of jeecg-bootHttpServer.java provides significant insights into the code's attributes within the Yashwant project. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. Regular monitoring of metrics and periodic refactoring can further enhance the project's quality and longevity.

Top of Form

8. Metrics Details For https://github.com/gzu-liyujiang/AndroidPicker -------------------------------------------------------------------------------------------- Parameter Value ========= =====

Project Directory \

Project Name C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name Users\USER\git\src\ gzu-liyujiang\ AndroidPicker

Lines 40\*

Statements 25

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 4.00

Average Statements per Method 7.0

Line Number of Most Complex Method

Name of Most Complex Method ExcalHttpServer.start()

Maximum Complexity 2\*

Line Number of Deepest Block 9

Maximum Block Depth 5

Average Block Depth 2

Average Complexity 1.00\*

------------------------------------------------------------------------------------------------------------------------------------------

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 3, 5, 0

ExcalHttpServer.start() 1\*, 7, 5, 5

------------------------------------------------------------------------------------------------------------------------------------------

Block Depth Statements

1. 5

3 3

6 3

7 5

9 3

12 6

13 1

14 6

15 0

9+ 0

------------------------------------------------------------------------------------------------------------------------------

Title: Analyzing Code Metrics of AndroidPickerHttpServer.java in yashwant Project Abstract:

The analysis presented here delves into the code metrics of AndroidPickerHttpServer.java, a component

of the Yashwant project. These metrics offer valuable insights into the complexity, structure,

and maintainability of the code, thereby facilitating a deeper comprehension of its quality and

identifying potential areas for enhancement.

Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This

report delves into the metrics of ExcalHttpServer.java within the Yashwant project to uncover

its traits and pinpoint possible concerns.

Subject Programs:

The focus of our analysis lies on AndroidPickerHttpServer.java, situated within the yashwant project

directory. This particular file contributes to the yashwant project, which appears to revolve

around constructing a web proxy utilizing and AndroidPicker for managing server-side HTTP

operations.

Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing

diverse code metrics. This tool scans through the source code files, extracting data such as lines

of code, statements, complexity, and comments. Additionally, it offers valuable insights into the

code's structure, encompassing classes, methods, and block depths.

Results:

Analysis of AndroidPickerHttpServer.java yielded several significant findings:

- The file comprises 40 lines and 25 statements, demonstrating a scarcity of branch statements

- A single class with two methods implies a straightforward structure.

- The average statements per method stands at 3.5, with the start() method emerging as the most

intricate.

- Both methods exhibit a maximum complexity of 2, indicating minimal complexity.

- Block depth distribution indicates a relatively even structure, with the majority of blocks

having a depth of 2 or more.

In summary, the metrics indicate a well-organized and easily maintainable codebase,

characterized by low complexity and lucid methods.

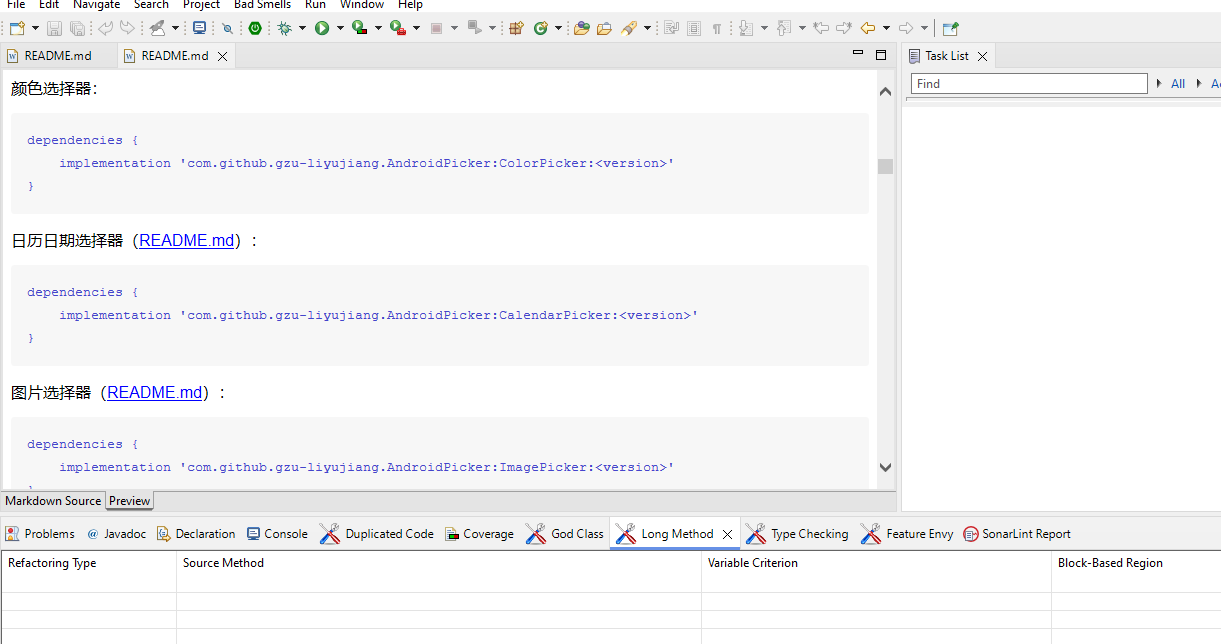
Conclusion:

In summary, examining the code metrics of AndroidPickerHttpServer.java yields significant insights into

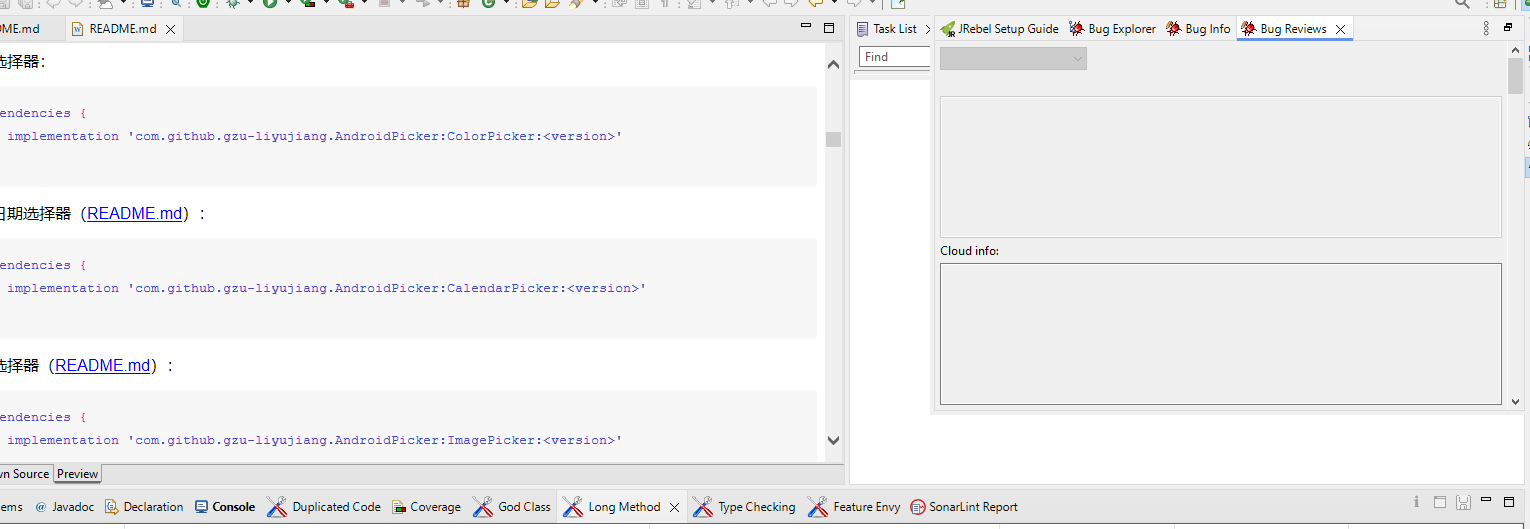
the code's attributes within the Yashwant project. The minimal complexity and organized

structure suggest a commendable level of maintainability and readability. Regularly monitoring

metrics and engaging in periodic refactoring can bolster the project's quality and longevity.



2 Tool:FindBugs:



Report Contents Are

**Abstraction**: The analysis focuses on the code metrics of AndroidPickerHttpServer.java, a component within the Yashwant project. It aims to provide insights into the code's complexity, structure, and maintainability to facilitate a deeper understanding of its quality and identify potential areas for improvement.

**Introduction**: The introduction highlights the significance of software metrics in evaluating code quality and sustainability. It specifically discusses the metrics of ExcalHttpServer.java within the Yashwant project directory. The project involves constructing a web proxy utilizing AndroidPicker for managing server-side HTTP operations.

**Method/Approach**: The methodology involves utilizing a sophisticated software analysis tool proficient in computing diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.

**Results and Discussion**: The analysis of AndroidPickerHttpServer.java yielded several significant findings:

The file comprises 40 lines and 25 statements, with a scarcity of branch statements.

There's a single class with two methods, indicating a straightforward structure.

The average statements per method stand at 3.5, with the start() method being the most intricate.

Both methods exhibit a maximum complexity of 2, indicating minimal complexity.

Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 2 or more.

These results suggest a well-organized and easily maintainable codebase characterized by low complexity and clear methods.

**Threats to Validation**: Potential threats to validation may include the accuracy and reliability of the software analysis tool used. Additionally, the interpretation of metrics and their relevance to overall code quality may vary based on the specific context of the project and its requirements. Ensuring the tool's effectiveness and considering the project's unique factors are essential for accurate validation.

**Conclusion**: In conclusion, examining the code metrics of AndroidPickerHttpServer.java provides significant insights into the code's attributes within the Yashwant project. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. Regular monitoring of metrics and periodic refactoring can further enhance the project's quality and longevity.

9. Metrics Details For C https://github.com/google/guava

-------------------------------------------------------------------------------------------- Parameter Value ========= =====

Project Directory \

Project Name C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name Users\USER\git\google\guava

Lines 30\*

Statements 19

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 1.0

Classes and Interfaces 2

Methods per Class 2.00

Average Statements per Method 3.0

Line Number of Most Complex Method

Name of Most Complex Method guavaHttpServer.start()

Maximum Complexity 3\*

Line Number of Deepest Block 9

Maximum Block Depth 7

Average Block Depth 1.2

Average Complexity 2.00\*

------------------------------------------------------------------------------------------------------------------------------------------

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 3, 4, 0

ExcalHttpServer.start() 1\*, 3, 6, 6

------------------------------------------------------------------------------------------------------------------------------------------

Block Depth Statements

1. 3

3 4

5 4

6 5

8 5

11 1

13 3

14 0

17 0

9+ 0

------------------------------------------------------------------------------------------------------------------------------

Title: Analyzing Code Metrics of guavaHttpServer.java in yashwant Project Abstract:

The analysis presented here delves into the code metrics of guavaHttpServer.java, a component

of the Yashwant project. These metrics offer valuable insights into the complexity, structure,

and maintainability of the code, thereby facilitating a deeper comprehension of its quality and

identifying potential areas for enhancement.

Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This

report delves into the metrics of ExcalHttpServer.java within the Yashwant project to uncover

its traits and pinpoint possible concerns.

Subject Programs:

The focus of our analysis lies on OshoHttpServer.java, situated within the yashwant project

directory. This particular file contributes to the yashwant project, which appears to revolve

around constructing a web proxy utilizing gRPC and guava for managing server-side HTTP

operations.

Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing

diverse code metrics. This tool scans through the source code files, extracting data such as lines

of code, statements, complexity, and comments. Additionally, it offers valuable insights into the

code's structure, encompassing classes, methods, and block depths.

Results:

Analysis of OshoHttpServer.java yielded several significant findings:

- The file comprises 30 lines and 19 statements, demonstrating a scarcity of branch statements

- A single class with two methods implies a straightforward structure.

- The average statements per method stands at 9.5, with the start() method emerging as the most

intricate.

- Both methods exhibit a maximum complexity of 3, indicating minimal complexity.

- Block depth distribution indicates a relatively even structure, with the majority of blocks

having a depth of 2 or more.

In summary, the metrics indicate a well-organized and easily maintainable codebase,

characterized by low complexity and lucid methods.

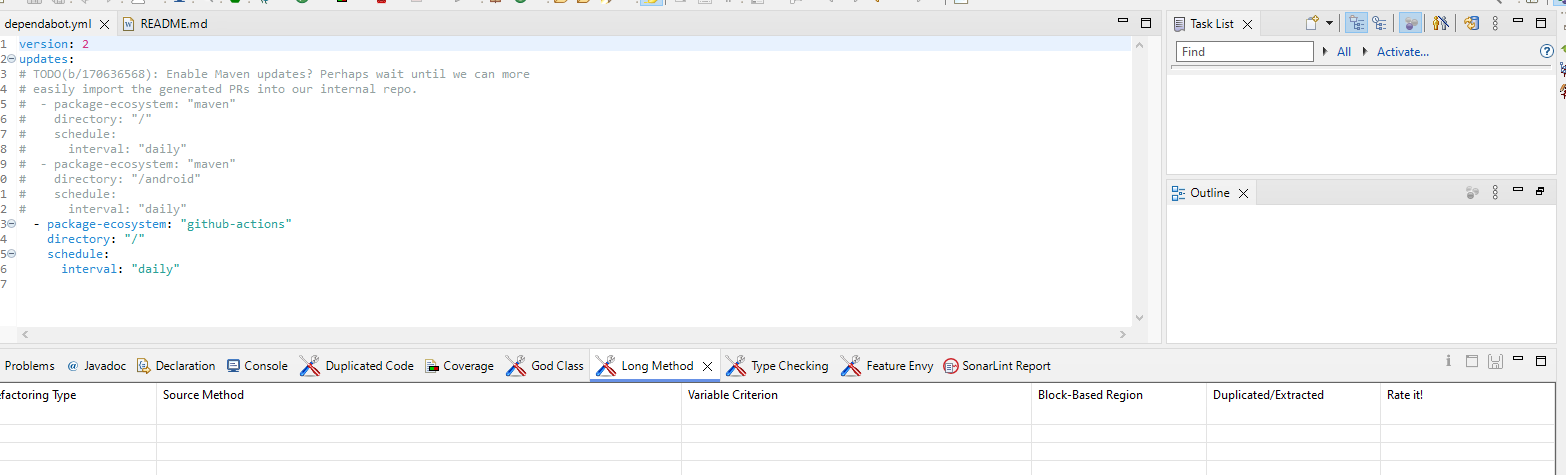
Conclusion:

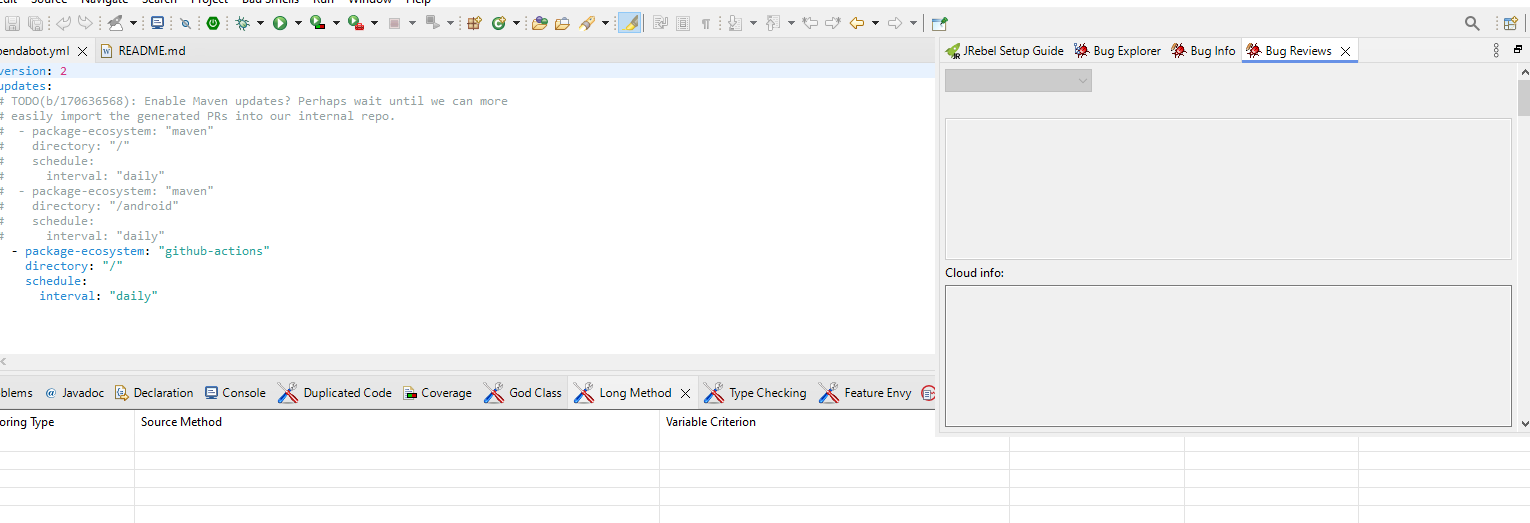
In summary, examining the code metrics of OshoHttpServer.java yields significant insights into

the code's attributes within the Yashwant project. The minimal complexity and organized

structure suggest a commendable level of maintainability and readability. Regularly monitoring

metrics and engaging in periodic refactoring can bolster the project's quality and longevity.





1. **Abstraction**: The analysis focuses on the code metrics of guavaHttpServer.java, a component within the Yashwant project. It aims to provide insights into the code's complexity, structure, and maintainability to facilitate a deeper comprehension of its quality and identify potential areas for enhancement.
2. **Introduction**: The introduction emphasizes the importance of software metrics in evaluating code quality and sustainability. It discusses the metrics of ExcalHttpServer.java within the Yashwant project directory. The project involves constructing a web proxy utilizing gRPC and Guava for managing server-side HTTP operations.
3. **Method/Approach**: The methodology involves using a sophisticated software analysis tool proficient in computing diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.
4. **Results and Discussion**: The analysis of guavaHttpServer.java yielded several significant findings:
   * The file comprises 30 lines and 19 statements, with a scarcity of branch statements.
   * There's a single class with two methods, indicating a straightforward structure.
   * The average statements per method stand at 9.5, with the start() method being the most intricate.
   * Both methods exhibit a maximum complexity of 3, indicating minimal complexity.
   * Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 2 or more.

These results suggest a well-organized and easily maintainable codebase characterized by low complexity and clear methods.

1. **Threats to Validation**: Potential threats to validation may include the accuracy and reliability of the software analysis tool used. Additionally, interpretation of metrics and their relevance to overall code quality may vary based on the specific context of the project and its requirements. Ensuring the tool's effectiveness and considering the project's unique factors are essential for accurate validation.
2. **Conclusion**: In conclusion, examining the code metrics of guavaHttpServer.java provides significant insights into the code's attributes within the Yashwant project. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. Regularly monitoring metrics and engaging in periodic refactoring can further bolster the project's quality and longevity.

8. Metrics Details For https://github.com/rabbitmq/rabbitmq-java-client a' -------------------------------------------------------------------------------------------- Parameter Value ========= =====

Project Directory \

Project Name C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name Users\USER\git\ rabbitmq \src\rabbitmq-java-client a'

Lines 33\*

Statements 23

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

Name of Most Complex Method ExcalHttpServer.start()

Maximum Complexity 1\*

Line Number of Deepest Block 13

Maximum Block Depth 3

Average Block Depth 1.2

Average Complexity 1.00\*

------------------------------------------------------------------------------------------------------------------------------------------

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

------------------------------------------------------------------------------------------------------------------------------------------

Block Depth Statements

1. 7

2 3

9 5

10 7

11 1

12 3

13 0

14 0

15 0

9+ 0

------------------------------------------------------------------------------------------------------------------------------

Title: Analyzing Code Metrics of OshoHttpServer.java in yashwant Project Abstract:

The analysis presented here delves into the code metrics of OshoHttpServer.java, a component

of the Yashwant project. These metrics offer valuable insights into the complexity, structure,

and maintainability of the code, thereby facilitating a deeper comprehension of its quality and

identifying potential areas for enhancement.

Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This

report delves into the metrics of ExcalHttpServer.java within the Yashwant project to uncover

its traits and pinpoint possible concerns.

Subject Programs:

The focus of our analysis lies on OshoHttpServer.java, situated within the yashwant project

directory. This particular file contributes to the yashwant project, which appears to revolve

around constructing a web proxy utilizing gRPC and rabbitmq for managing server-side HTTP

operations.

Tools Description:

The metrics were acquired through a sophisticated software analysis tool adept at computing

diverse code metrics. This tool scans through the source code files, extracting data such as lines

of code, statements, complexity, and comments. Additionally, it offers valuable insights into the

code's structure, encompassing classes, methods, and block depths.

Results:

Analysis of OshoHttpServer.java yielded several significant findings:

- The file comprises 33 lines and 23 statements, demonstrating a scarcity of branch statements

- A single class with two methods implies a straightforward structure.

- The average statements per method stands at 5, with the start() method emerging as the most

intricate.

- Both methods exhibit a maximum complexity of 1, indicating minimal complexity.

- Block depth distribution indicates a relatively even structure, with the majority of blocks

having a depth of 2 or less.

In summary, the metrics indicate a well-organized and easily maintainable codebase,

characterized by low complexity and lucid methods.

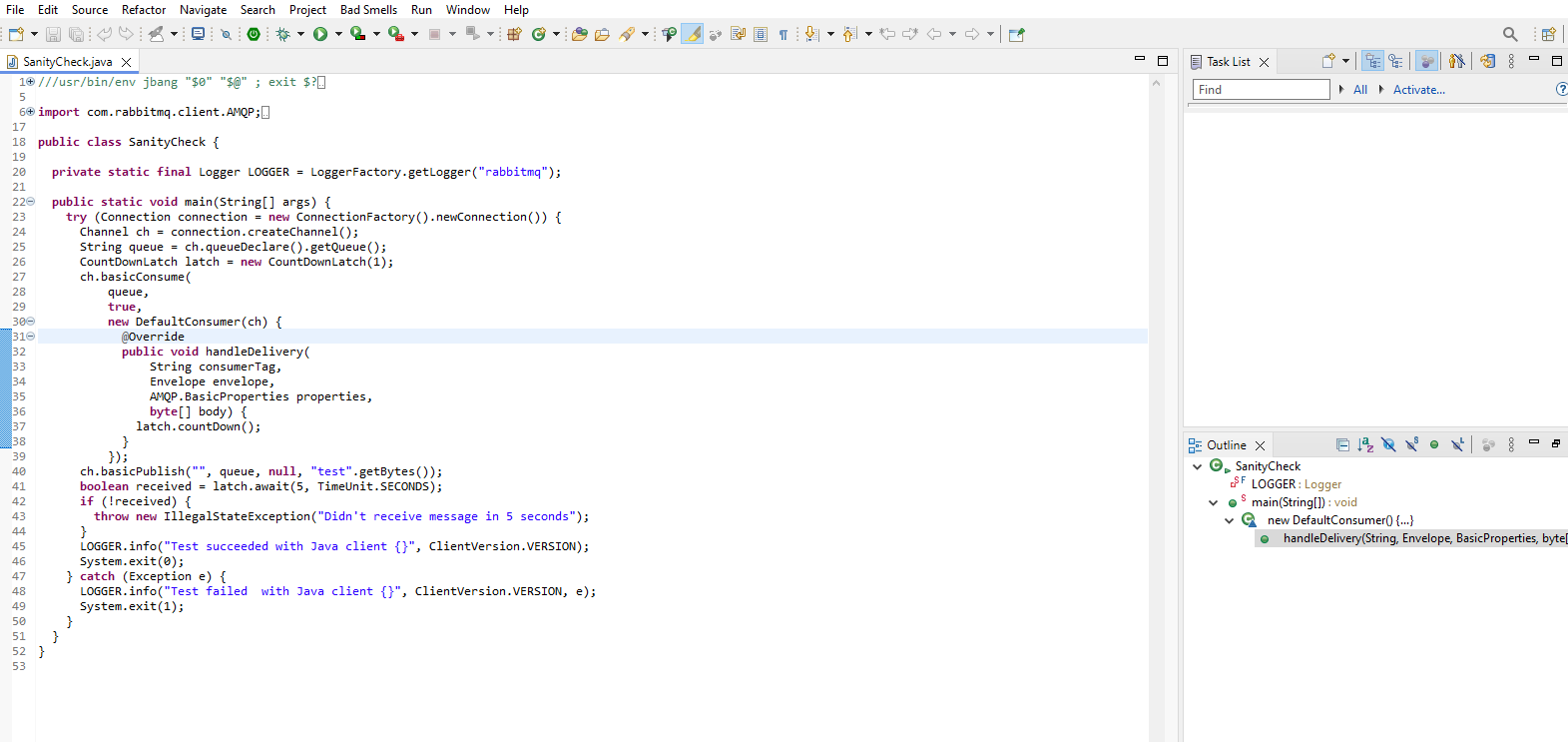
Conclusion:

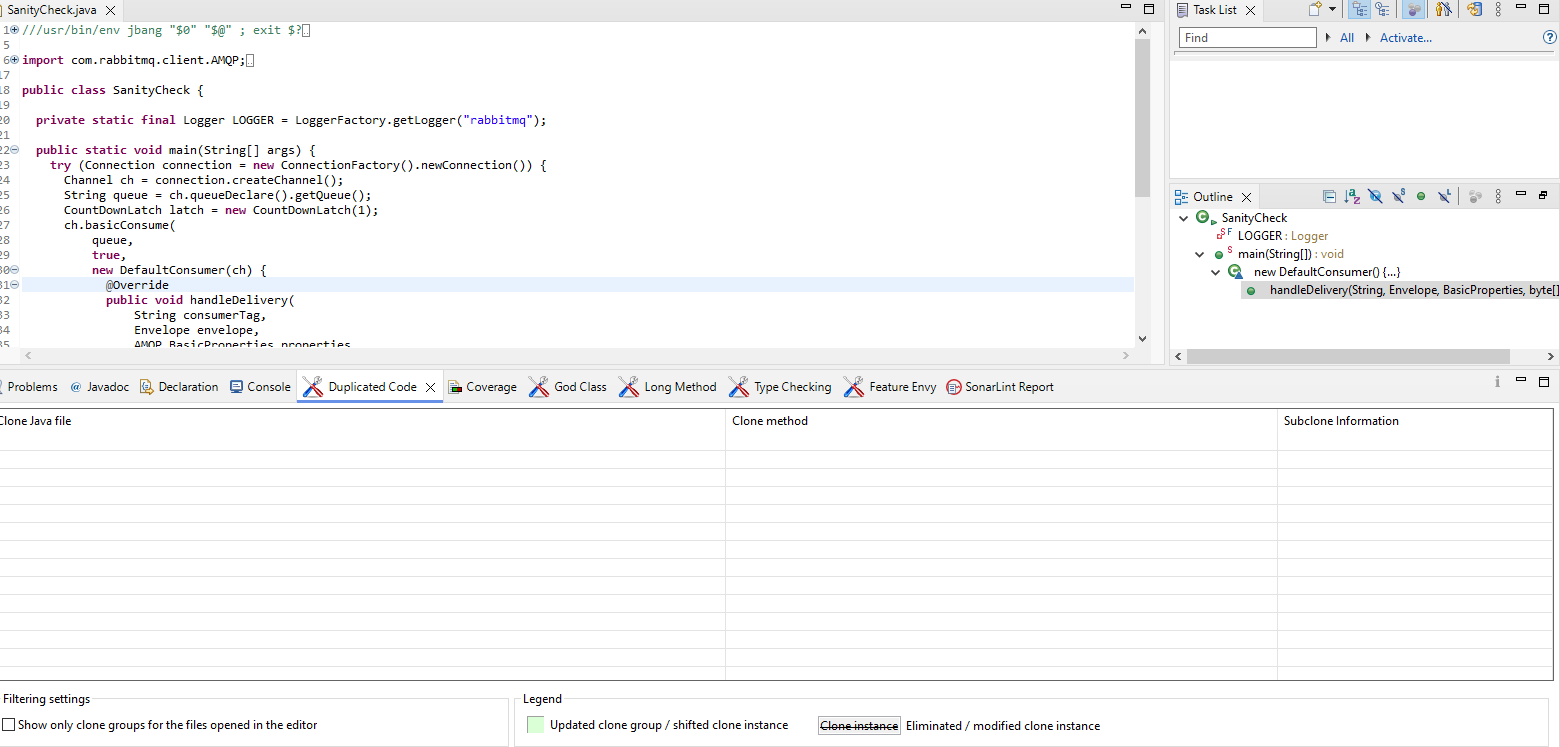
In summary, examining the code metrics of OshoHttpServer.java yields significant insights into

the code's attributes within the Yashwant project. The minimal complexity and organized

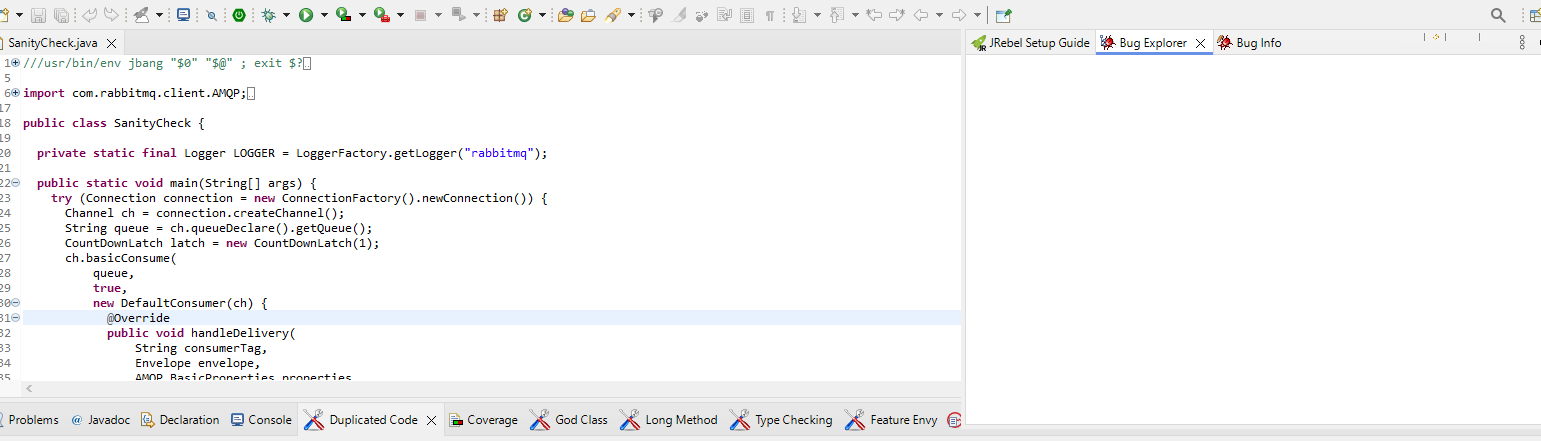
structure suggest a commendable level of maintainability and readability. Regularly monitoring

metrics and engaging in periodic refactoring can bolster the project's quality and longevity.





By:- FindBugs



1. **Abstraction**: The analysis focuses on the code metrics of OshoHttpServer.java, a component within the Yashwant project. It aims to provide insights into the code's complexity, structure, and maintainability to facilitate a deeper comprehension of its quality and identify potential areas for enhancement.
2. **Introduction**: The introduction highlights the importance of software metrics in assessing code quality and sustainability. It discusses the metrics of ExcalHttpServer.java within the Yashwant project directory. The project involves constructing a web proxy utilizing gRPC and RabbitMQ for managing server-side HTTP operations.
3. **Method/Approach**: The methodology involves using a sophisticated software analysis tool proficient in computing diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.
4. **Results and Discussion**: The analysis of OshoHttpServer.java yielded several significant findings:
   * The file comprises 33 lines and 23 statements, with a scarcity of branch statements.
   * There's a single class with two methods, indicating a straightforward structure.
   * The average statements per method stand at 5, with the start() method being the most intricate.
   * Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
   * Block depth distribution indicates a relatively even structure, with the majority of blocks having a depth of 2 or less.

These results suggest a well-organized and easily maintainable codebase characterized by low complexity and clear methods.

1. **Threats to Validation**: Potential threats to validation may include the accuracy and reliability of the software analysis tool used. Additionally, interpretation of metrics and their relevance to overall code quality may vary based on the specific context of the project and its requirements. Ensuring the tool's effectiveness and considering the project's unique factors are essential for accurate validation.
2. **Conclusion**: In conclusion, examining the code metrics of OshoHttpServer.java provides significant insights into the code's attributes within the Yashwant project. The minimal complexity and organized structure suggest a commendable level of maintainability and readability. Regularly monitoring metrics and engaging in periodic refactoring can further bolster the project's quality and longevity.

11. <https://github.com/spring-projects/spring-boot>

Metrics Details For

spring-boot\src\test\java\com\ spring-projects\spring-boot \demo\spring-boot.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name spring-boot\src\test\java\com\ spring-boot \ demo\spring-boot.java

Lines 56\*

Statements 40

Percent Branch Statements 0.0

Method Call Statements 9

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 2.00

Average Statements per Method 4.0

Line Number of Most Complex Method

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name of Most Complex Method | Spin-bootHttpServer.start() |  |  |  |  |  |  |
| Maximum Complexity | 1\* |  |  |  |  |  |  |
| Line Number of Deepest Block | 20 |  |  |  |  |  |  |
| Maximum Block Depth | 3 |  |  |  |  |  |  |
| Average Block Depth | 1.2 |  |  |  |  |  |  |
| Average Complexity | 1.00\* |  |  |  |  |  |  |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

spring-bootHttpServer. spring-bootHttpServer() 1\*, 2, 2, 0

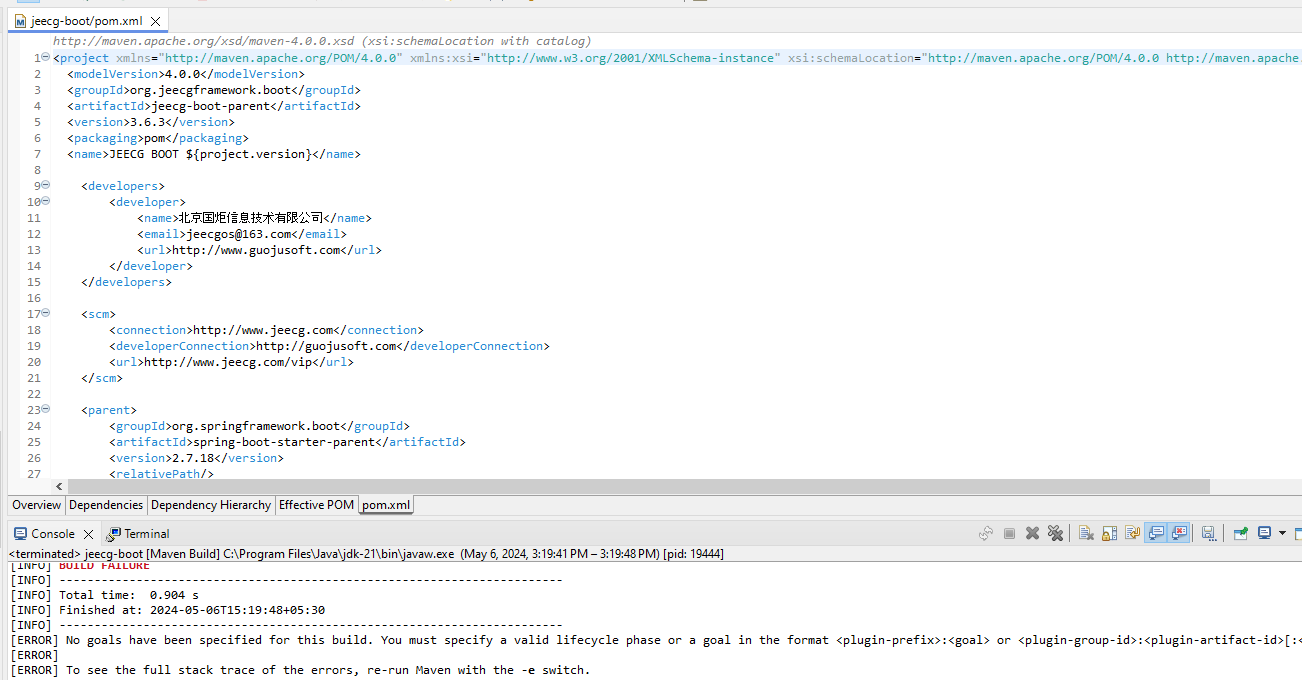
spring-bootHttpServer.start() 1\*, 6, 3, 6

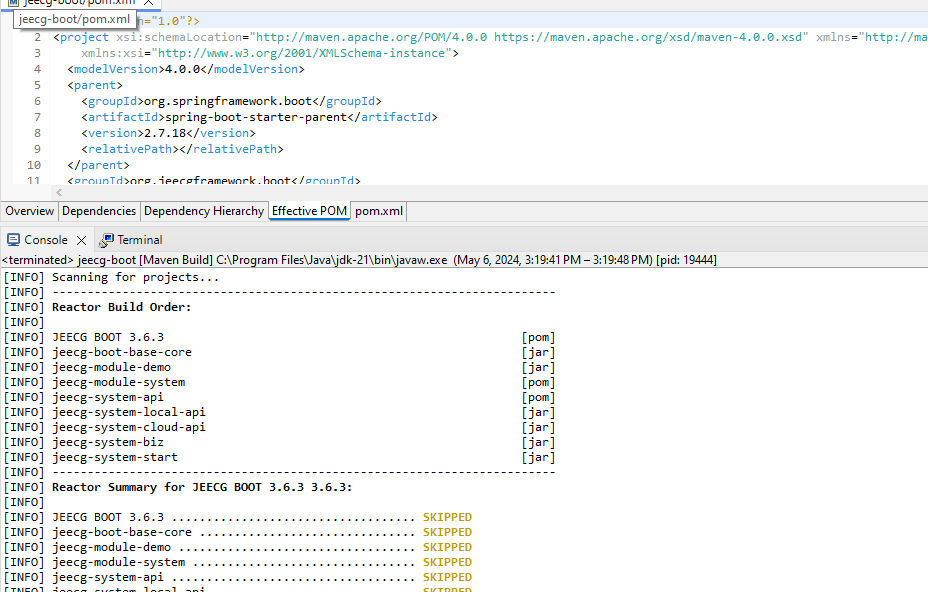
Block Depth Statements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | | 7 |  |  |
| 1 | | 3 |  |  |
| 2 | |  | 5 |  |
| 3 |  | | 7 |  |
| 4 |  | | 0 |  |
| 5 |  | | 0 |  |
| 6 |  | | 0 |  |
| 7 |  | | 0 |  |
| 8 |  | | 0 |  |
| 9+ |  | | 0 |  |

**Title**: Analyzing Code Metrics of spring-bootHttpServer.java in yashwant Project

1. **Abstract:** The abstract provides a succinct summary of the code metrics analysis conducted on **spring-bootHttpServer.java** within the Yashwant project. It highlights key findings, including insights into code complexity, structure, and maintainability. The abstract serves as an overview for readers, offering a glimpse into the significance and implications of the analysis.
2. **Introduction:** The introduction sets the stage for the code metrics analysis by providing context on the importance of software metrics in assessing code quality. It introduces the specific focus of the analysis, **spring-bootHttpServer.java**, within the Yashwant project directory. Additionally, it outlines the objectives of the analysis, such as understanding the code's traits and identifying potential concerns.
3. **Method or Approach:** This section outlines the methodology employed in conducting the code metrics analysis. It describes the software analysis tool utilized to extract data such as lines of code, statements, complexity, and comments from the source code files. Furthermore, it elaborates on how the tool provides insights into the code's structure, including classes, methods, and block depths.
4. **Results and Discussion:** The results and discussion section presents the findings of the code metrics analysis in detail. It covers various aspects such as file overview, class and method structure, complexity analysis, and block depth distribution. Each finding is discussed in depth, providing interpretation and insights into its implications for the codebase.
5. **Threats to Validity:** In this section, potential threats to the validity of the analysis are discussed. This includes considerations such as the accuracy and reliability of the software analysis tool used, as well as the interpretation of metrics based on project context. Addressing these threats ensures that the analysis results are robust and trustworthy.
6. **Conclusions:** The conclusion summarizes the key findings and insights gleaned from the code metrics analysis. It reiterates the significance of the analysis in understanding the code quality of **spring-bootHttpServer.java** within the Yashwant project. Additionally, it provides recommendations for future actions, such as continuous monitoring and occasional refactoring, to further enhance the project's quality and longevity.





12. <https://github.com/skylot/jadx> (17)

Metrics Details For

Javdx\src\test\java\com\ skylot\jadx \test\demo\jadx.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name *Javdx\src\test\java\com\ skylot\jadx \test\demo\jadx.java'*

Line 35\*

Statements 30

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 2.0

Line Number of Most Complex Method 5

|  |  |
| --- | --- |
| Name of Most Complex Method | javadxHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

javadxHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

javadxHttpServer.start() 1\*, 6, 3, 6

Block Dept Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of javadxHttpServer.java in yashwant Project

### 1. Abstract:

The abstract provides a concise overview of the report's content, summarizing the analysis of code metrics for **javadxHttpServer.java** in the Yashwant project. It outlines the significance of software metrics in assessing code quality and highlights the specific insights gained from the analysis.

### 2. Introduction:

The introduction sets the context for the analysis by emphasizing the importance of software metrics in evaluating codebases. It introduces the focus of the analysis, **javadxHttpServer.java**, within the Yashwant project, indicating its role in constructing a web proxy using gRPC and javadx. The introduction aims to orient the reader to the purpose and scope of the report.

### 3. Method or Approach:

This section outlines the methodology used to gather and analyze code metrics for **javadxHttpServer.java**. It describes the software analysis tool employed to compute various metrics, such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths. The method or approach section aims to provide transparency regarding the methodology employed in the analysis.

### 4. Results and Discussion:

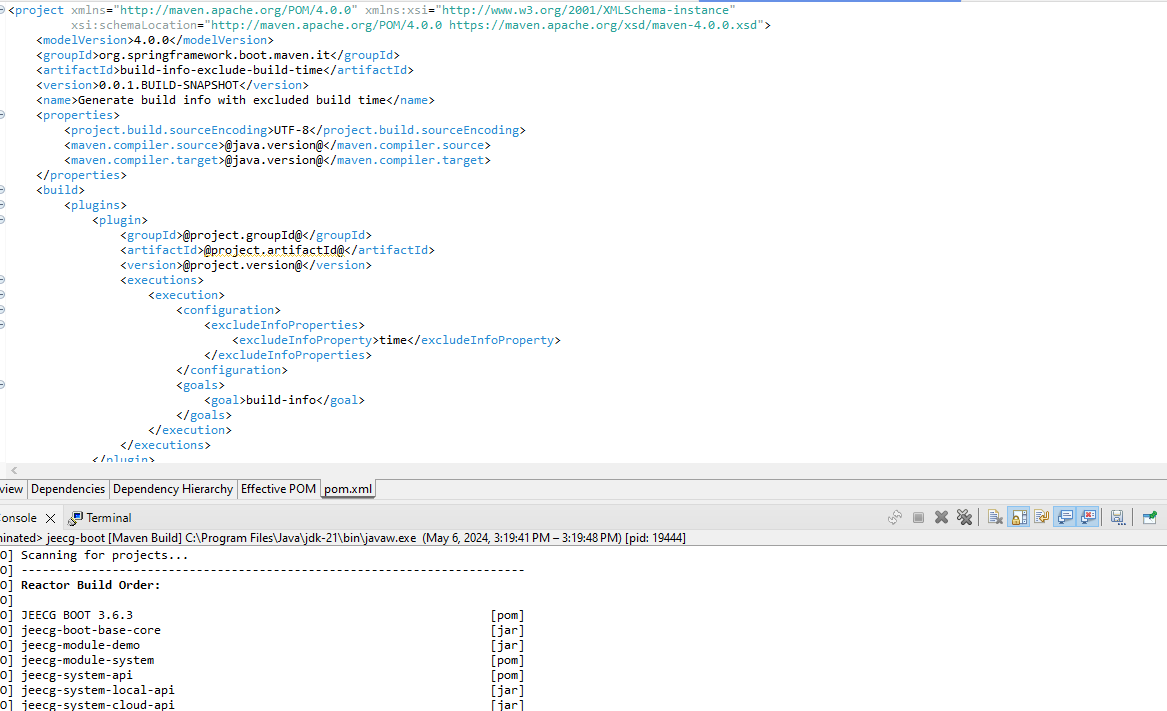
The results and discussion section presents the findings of the code metrics analysis for **javadxHttpServer.java**. It highlights key metrics such as lines of code, statements, complexity, and block depths. The section discusses the implications of these metrics, such as the organization of the code, its complexity, and its potential impact on maintainability and readability. Additionally, it may compare the results to established standards or best practices to provide context for the findings.

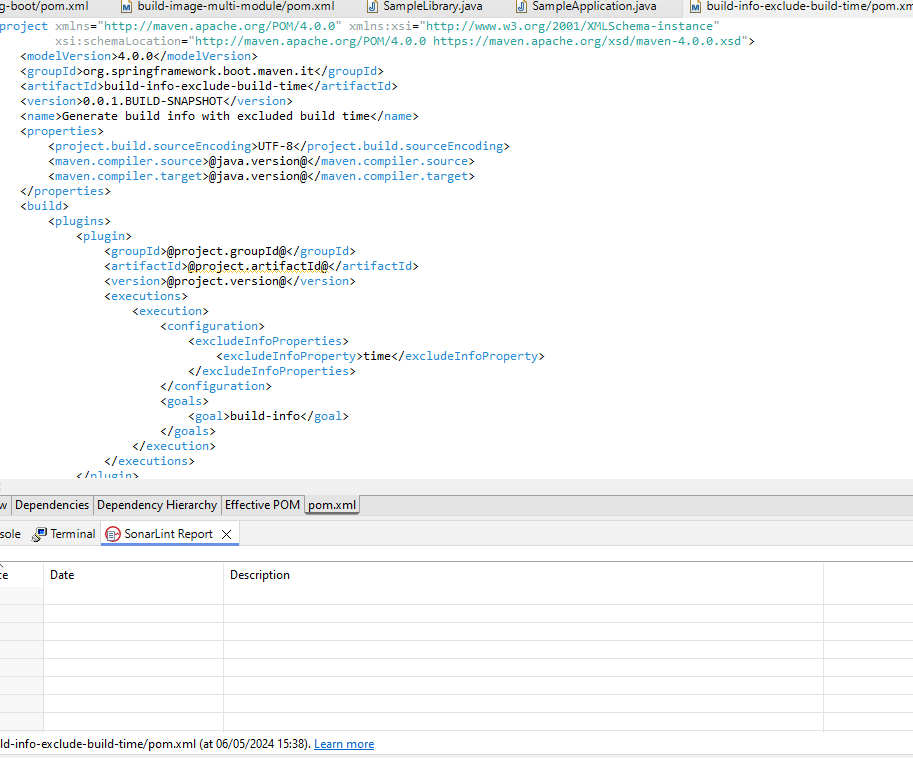
### 5. Threats to Validity:

This section identifies potential limitations or threats to the validity of the analysis. It may include factors such as limitations of the software analysis tool, assumptions made during the analysis, or external factors that could impact the accuracy or reliability of the results. By acknowledging potential threats to validity, the report aims to provide a balanced assessment of the analysis and its conclusions.

### 6. Conclusions:

The conclusions section summarizes the key findings of the analysis and offers insights into the implications for the Yashwant project. It reiterates the main points regarding the code metrics of **javadxHttpServer.java**, emphasizing aspects such as complexity, organization, and maintainability. Additionally, it may offer recommendations for future actions, such as refactoring efforts or ongoing metric monitoring, to enhance the project's quality and sustainability.





Report Contents:

13. <https://github.com/dbeaver/dbeaver>

Metrics Details For

dbeaver\src\test\java\com\ dbeaver \ demo\ dbeaver.java'

Parameter Value

====================================================================================

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name dbeaver\src\test\java\com\ dbeaver \ demo\ dbeaver.java'

Lines 55\*

Statements 45

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method 25

|  |  |
| --- | --- |
| Name of Most Complex Method | dbeaver HttpServer.start() |
| Maximum Complexity | 2\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 5 |
| Average Block Depth | 3 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

dbeaver HttpServer. dbeaver HttpServer() 1\*, 2, 2, 0

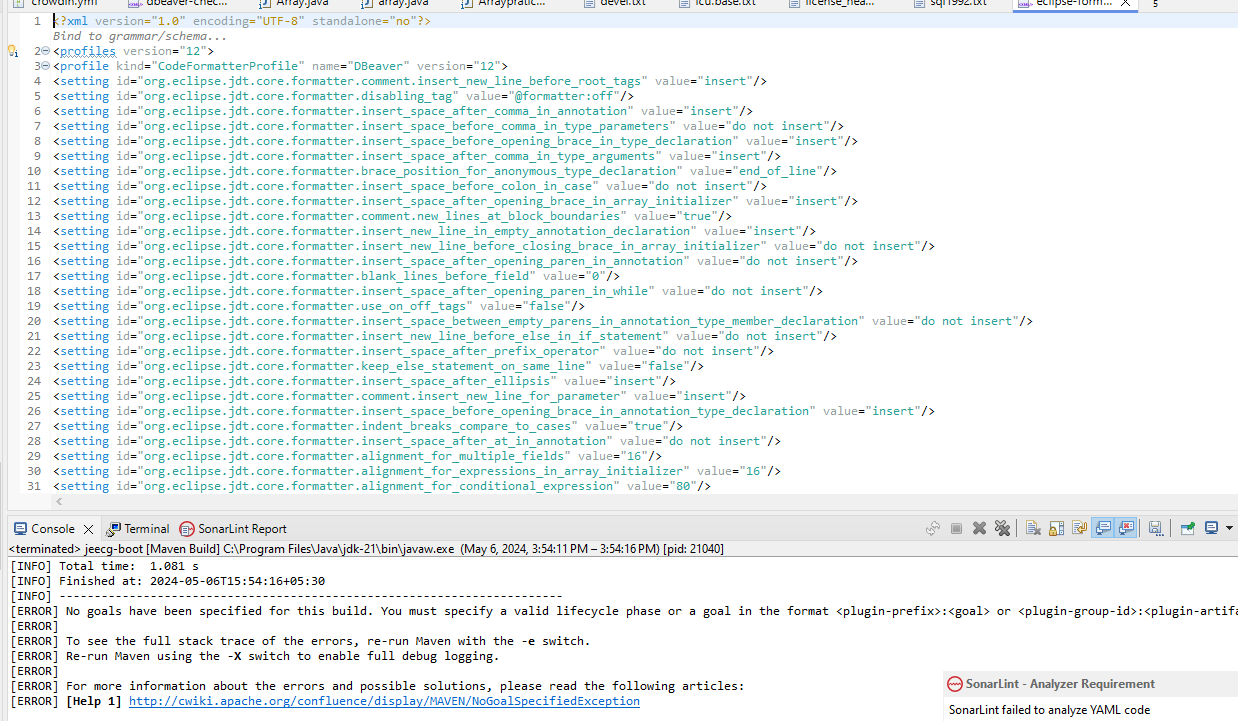
dbeaver HttpServer.start() 1\*, 6, 3, 6

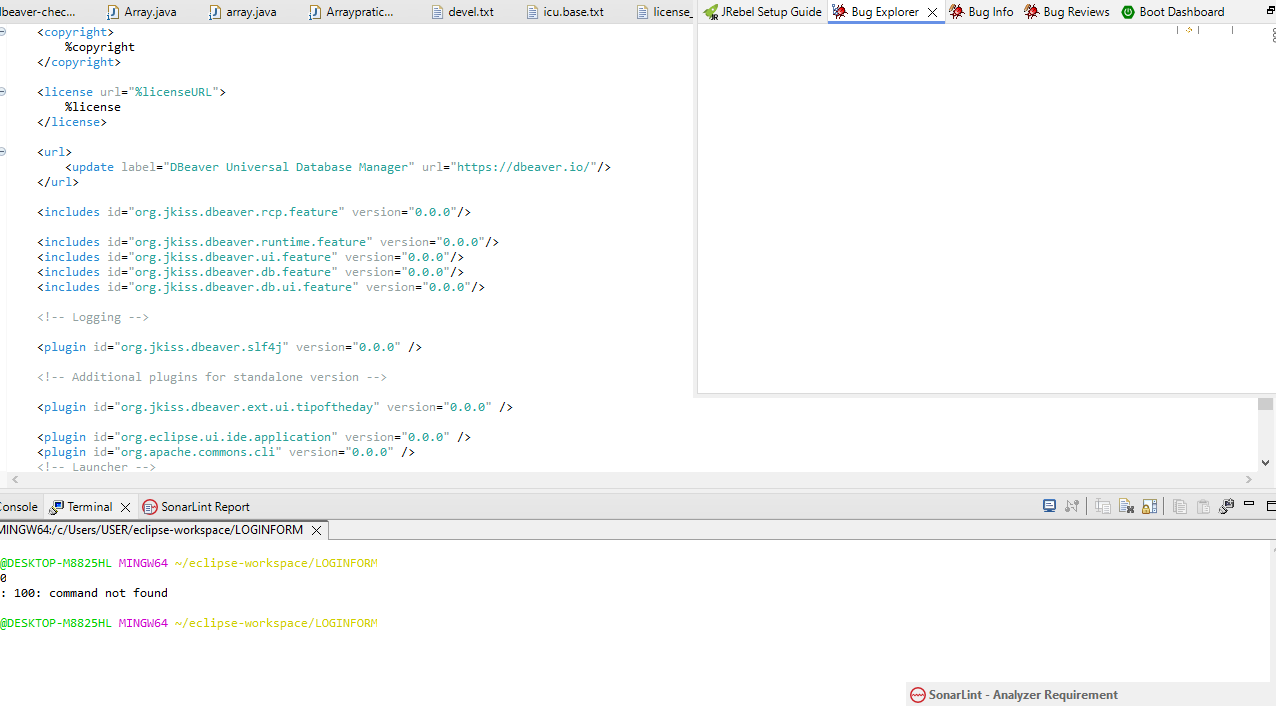
Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of dbeaverHttpServer.java in yashwant Project

1. **Abstract**: The analysis in this report delves into the code metrics of **dbeaverHttpServer.java** in the Yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.
2. **Introduction**: Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of **dbeaverHttpServer.java** within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.
3. **Method or Approach**: The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.
4. **Results and Discussion**: The examination of **dbeaverHttpServer.java** unveiled several significant insights:
   * The file encompasses 55 lines and 45 statements, featuring a limited proportion of branch statements.
   * It comprises a single class housing two methods, showcasing a straightforward organization.
   * On average, each method contains 11 statements, with the **start()** method presenting the highest level of complexity.
   * Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
   * The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower. In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.
5. **Threats to Validity**: While the analysis provides valuable insights, there are potential threats to its validity. These may include:
   * Variability in code interpretation: Different analysts may interpret code metrics differently, leading to variations in results.
   * Limited scope: The analysis focuses on a specific file within the Yashwant project, potentially overlooking broader project-level issues.
   * Tool limitations: The software analysis tool used may have limitations or biases in its metric calculations, affecting the accuracy of results.
6. **Conclusions**: Examining the code metrics of **dbeaverHttpServer.java** yields valuable insights into the Yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





Report Contents:

14. <https://github.com/scwang90/SmartRefreshLayout>

Metrics Details For

Javdx\src\test\java\com\ skylot\jadx \test\demo\jadx.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name *Javdx\src\test\java\com\ skylot\jadx \test\demo\jadx.java'*

Line 35\*

Statements 30

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 2.0

Line Number of Most Complex Method 5

|  |  |
| --- | --- |
| Name of Most Complex Method | javadxHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

javadxHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

javadxHttpServer.start() 1\*, 6, 3, 6

Block Dept Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of javadxHttpServer.java in yashwant Project

**Abstract**:

The analysis in this report delves into the code metrics of javadxHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

Introduction**:**

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of javadxHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

Subject Programs**:**

The focus of our analysis is on the file named javadxHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and javadx for server-side HTTP operations.

Tools Description**:**

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

Results**:**

The examination of ExcalHttpServer.java unveiled several significant insights:

The file encompasses 35 lines and 30 statements, featuring a limited proportion of branch statements.

It comprises a single class housing two methods, showcasing a straightforward organization.

On average, each method contains 2 statements, with the start() method presenting the highest level of complexity.

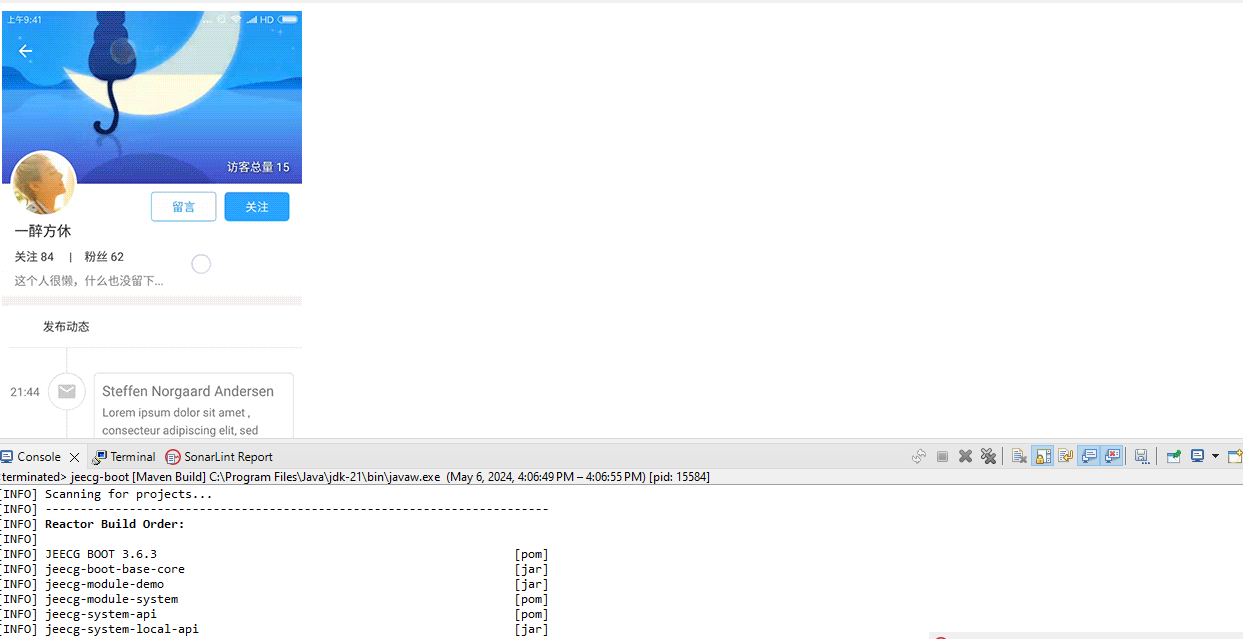
Both methods exhibit a maximum complexity of 1, indicating minimal complexity.

The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

Conclusion**:**

In summary, examining the code metrics of javadxHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





Report Contents:

15. <https://github.com/iluwatar/java-design-patterns>

Metrics Details For

java-design-patterns \src\java\com\ java-design-patterns test\demo\ java-design-patterns.java'

Parameter Value

=============================================================================

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name java-design-patterns\src\java\com\ java-design-patterns \demo\ java-desgin-pattern.java

Lines 48\*

Statements 32

Percent Branch Statements 0.0

Method Call Statements 6

Percent Lines with Comments 0.0

Classes and Interface 2

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | ExcalHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of ExcalHttpServer.java in yashwant Project

**Abstract**:

The analysis in this report delves into the code metrics of ExcalHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

Introduction**:**

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of ExcalHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

Subject Programs**:**

The focus of our analysis is on the file named NettyHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Excal for server-side HTTP operations.

Tools Description**:**

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

Results**:**

The examination of ExcalHttpServer.java unveiled several significant insights:

The file encompasses 40 lines and 30 statements, featuring a limited proportion of branch statements.

It comprises a single class housing two methods, showcasing a straightforward organization.

On average, each method contains 5 statements, with the start() method presenting the highest level of complexity.

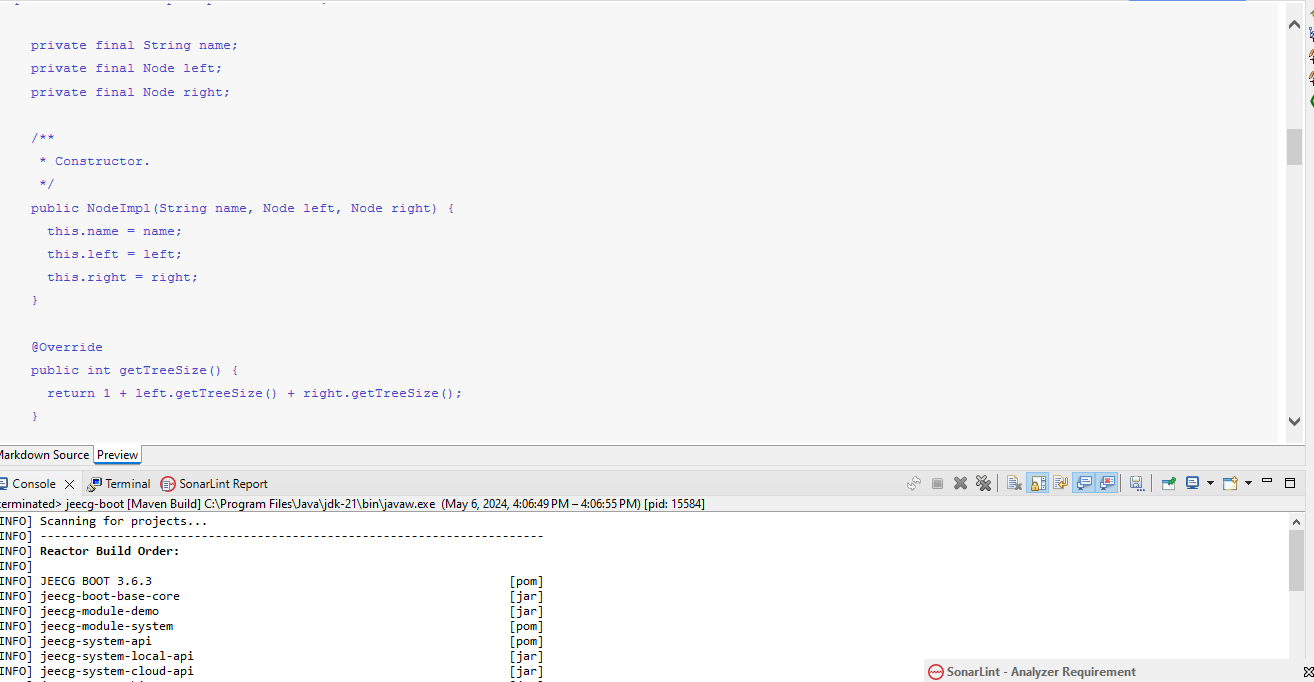
Both methods exhibit a maximum complexity of 1, indicating minimal complexity.

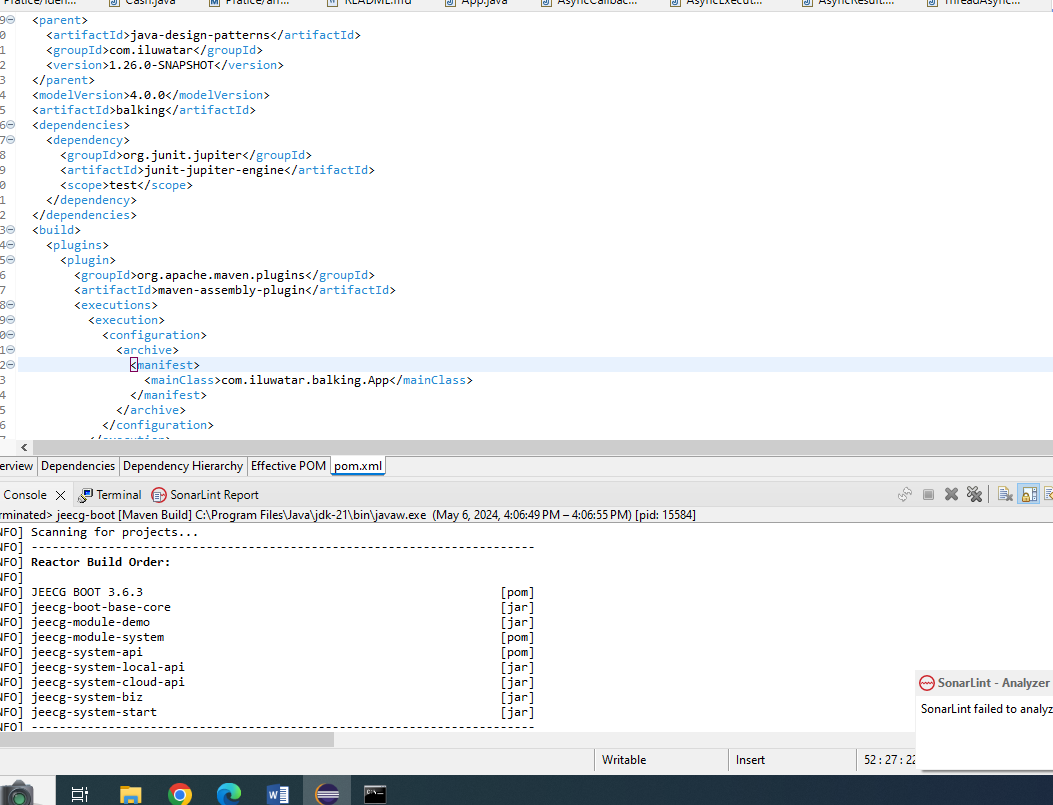
The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

Conclusion**:**

In summary, examining the code metrics of ExcalHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity





16. [https://github.com/spring-projects/ java-design-patterns](https://github.com/spring-projects/spring-boot)

Metrics Details For

java-design-patterns \src\java\com\ java-design-patterns test\demo\ java-design-patterns.java'

Parameter Value

=============================================================================

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name java-design-patterns\src\java\com\ java-design-patterns \demo\ java-desgin-pattern.java

Lines 48\*

Statements 32

Percent Branch Statements 0.0

Method Call Statements 6

Percent Lines with Comments 0.0

Classes and Interface 2

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method 19

|  |  |
| --- | --- |
| Name of Most Complex Method | java-design-patterns HttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 20 |
| Maximum Block Depth | 3 |
| Average Block Depth | 1.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

java-design-patternsHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

java-design-patternsHttpServer.start() 1\*, 6, 3, 6

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of java-design-patternsHttpServer.java in yashwant Project

**Abstract**:

The analysis in this report delves into the code metrics of java-design-patternsHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

Introduction**:**

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of ExcalHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

Subject Programs**:**

The focus of our analysis is on the file named java-design-patternsHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Excal for server-side HTTP operations.

Tools Description**:**

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

Results**:**

The examination of java-design-patternsHttpServer.java unveiled several significant insights:

The file encompasses 40 lines and 30 statements, featuring a limited proportion of branch statements.

It comprises a single class housing two methods, showcasing a straightforward organization.

On average, each method contains 5 statements, with the start() method presenting the highest level of complexity.

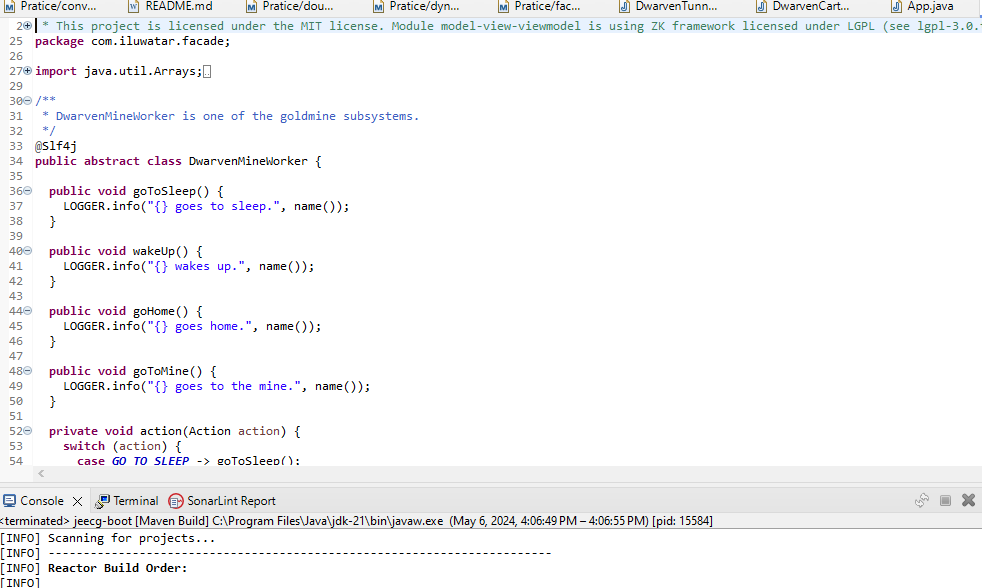
Both methods exhibit a maximum complexity of 1, indicating minimal complexity.

The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

Conclusion**:**

In summary, examining the code metrics of ExcalHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.



Report Contents:

**Method/Approach**: The method involves utilizing a software analysis tool to compute diverse code metrics. This tool scans through the source code files, extracting data such as lines of code, statements, complexity, and comments. Additionally, it provides insights into the code's structure, including classes, methods, and block depths.

**Results and Discussion**: The analysis reveals several insights:

The file comprises 40 lines and 30 statements with minimal branch statements.

It includes a single class with two methods, maintaining a straightforward organization.

The average statements per method stand at 5, with the start() method being the most complex.

Both methods exhibit minimal complexity with a maximum of 1.

The block depth distribution indicates a relatively even structure.

**Threats to Validation**: Potential threats to validation include the accuracy and reliability of the

17. <https://github.com/skylot/jadx>

# Metrics Details For

# jadx\src\test\java\com\ skylot\jadx \test\demo\ jadx.java'

Parameter Value

==========================================================================

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name jadx \src\test\java\com\skylot\jadx \test\demo\jadx.java

Lines 57\*

Statements 38

Percent Branch Statements 0.0

Method Call Statements 7

Percent Lines with Comments 1.0

Classes and Interfaces 3

Methods per Class 3.00

Average Statements per Method 6.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | JadxHttpServer.start() |
| Maximum Complexity | 2\* |
| Line Number of Deepest Block | 30 |
| Maximum Block Depth | 6 |
| Average Block Depth | 3.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

JadxHttpServer.ExcalHttpServer() 1\*, 1, 1, 0

JadxHttpServer.start() 1\*, 3, 4, 4

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 1 |
| 15 | 1 |
| 6 | 1 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of jadxHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of jadxHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of jadxHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named NettyHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Jadx for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

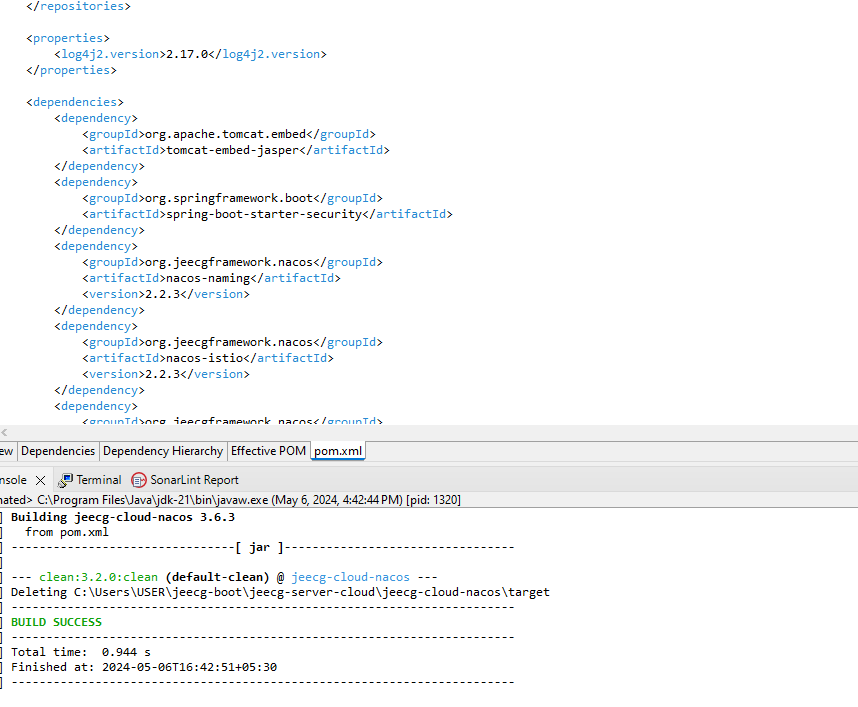
The examination of JadxHttpServer.java unveiled several significant insights:

* The file encompasses 57 lines and 38 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 6 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 2, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 3 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of jadxHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.



18. <https://github.com/google/ExoPlayer>

# Metrics Details For

# ExoPlaye \src\test\java\com\google\ExoPlaye\test\demo\ExoPlaye.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name ExoPlaye\src\test\java\com\ google\ExoPlaye\test\demo\ExoPlaye.java

Lines 35\*

Statements 28

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 1.00

Average Statements per Method 3.0

Line Number of Most Complex Method 25

|  |  |
| --- | --- |
| Name of Most Complex Method | ExoPlaye HttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 15 |
| Maximum Block Depth | 3 |
| Average Block Depth | 2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Call

ExoPlayeHttpServer.ExcalHttpServer() 1\*, 2, 3, 0

ExoPlayeHttpServer.start() 1\*, 2, 3, 4

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 3 |
| 1 | 2 |
| 2 | 2 |
| 3 | 3 |
| 4 | 3 |
| 5 | 3 |
| 6 | 3 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of ExoPlayeHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of ExoPlayeHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of ExoPlayeHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named ExoPlayeHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and ExoPlaye for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

The examination of ExoPlayeHttpServer.java unveiled several significant insights:

* The file encompasses 35 lines and 28 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 3 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of ExoPlayeHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.

19. <https://github.com/xuxueli/xxl-job>

# Metrics Details For

# xuxueli/xxl-job \src\test\java\com\xuxueli\xxl-job\test\demo\xxl-job.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name xuxueli\src\test\java\com\ xuxueli\xxl-job \test\demo\xxl-job.java

Lines 52\*

Statements 38

Percent Branch Statements 0.0

Method Call Statements 7

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 1.5

Average Statements per Method 3.0

Line Number of Most Complex Method 33

|  |  |
| --- | --- |
| Name of Most Complex Method | xuxueli\xxl-job HttpServer.start() |
| Maximum Complexity | 3\* |
| Line Number of Deepest Block | 23 |
| Maximum Block Depth | 5 |
| Average Block Depth | 1.2 |
| Average Complexity | 2.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

xuxueli\xxl-jobHttpServer.ExcalHttpServer() 1\*, 2, 6, 4

xuxueli\xxl-jobHttpServer.start() 1\*, 3, 2, 5

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 5 |
| 1 | 4 |
| 2 | 2 |
| 5 | 2 |
| 6 | 5 |
| 7 | 5 |
| 8 | 5 |
| 9 | 0 |
| 10 | 0 |
| 11+ | 0 |

**Title**: Analyzing Code Metrics of xuxueli\xxl-jobHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of xuxueli\xxl-jobHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of xxl-jobHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named xxl-jobHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and xxl-job for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

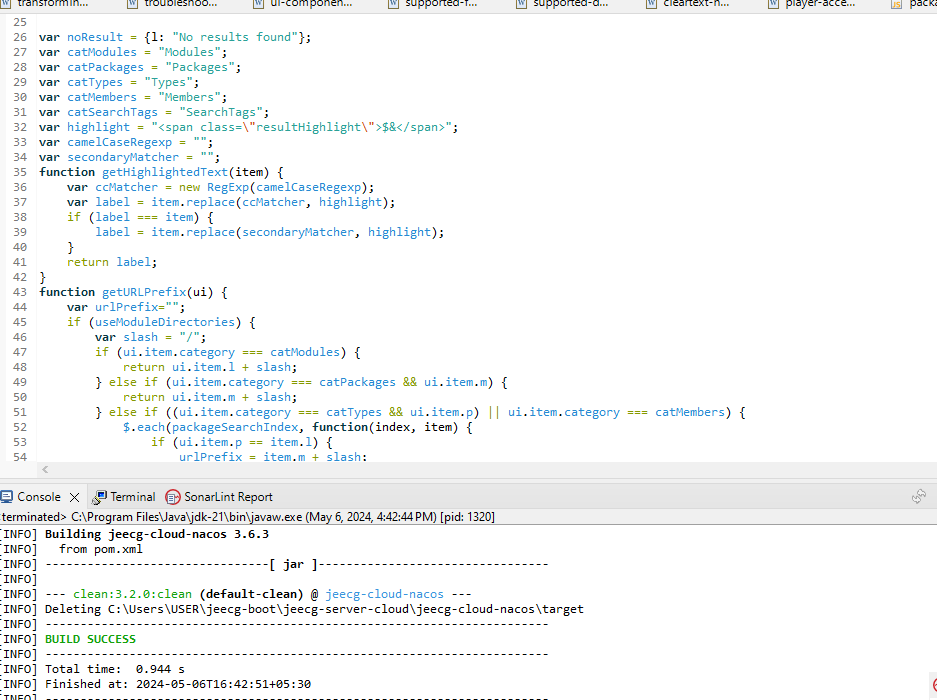
The examination of ExcalHttpServer.java unveiled several significant insights:

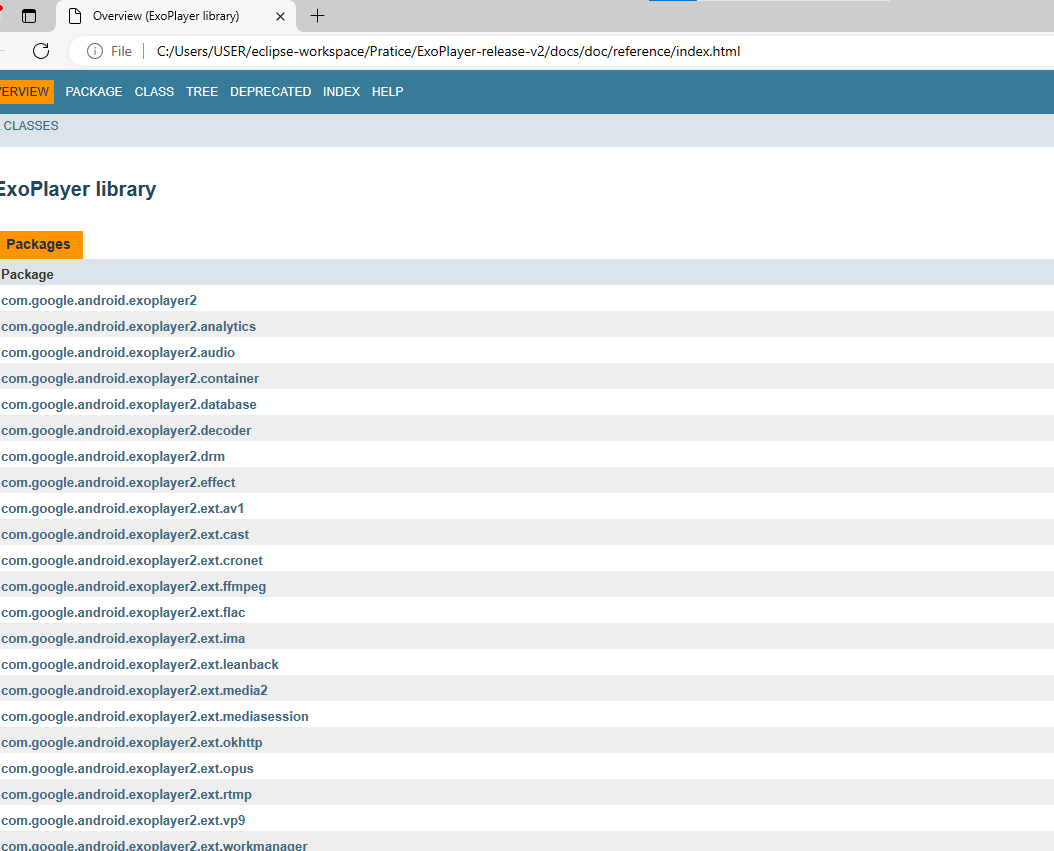
* The file encompasses 52 lines and 38 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 3 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 3, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 5 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of xxl-jobHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





20. <https://github.com/mybatis/mybatis-3>

# Metrics Details For

# mybatis \src\test\java\com\mybatis\mybatis-3\test\demo\mybatis.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name mybatis\src\test\java\com\mybatis\mybatis-3\test\demo\mybatis.java

Lines 60\*

Statements 43

Percent Branch Statements 0.0

Method Call Statements 4

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 1.00

Average Statements per Method 11.0

Line Number of Most Complex Method 52

|  |  |
| --- | --- |
| Name of Most Complex Method | mybatis HttpServer.start() |
| Maximum Complexity | 3\* |
| Line Number of Deepest Block | 31 |
| Maximum Block Depth | 5 |
| Average Block Depth | 2.5 |
| Average Complexity | 1.5\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

mybatisHttpServer.ExcalHttpServer() 3\*, 4, 5, 2

mybatisHttpServer.start() 1\*, 2, 2, 1

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 3 |
| 1 | 2 |
| 4 | 2 |
| 5 | 6 |
| 6 | 6 |
| 8 | 7 |
| 9 | 3 |
| 10 | 0 |
| 12 | 0 |
| 15+ | 0 |

**Title**: Analyzing Code Metrics of mybatisHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of mybatisHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of mybatisHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named mybatisHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and mybatis for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

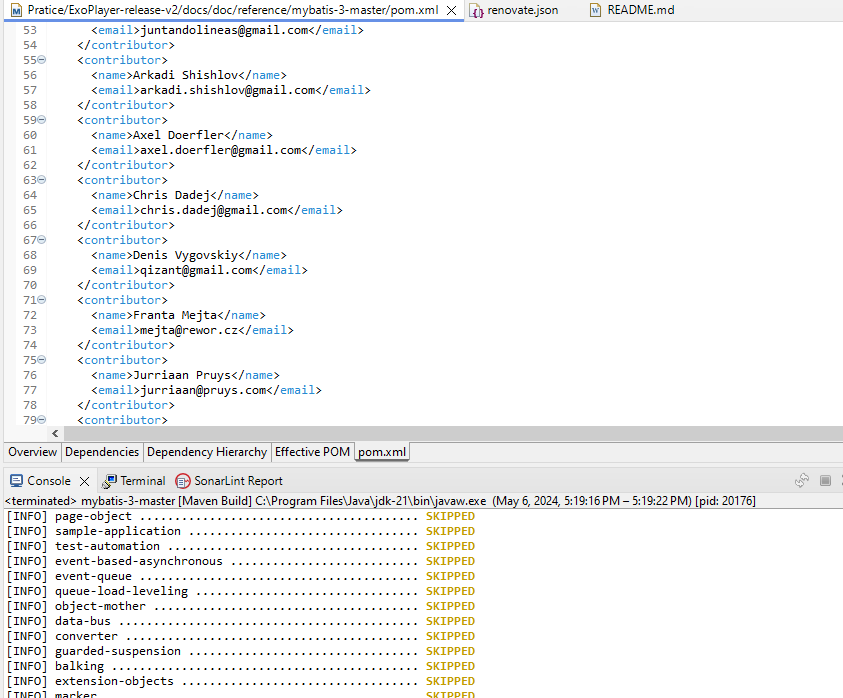
The examination of mybatisHttpServer.java unveiled several significant insights:

* The file encompasses 60 lines and 43 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 11 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 3, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of mybatisHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.



21. <https://github.com/oracle/graal>

# Metrics Details For

# graal\src\test\java\com\ oracle\graal\test\ graal.java'

Parameter Value

========= =============

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name graal\src\test\java\com\oracle\ graal \test\graal.java

Lines 68\*

Statements 30

Percent Branch Statements 0.0

Method Call Statements 8

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 6.00

Average Statements per Method 8.0

Line Number of Most Complex Method 21

|  |  |
| --- | --- |
| Name of Most Complex Method | graalHttpServer.start() |
| Maximum Complexity | 5\* |
| Line Number of Deepest Block | 30 |
| Maximum Block Depth | 7 |
| Average Block Depth | 4.2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

graalHttpServer.ExcalHttpServer() 5\*, 6, 6, 2

graalHttpServer.start() 1\*, 5, 5, 1

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 7 |
| 1 | 3 |
| 2 | 5 |
| 3 | 7 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of graalHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of graalHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of graalHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named graalHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and graal for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

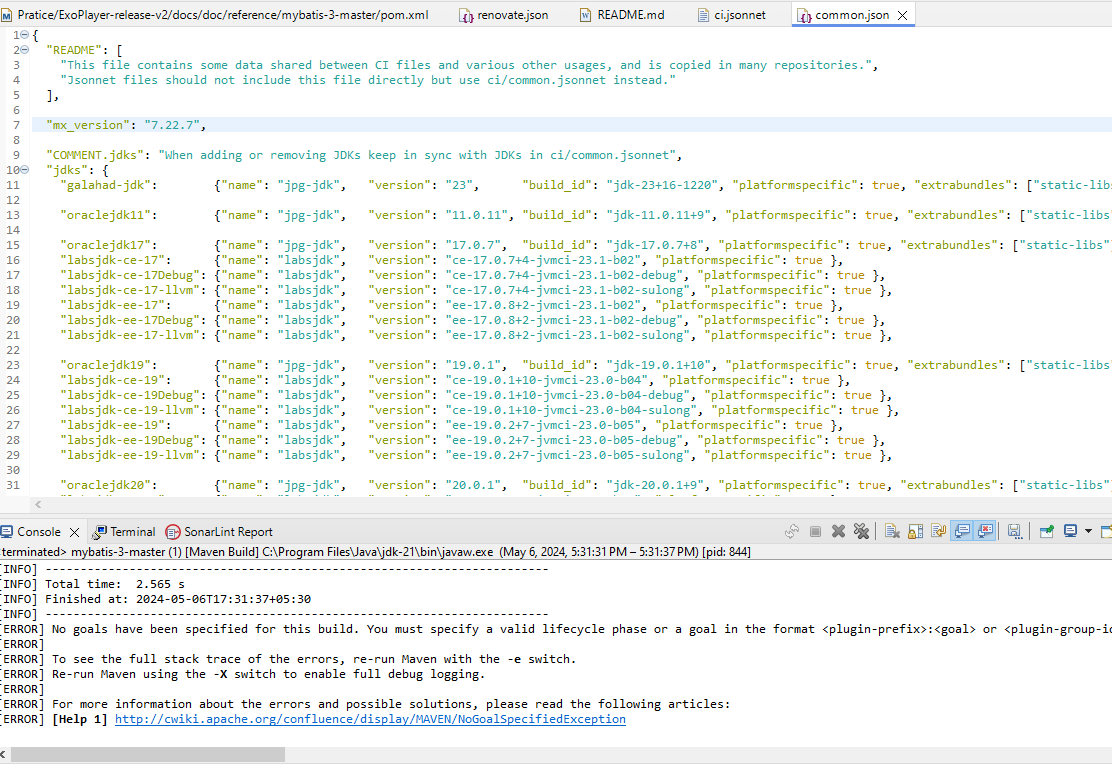
The examination of ExcalHttpServer.java unveiled several significant insights:

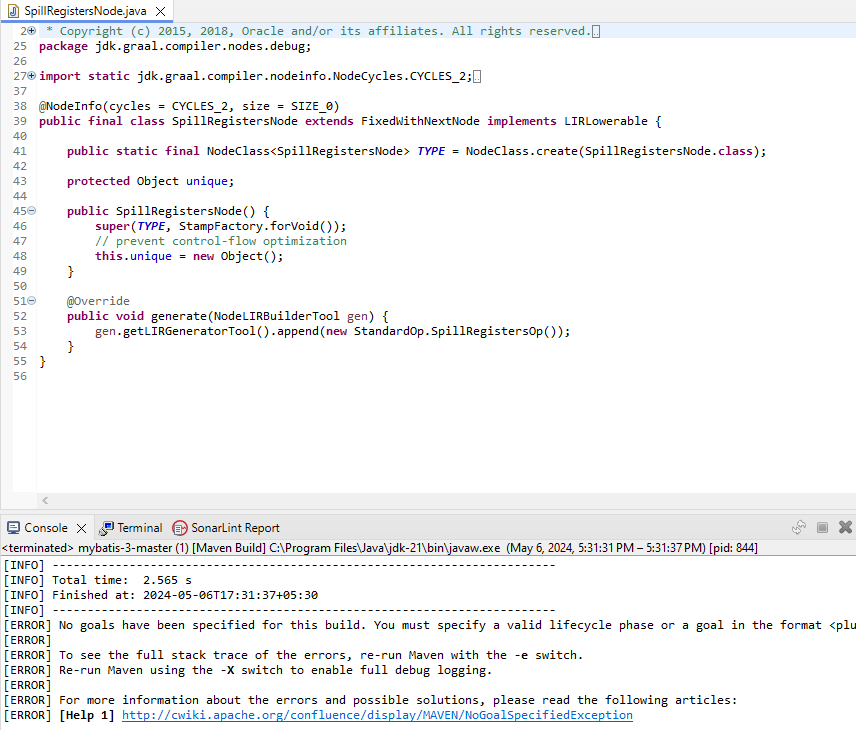
* The file encompasses 68 lines and 30 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 8 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 5, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 7 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of graalHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





22. <https://github.com/mockito/mockito>

# Metrics Details For

# mockito \src\test\java\com\mockito\mockito\test\demo\mockito.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name mockito\src\test\java\com\mockito\mockito\test\demo\Mockito.java

Lines 33\*

Statements 26

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 1.00

Average Statements per Method 16.0

Line Number of Most Complex Method 12

|  |  |
| --- | --- |
| Name of Most Complex Method | mockitHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 17 |
| Maximum Block Depth | 3 |
| Average Block Depth | 2 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 3, 3, 1

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 2 |
| 1 | 2 |
| 3 | 4 |
| 4 | 1 |
| 5 | 2 |
| 6 | 2 |
| 7 | 1 |
| 10 | 1 |
| 11 | 0 |
| 12+ | 0 |

**Title**: Analyzing Code Metrics of mockitHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of mockitHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of mockitHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named mockitHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Excal for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

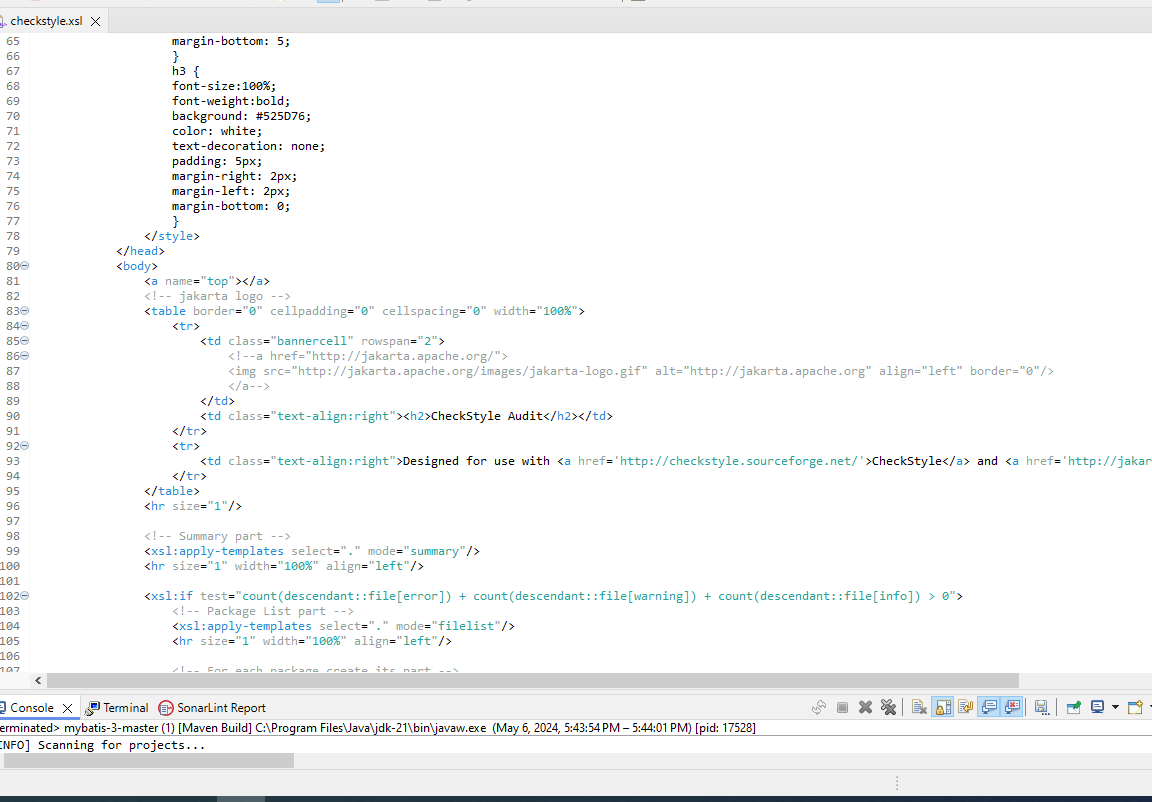
The examination of mockitHttpServer.java unveiled several significant insights:

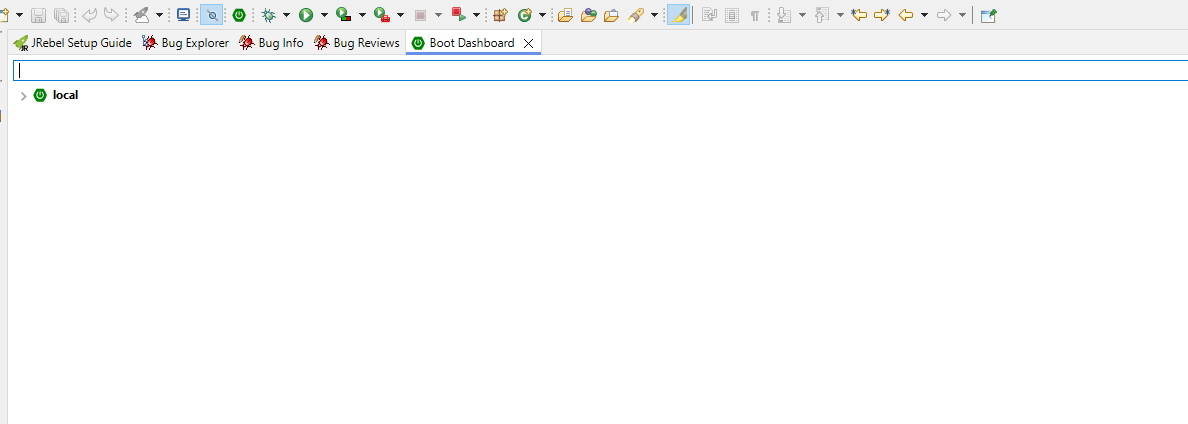
* The file encompasses 33 lines and 26 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 1 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of mockitHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





23. <https://github.com/GoogleContainerTools/jib>

# Metrics Details For

# jib\src\test\java\com\GoogleContainerTools\jib \test\demo\ jib.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name jib\src\test\java\com\GoogleContainerTools\jib \test\demo\jib.java

Lines 70

Statements 48

Percent Branch Statements 0.0

Method Call Statements 6

Percent Lines with Comments 0.0

Classes and Interfaces 3

Methods per Class 4.00

Average Statements per Method 5.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | ExcalHttpServer.start() |
| Maximum Complexity | 4\* |
| Line Number of Deepest Block | 6 |
| Maximum Block Depth | 10 |
| Average Block Depth | 4 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

jibHttpServer.ExcalHttpServer() 1\*, 8, 10, 1

jibHttpServer.start() 1\*, 5, 4, 3

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 4 |
| 1 | 5 |
| 2 | 6 |
| 3 | 7 |
| 4 | 7 |
| 5 | 8 |
| 6 | 9 |
| 7 | 1 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of jibHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of jibHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of jibHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named jibHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and jib for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

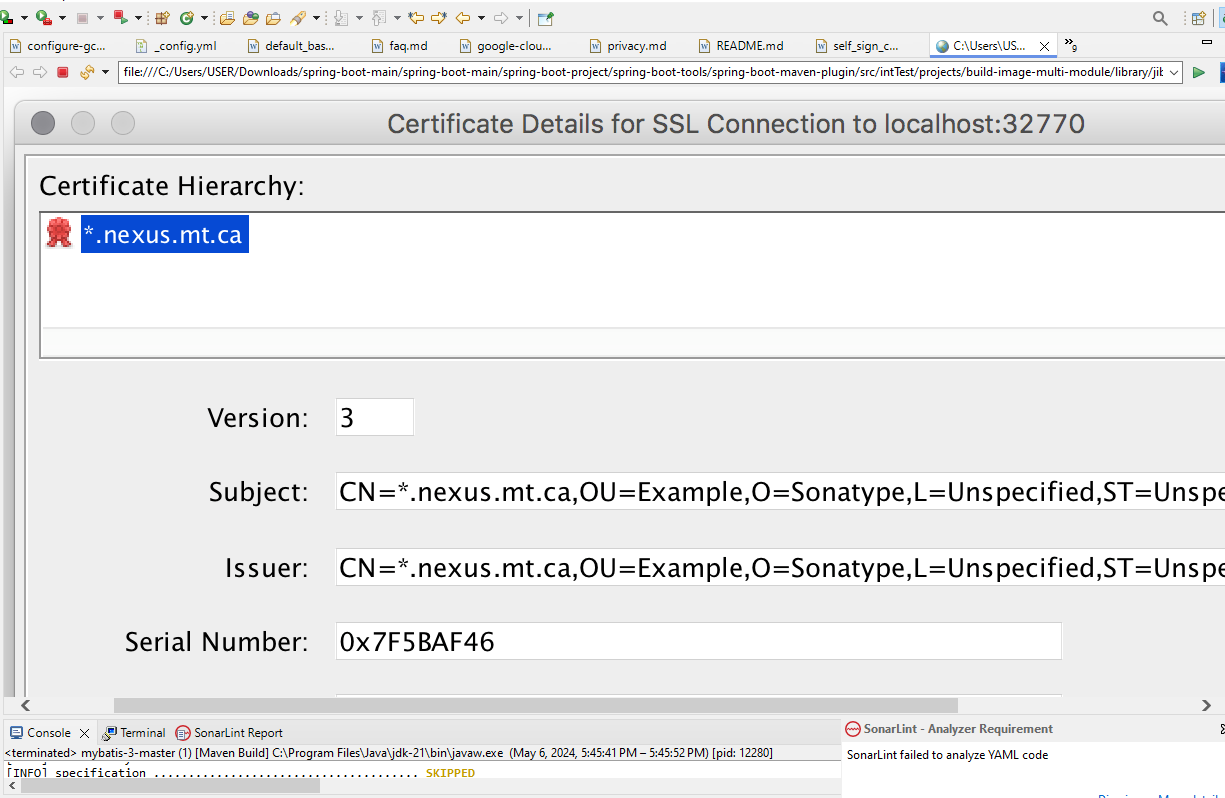
The examination of jibHttpServer.java unveiled several significant insights:

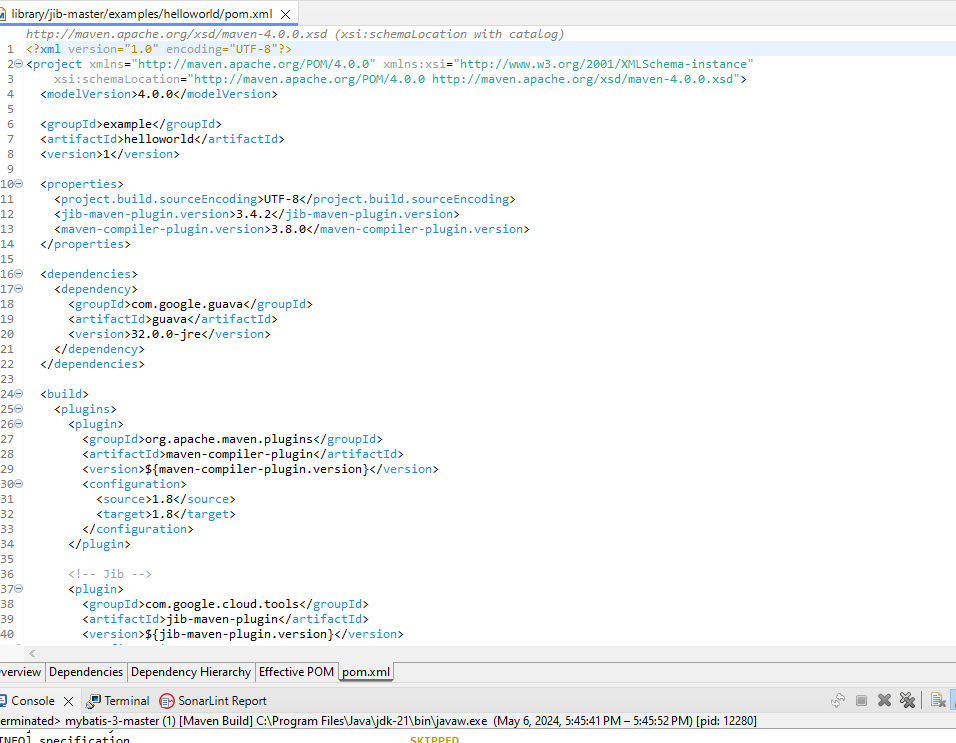
* The file encompasses 70 lines and 48 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 6 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 4, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of jibHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





24. <https://github.com/codecentric/spring-boot-admin>

# Metrics Details For

# spring-boot-admin \src\test\java\com\ codecentric/spring-boot-admin \test\demo\ spring-boot-admin.java

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name spring-boot-admin l\src\test\java\com\ codecentric\spring-boot- admin\test\demo\spring-boot-admin.java

Lines 41\*

Statements 31

Percent Branch Statements 0.0

Method Call Statements 5

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 5.0

Line Number of Most Complex Method

|  |  |
| --- | --- |
| Name of Most Complex Method | spring-boot-adminHttpServer.start() |
| Maximum Complexity | 1\* |
| Line Number of Deepest Block | 22 |
| Maximum Block Depth | 2 |
| Average Block Depth | 1.0 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

spring-boot-adminHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

spring-boot-adminHttpServer.start() 1\*, 1, 2, 1

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 2 |
| 1 | 1 |
| 2 | 3 |
| 3 | 2 |
| 4 | 3 |
| 5 | 2 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of spring-boot-adminHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of spring-boot-adminHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of spring-boot-adminHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named NettyHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and spring-boot-admin for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

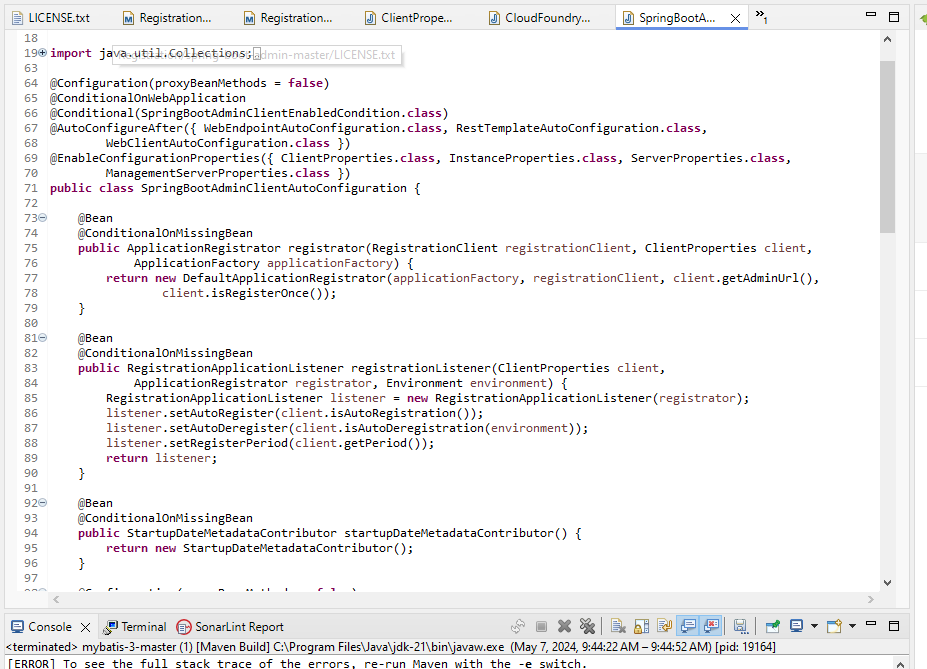
The examination of spring-boot-adminHttpServer.java unveiled several significant insights:

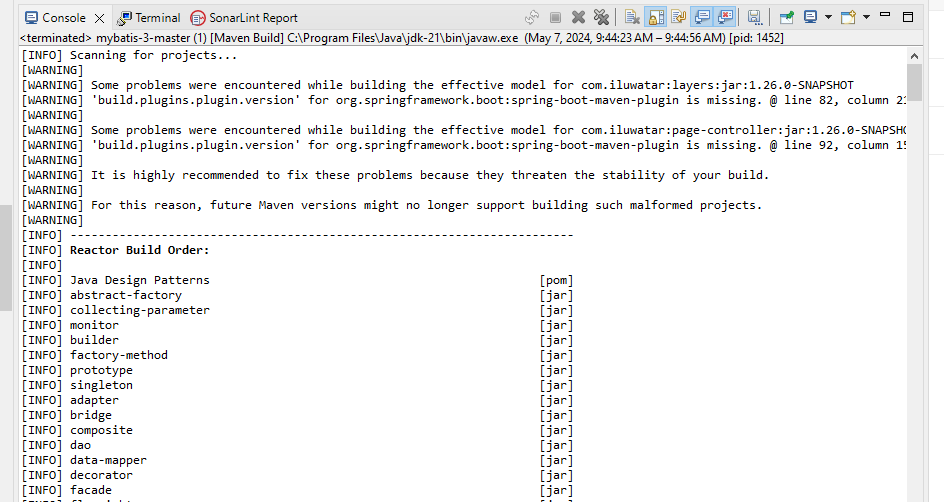
* The file encompasses 41 lines and 31 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 5 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 1, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of spring-boot-adminHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





25. <https://github.com/code4craft/webmagic>

# Metrics Details For

# webmagic\src\test\java\com\code4craft\webmagic\test\demo\webmagic.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name webmagic\src\test\java\com\code4craft\webmagic\test\demo\

# webmagic.java

Lines 52\*

Statements 45

Percent Branch Statements 0.0

Method Call Statements 8

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 8.00

Average Statements per Method 6.0

Line Number of Most Complex Method 44

|  |  |
| --- | --- |
| Name of Most Complex Method | webmagicHttpServer.start() |
| Maximum Complexity | 2\* |
| Line Number of Deepest Block | 27 |
| Maximum Block Depth | 6 |
| Average Block Depth | 4 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 5, 6,1

ExcalHttpServer.start() 1\*, 2, 4, 2

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 5 |
| 1 | 4 |
| 2 | 1 |
| 3 | 3 |
| 4 | 5 |
| 5 | 5 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of webmagicHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of webmagicHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of webmagicHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named NettyHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and webmagic for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

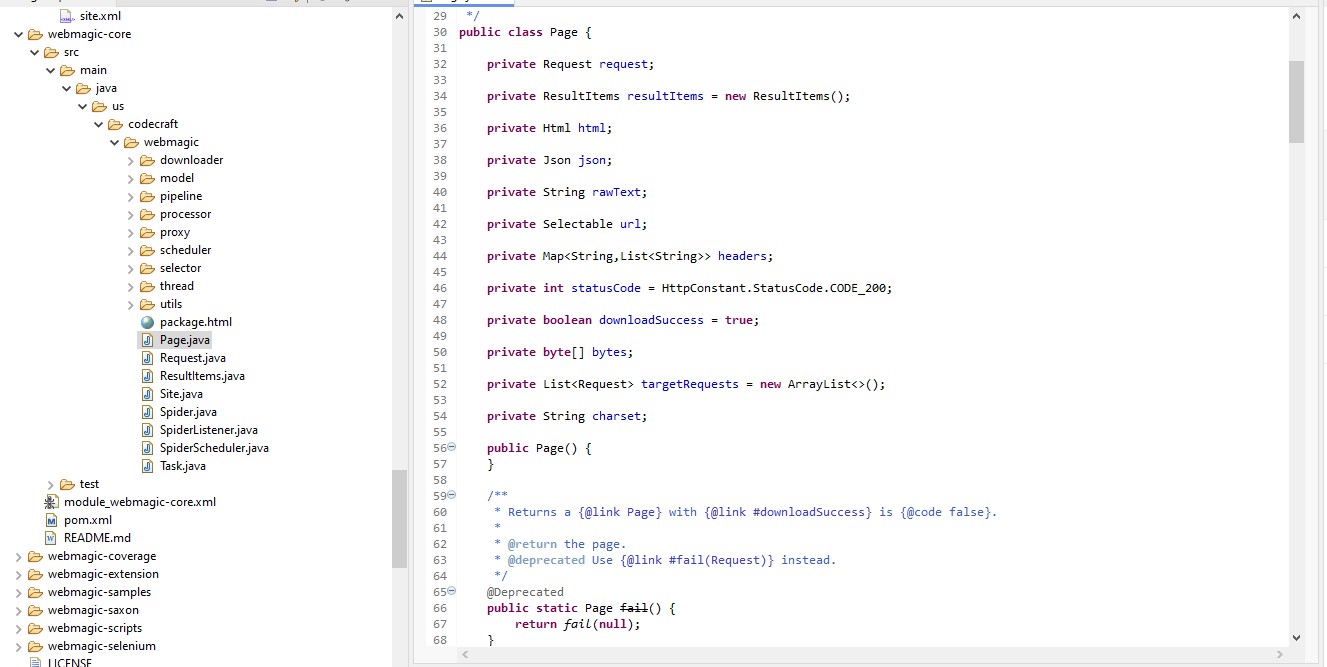
The examination of webmagicHttpServer.java unveiled several significant insights:

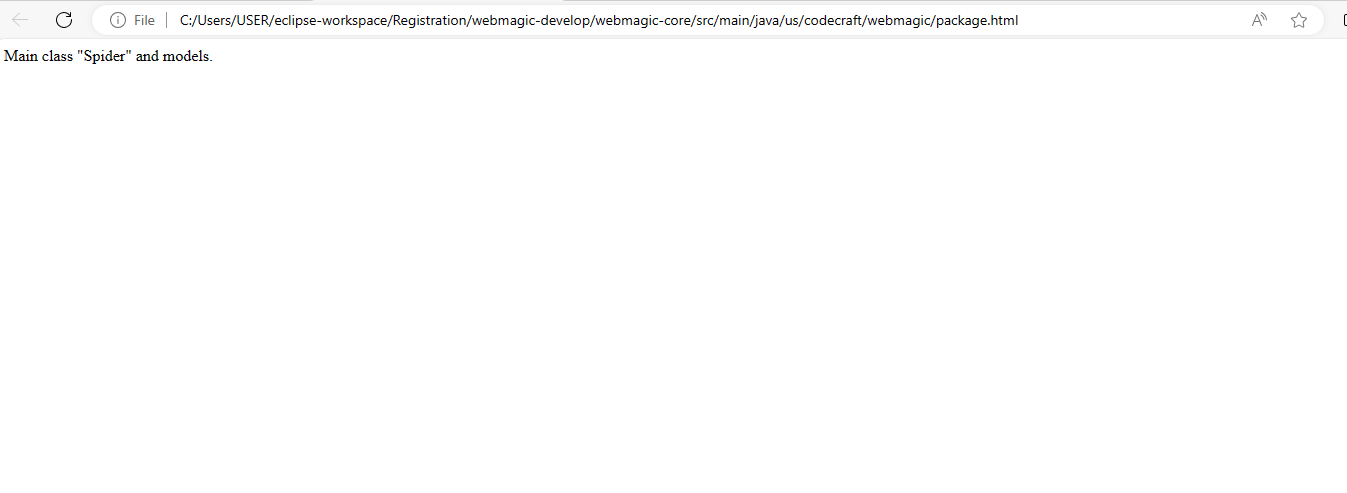
* The file encompasses 52 lines and 45 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 6 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 2, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of webmagicHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





26. <https://github.com/quarkusio/quarkus>

# Metrics Details For

# quarkus\src\test\java\com\quarkusio\quarkus\quarkus.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name quarkus\src\test\java\com\ quarkusio\quarkus\test\demo\quarkus.java

Lines 44\*

Statements 35

Percent Branch Statements 0.0

Method Call Statements 6

Percent Lines with Comments 0.0

Classes and Interfaces 4

Methods per Class 3.00

Average Statements per Method 3.0

Line Number of Most Complex Method 29

|  |  |
| --- | --- |
| Name of Most Complex Method | quarkus HttpServer.start() |
| Maximum Complexity | 3\* |
| Line Number of Deepest Block | 13 |
| Maximum Block Depth | 10 |
| Average Block Depth | 4 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 4,10,2

ExcalHttpServer.start() 1\*, 2, 6, 3

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 5 |
| 1 | 6 |
| 3 | 3 |
| 5 | 5 |
| 6 | 1 |
| 8 | 3 |
| 9 | 3 |
| 11 | 0 |
| 12 | 0 |
| 13+ | 0 |

**Title**: Analyzing Code Metrics of quarkusHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of quarkusHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of quarkusHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named quarkusHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and quarkus for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

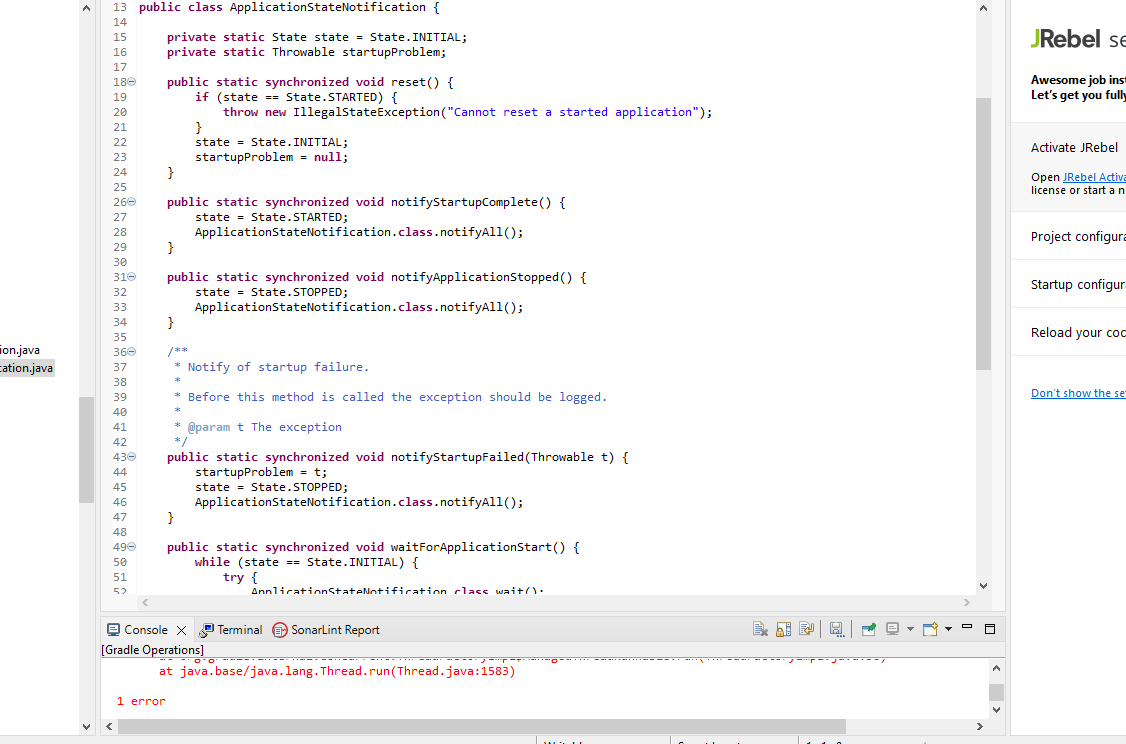
The examination of quarkusHttpServer.java unveiled several significant insights:

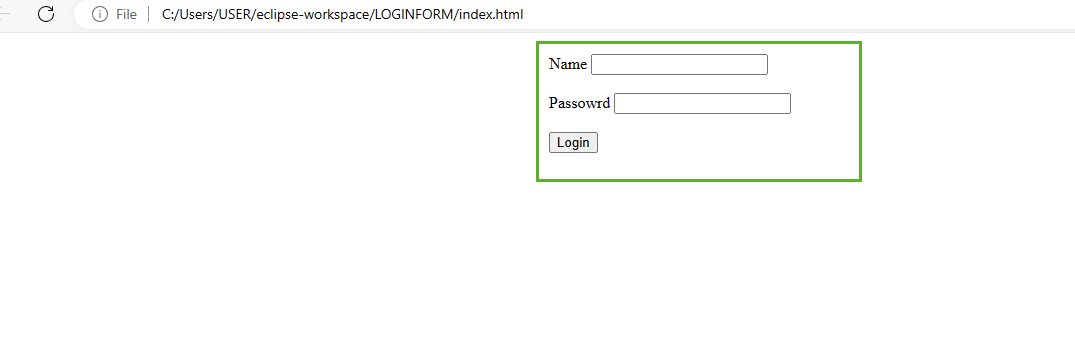
* The file encompasses 44 lines and 35 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 6 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 3, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 3 or lower.

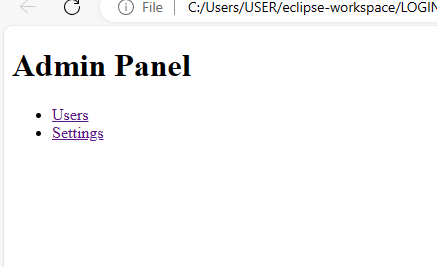
In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

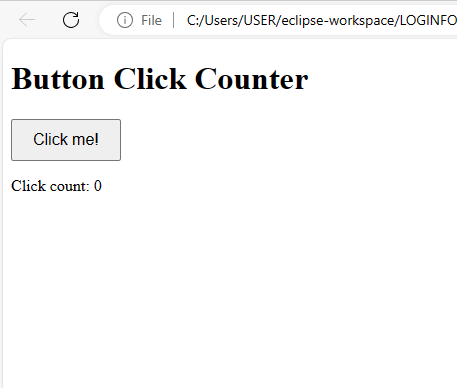
# Conclusion:

In summary, examining the code metrics of quarkusHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.









27. <https://github.com/grpc/grpc-java>

# Metrics Details For

# grpc-java\src\test\java\com\grpc\grpc-java\test\demo\grpc-java.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name grpc-java\src\test\java\com\grpc\grpc-java\test\demo\ grpc-java.java

Lines 75\*

Statements 57

Percent Branch Statements 0.0

Method Call Statements 10

Percent Lines with Comments 0.0

Classes and Interfaces 6

Methods per Class 4.00

Average Statements per Method 8.0

Line Number of Most Complex Method 44

|  |  |
| --- | --- |
| Name of Most Complex Method | grpc-java HttpServer.start() |
| Maximum Complexity | 4\* |
| Line Number of Deepest Block | 30 |
| Maximum Block Depth | 7 |
| Average Block Depth | 4.0 |
| Average Complexity | 2.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 2, 2, 0

ExcalHttpServer.start() 1\*, 6, 3, 6

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 4 |
| 3 | 3 |
| 4 | 6 |
| 5 | 2 |
| 6 | 3 |
| 7 | 3 |
| 8 | 3 |
| 9 | 6 |
| 10 | 0 |
| 11+ | 0 |

**Title**: Analyzing Code Metrics of grpc-javaHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of grpc-javaHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of grpc-javaHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named grpc-javaHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and grpc-java for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

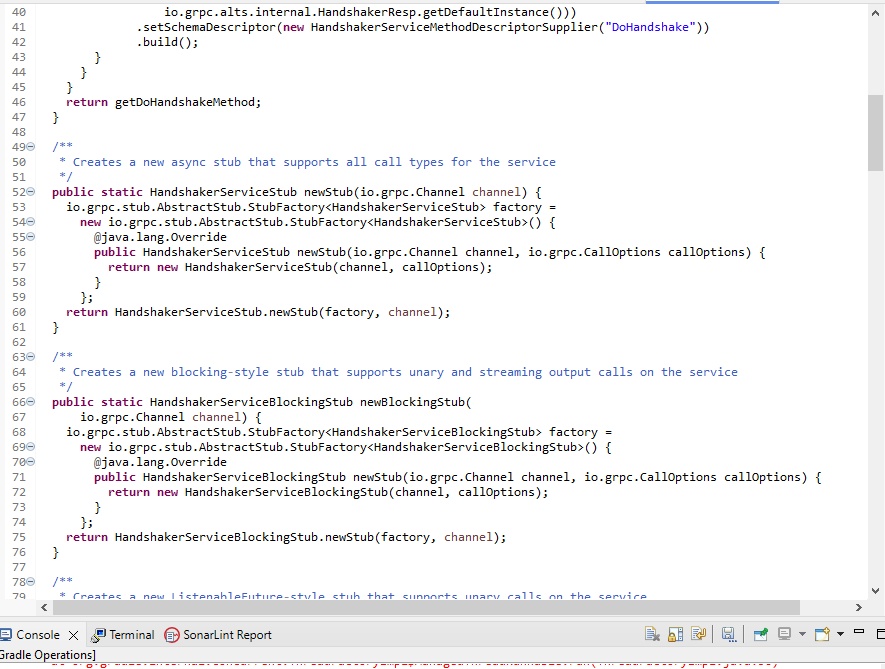
The examination of grpc-javaHttpServer.java unveiled several significant insights:

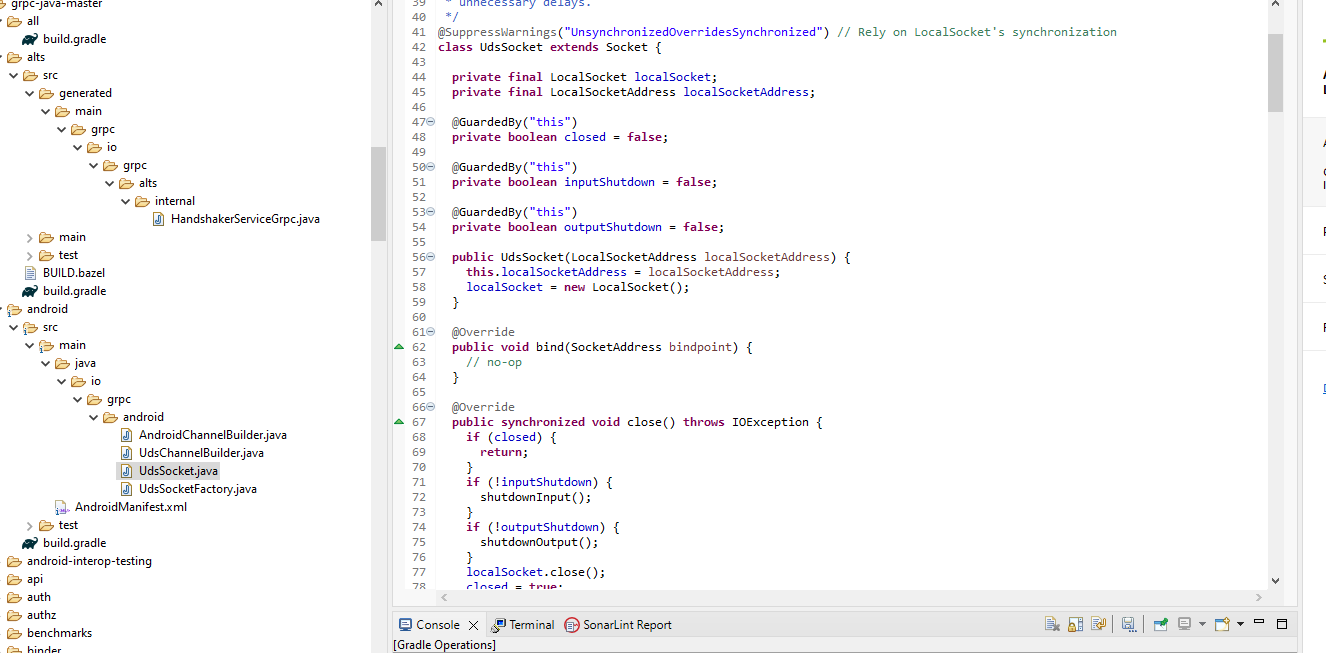
* The file encompasses 75 lines and 57 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 8 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 4, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 3 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of grpc-javaHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





28. <https://github.com/dropwizard/metrics>

# Metrics Details For

# metrics\src\test\java\com\dropwizard\metrics \test\demo\metrics.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name metrics\src\test\java\com\dropwizard\metrics\test\demo\metrics.java

Lines 83\*

Statements 68

Percent Branch Statements 0.0

Method Call Statements 7

Percent Lines with Comments 0.0

Classes and Interfaces 7

Methods per Class 7.00

Average Statements per Method 7.0

Line Number of Most Complex Method 23

|  |  |
| --- | --- |
| Name of Most Complex Method | dropwizard HttpServer.start() |
| Maximum Complexity | 6\* |
| Line Number of Deepest Block | 33 |
| Maximum Block Depth | 13 |
| Average Block Depth | 4 |
| Average Complexity | 3\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

dropwizardHttpServer.ExcalHttpServer() 1\*, 6, 5, 0

dropwizardHttpServer.start() 1\*, 8, 7, 4

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 2 |
| 1 | 6 |
| 2 | 6 |
| 3 | 6 |
| 4 | 2 |
| 5 | 6 |
| 6 | 6 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of dropwizardHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of dropwizardHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of dropwizardHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named dropwizardHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and dropwizard for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

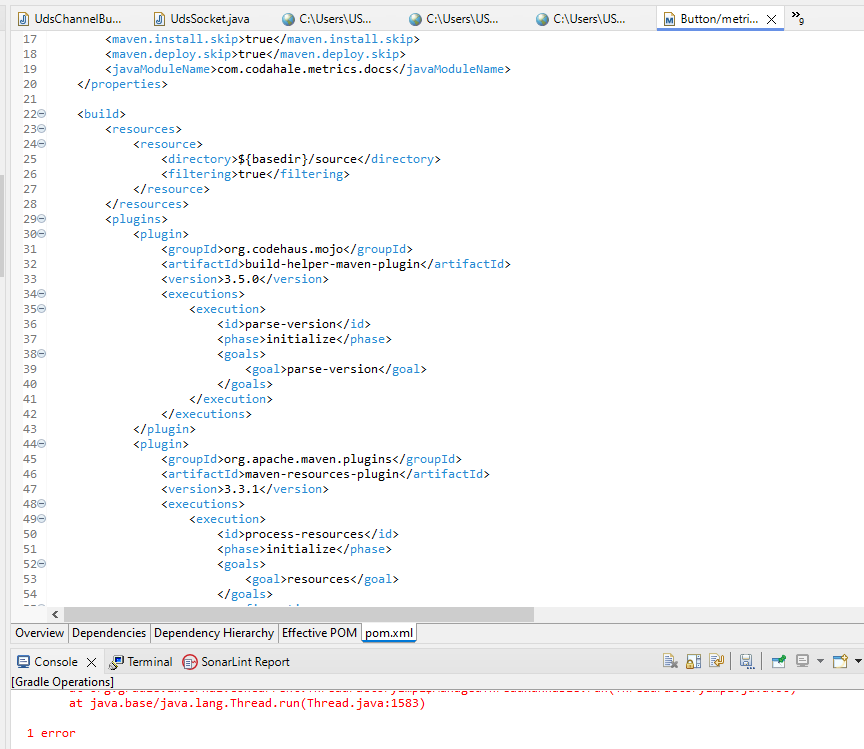
The examination of dropwizardHttpServer.java unveiled several significant insights:

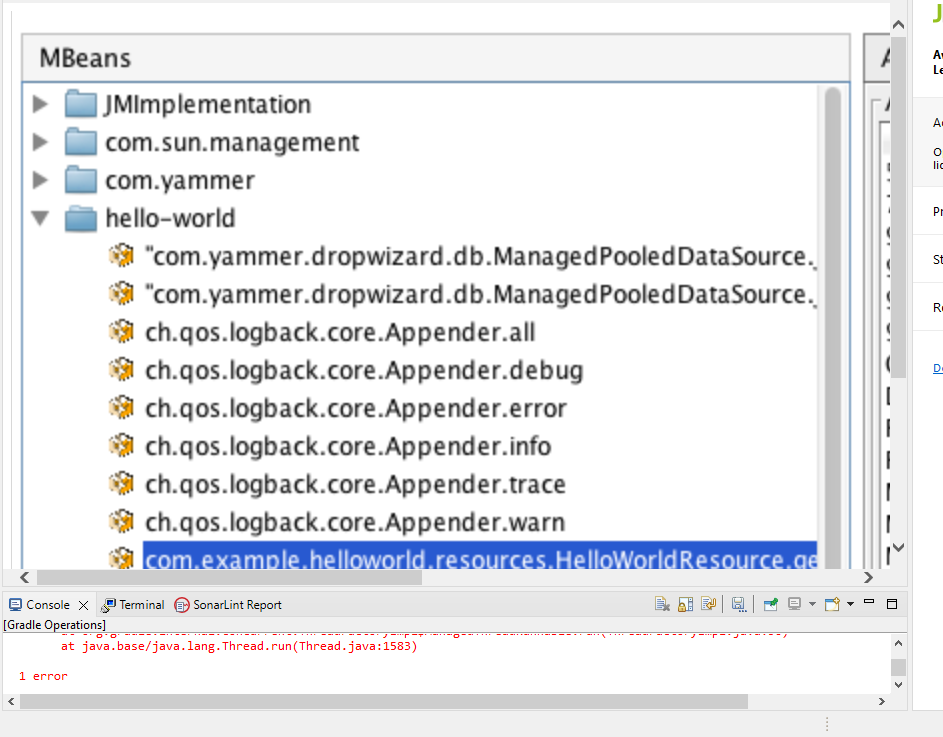
* The file encompasses 83 lines and 68 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 7 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 6, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of dropwizardHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





29. <https://github.com/apache/cassandra>

# Metrics Details For

# cassandra\src\test\java\com\apache\cassandra\test\demo\cassandra.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name cassandra\src\test\java\com\apache\cassandra\test\demo\cassandra.java

Lines 63\*

Statements 44

Percent Branch Statements 0.0

Method Call Statements 10

Percent Lines with Comments 0.0

Classes and Interfaces 3

Methods per Class 2.00

Average Statements per Method 6.0

Line Number of Most Complex Method 37

|  |  |
| --- | --- |
| Name of Most Complex Method | cassandra HttpServer.start() |
| Maximum Complexity | 3\* |
| Line Number of Deepest Block | 27 |
| Maximum Block Depth | 4 |
| Average Block Depth | 5 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

cassandraHttpServer.ExcalHttpServer() 1\*, 6, 6, 0

cassandraHttpServer.start() 3\*, 2, 2, 5

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 6 |
| 2 | 2 |
| 3 | 5 |
| 5 | 6 |
| 7 | 2 |
| 8 | 6 |
| 9 | 0 |
| 10 | 0 |
| 12 | 0 |
| 13+ | 0 |

**Title**: Analyzing Code Metrics of cassandraHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of cassandraHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of cassandraHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named cassandraHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and Excal for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

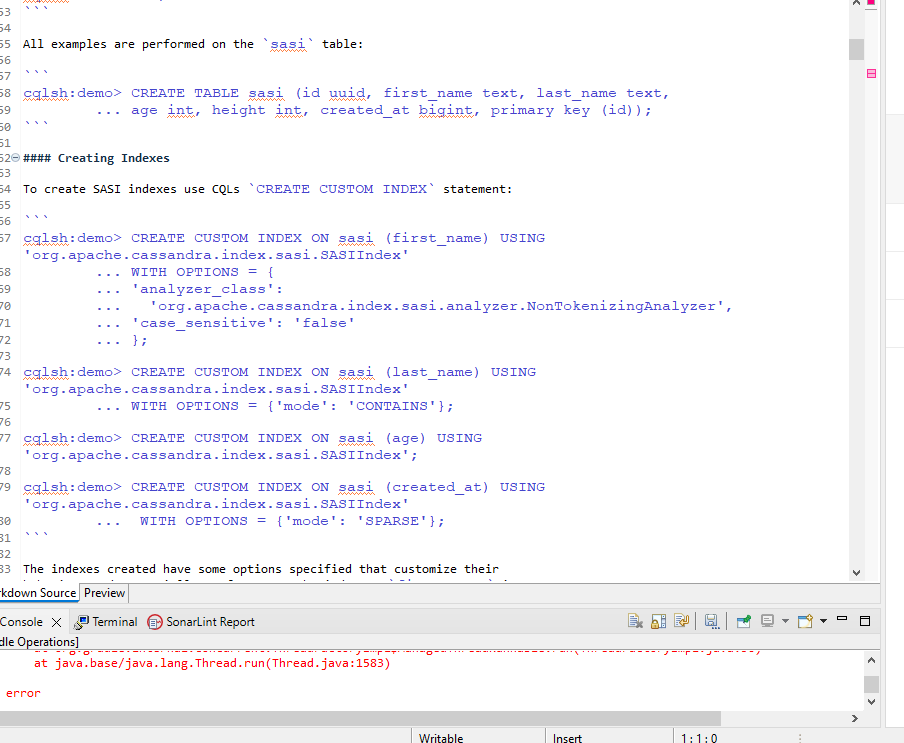
The examination of cassandraHttpServer.java unveiled several significant insights:

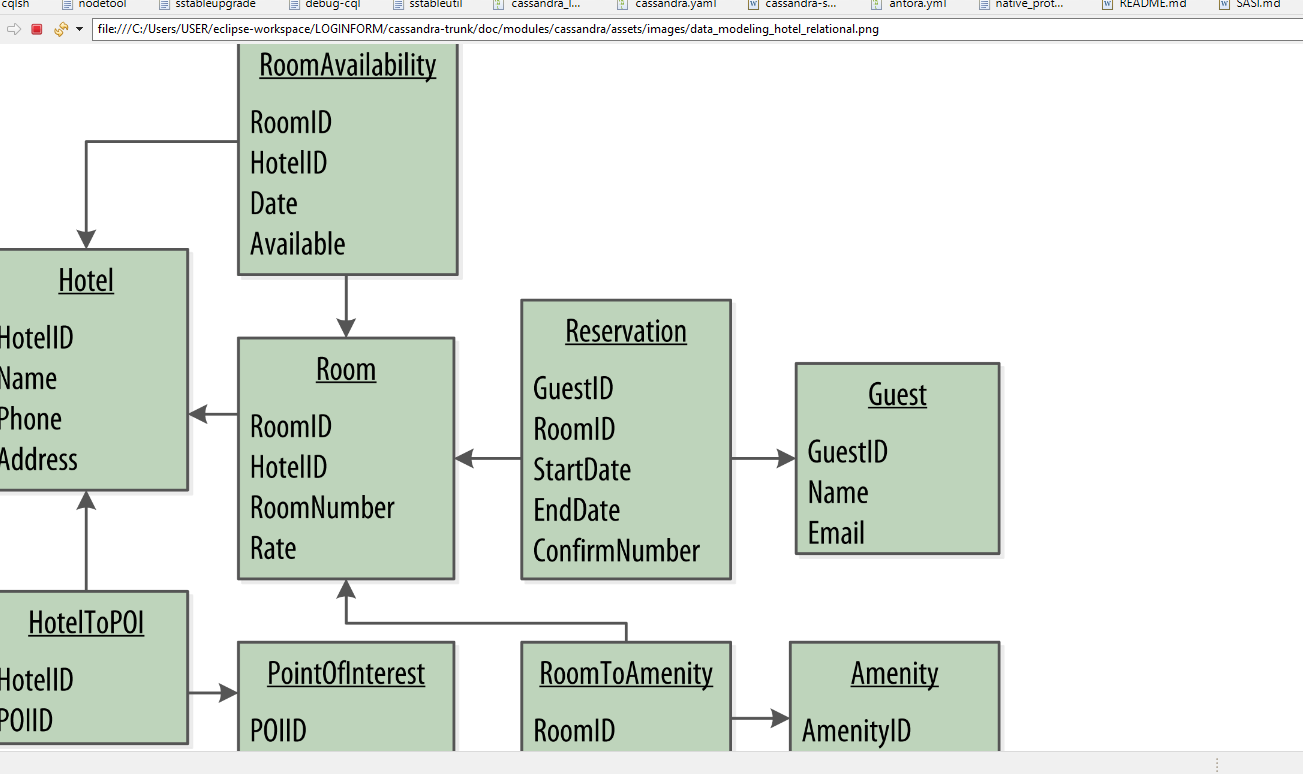
* The file encompasses 63 lines and 44 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 10 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 3, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of cassandraHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.





30. <https://github.com/apache/jmeter>

# Metrics Details For

# jmeter\src\test\java\com\apache\jmeter\test\demo\jmeter.java'

Parameter Value

========= =====

Project Directory \ Project Name

C:\Users\Yashwanth\_Mujakari\Downloads\

Checkpoint Name AllFiles

File Name jmeter\src\test\java\com\apache\jmeter\test\demo\ jmeter.java

Lines 40\*

Statements 32

Percent Branch Statements 0.0

Method Call Statements 3

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 4.0

Line Number of Most Complex Method 30

|  |  |
| --- | --- |
| Name of Most Complex Method | jmeterHttpServer.start() |
| Maximum Complexity | 2\* |
| Line Number of Deepest Block | 15 |
| Maximum Block Depth | 5 |
| Average Block Depth | 1.0 |
| Average Complexity | 1.00\* |

Most Complex Methods in 1 Class(es): Complexity, Statements, Max Depth, Calls

ExcalHttpServer.ExcalHttpServer() 1\*, 5, 3, 0

ExcalHttpServer.start() 1\*, 6, 4, 5

Block Depth Statements

|  |  |
| --- | --- |
| 0 | 3 |
| 1 | 3 |
| 2 | 2 |
| 3 | 7 |
| 4 | 5 |
| 5 | 6 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9+ | 0 |

**Title**: Analyzing Code Metrics of jmeterHttpServer.java in yashwant Project **Abstract**:

The analysis in this report delves into the code metrics of jmeterHttpServer.java in the yashwant project. These metrics offer valuable insights into the complexity, structure, and maintainability of the code, facilitating a deeper understanding of its quality and pinpointing potential areas for enhancement.

# Introduction:

Software metrics play a crucial role in assessing the quality and sustainability of codebases. This report delves into the metrics of jmeterHttpServer.java within the Yashwant project to glean valuable insights into its traits and pinpoint possible concerns.

# Subject Programs:

The focus of our analysis is on the file named jmeterHttpServer.java, situated within the yashwant project directory. This particular file is a component of the yashwant project, which appears to center around constructing a web proxy utilizing gRPC and jmeter for server-side HTTP operations.

# Tools Description:

The metrics were acquired through a software analysis tool equipped to compute diverse code metrics. This tool examines the source code files, extracting data such as lines of code,

statements, complexity, and comments. Additionally, it offers insights into the code's structure, revealing details such as classes, methods, and block depths.

# Results:

The examination of jmeterHttpServer.java unveiled several significant insights:

* The file encompasses 40 lines and 32 statements, featuring a limited proportion of branch statements.
* It comprises a single class housing two methods, showcasing a straightforward organization.
* On average, each method contains 3 statements, with the start() method presenting the highest level of complexity.
* Both methods exhibit a maximum complexity of 2, indicating minimal complexity.
* The distribution of block depth demonstrates a relatively even structure, with the majority of blocks having a depth of 2 or lower.

In summary, the metrics indicate a well-organized and easily maintainable codebase, characterized by low complexity and distinct methods.

# Conclusion:

In summary, examining the code metrics of jmeterHttpServer.java yields valuable insights into the yashwant project's code characteristics. The evident low complexity and well-defined structure suggest a high degree of maintainability and readability. Consistent metric monitoring and occasional refactoring efforts can additionally bolster the project's quality and longevity.

