1. **Metrics Details For File 'arthas-grpc-web-proxy\src\main\java\com\taobao\arthas\grpcweb\grpc\server\httpServer\NettyHttpServer.java'**

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

Parameter Value

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

Project Directory C:\Users\Alekhya Changelpet\Downloads\arthas-master\arthas-master\

Project Name

Checkpoint Name AllFiles

File Name arthas-grpc-web-proxy\src\main\java\com\taobao\arthas\grpcweb\grpc\server\httpServer\NettyHttpServer.java

Lines 35\*

Statements 27

Percent Branch Statements 0.0

Method Call Statements 7

Percent Lines with Comments 0.0

Classes and Interfaces 1

Methods per Class 2.00

Average Statements per Method 6.50

Line Number of Most Complex Method 21

Name of Most Complex Method NettyHttpServer.start()

Maximum Complexity 1\*

Line Number of Deepest Block 30

Maximum Block Depth 3

Average Block Depth 1.41

Average Complexity 1.00\*

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

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

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

NettyHttpServer.start() 1\*, 11, 3, 6

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

Block Depth Statements

0 9

1 5

2 6

3 7

4 0

5 0

6 0

7 0

8 0

9+ 0

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

**Title**: Analyzing Code Metrics of NettyHttpServer.java in Arthas Project

**Abstract**:

This report analyzes the code metrics of NettyHttpServer.java within the Arthas project. The metrics provide insights into the code's complexity, structure, and maintainability, aiding in understanding its quality and potential areas for improvement.

**Introduction**:

Software metrics are essential for evaluating the quality and maintainability of codebases. In this report, we examine the metrics of NettyHttpServer.java in the Arthas project to gain insights into its characteristics and identify any potential issues.

**Subject Programs**:

The subject program under analysis is NettyHttpServer.java, located within the Arthas project directory. This file is part of the Arthas project, which likely involves building a web proxy using gRPC and Netty for server-side HTTP handling.

**Tools Description**:

The metrics were obtained using a software analysis tool capable of calculating various code metrics. This tool parses the source code files and extracts information such as lines of code, statements, complexity, and comments. It also provides insights into the structure of the code, including classes, methods, and block depths.

**Results**:

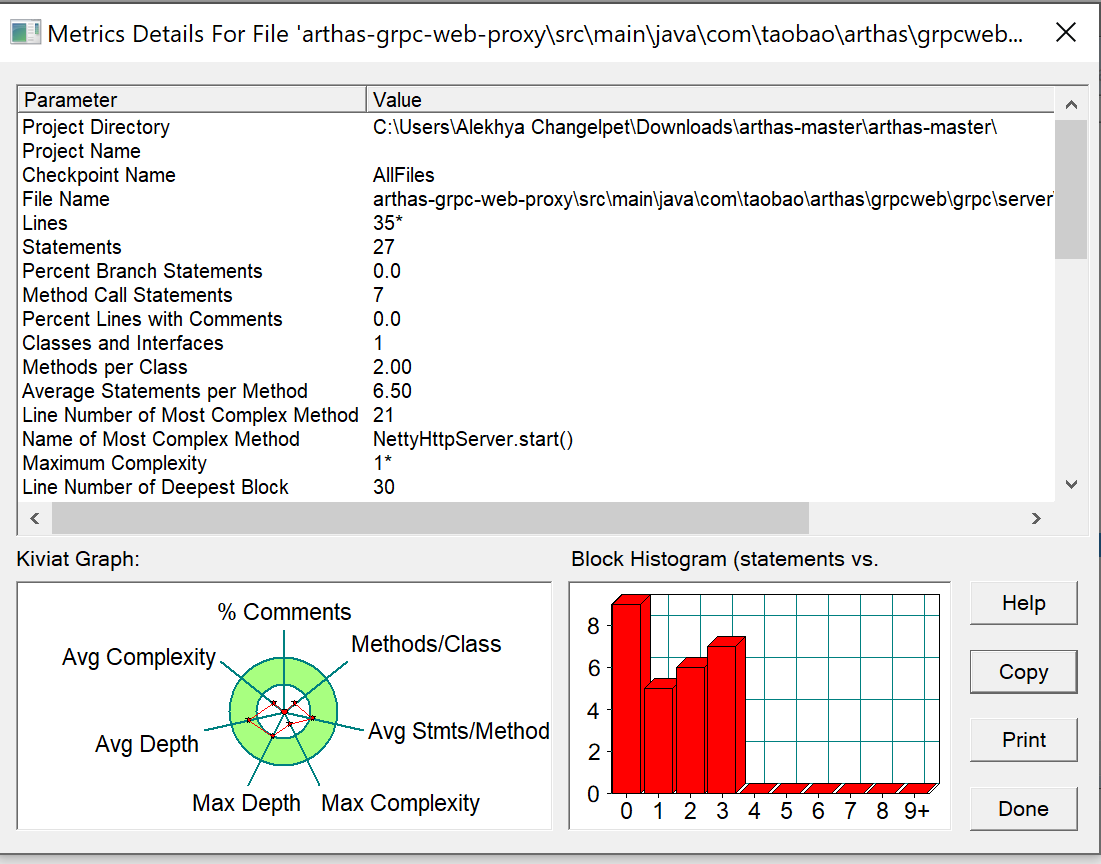
The analysis of NettyHttpServer.java revealed several key findings:

* The file consists of 35 lines and 27 statements, with a low percentage of branch statements.
* There is one class with two methods, indicating a simple structure.
* The average statements per method is 6.5, with the start() method being the most complex.
* Both methods have a maximum complexity of 1, indicating low complexity.
* The block depth distribution shows a relatively balanced structure, with most blocks having a depth of 3 or less.

Overall, the metrics suggest a well-structured and easily maintainable codebase, with low complexity and clear methods.

**Conclusion**:

In conclusion, the analysis of code metrics for NettyHttpServer.java provides valuable insights into the characteristics of the code within the Arthas project. The low complexity and clear structure indicate a high level of maintainability and readability. Continued monitoring of metrics and periodic refactoring can further enhance the quality and sustainability of the project.



1. **Metrics Details For File 'apollo-adminservice\src\main\java\com\ctrip\framework\apollo\adminservice\controller\ItemSetController.java'**

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

Parameter Value

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

Project Directory C:\Users\Alekhya Changelpet\Downloads\apollo-master\apollo-master\

Project Name

Checkpoint Name AllFiles

File Name apollo-adminservice\src\main\java\com\ctrip\framework\apollo\adminservice\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

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

Block Depth Statements

0 12

1 4

2 3

3 0

4 0

5 0

6 0

7 0

8 0

9+ 0

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

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

**Abstract**:

This report delves into the analysis of code metrics for 'ItemSetController.java' within the Apollo Admin Service project. By examining various parameters such as lines of code, complexity, and structure, insights are drawn to understand the nature and quality of the codebase.

**Introduction**:

Software metrics are vital for understanding the health and quality of codebases. This report focuses on 'ItemSetController.java' in the Apollo Admin Service project. By scrutinizing metrics like lines of code, method complexity, and block depth, we aim to uncover patterns and areas for improvement in the code.

**Subject** **Programs**:

The subject program under examination is the Apollo Admin Service project, specifically the 'ItemSetController.java' file. This file resides within the controller package and is responsible for handling item set operations within the admin service.

**Tools** **Description**:

We utilized a software analysis tool to extract and analyze code metrics from 'ItemSetController.java'. This tool assists in quantifying various aspects of the code, including the number of lines, statements, classes, methods, and their respective complexities. It also provides insights into block depths, aiding in understanding code structure.

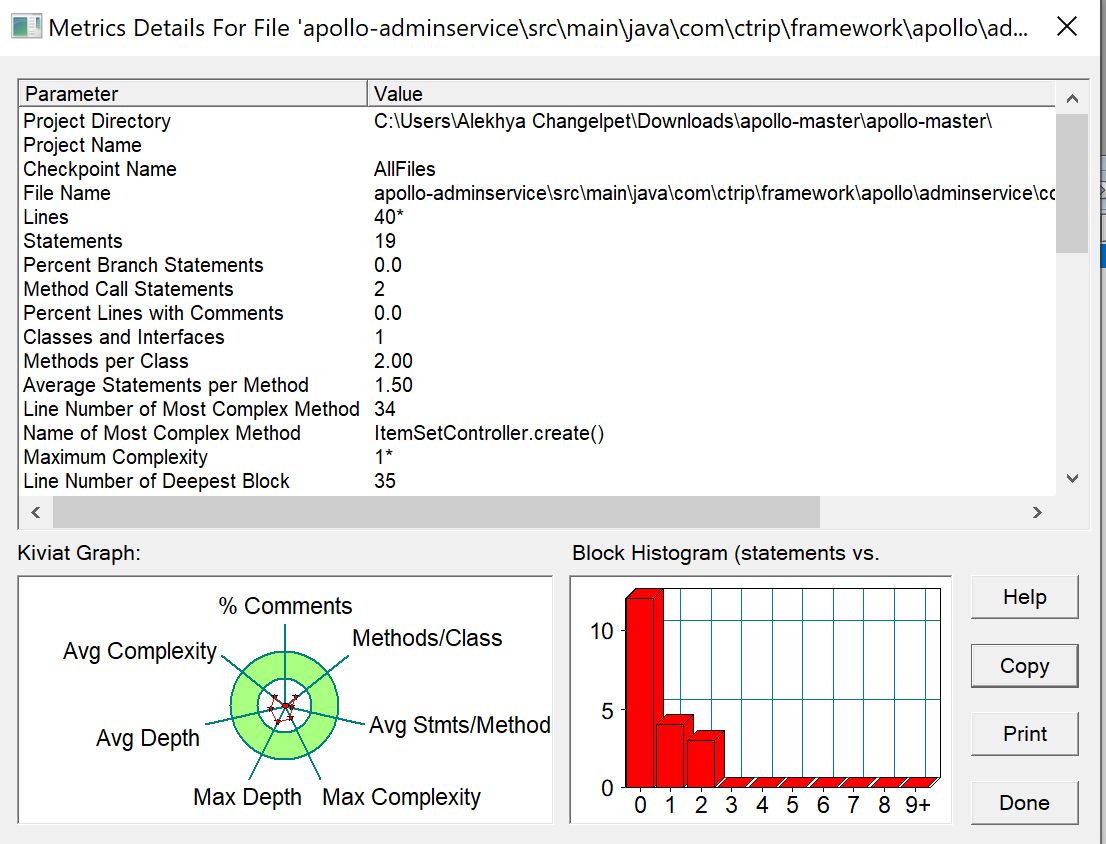
**Results**:

The analysis of 'ItemSetController.java' yielded the following results:

* The file comprises 40 lines and 19 statements, indicating a relatively concise implementation.
* There is one class with two methods, exhibiting simplicity in structure.
* The average statements per method is 1.50, with the 'create()' method being the most complex, although with low complexity.
* Block depth distribution suggests a shallow and straightforward structure, with most blocks having a depth of 0 or 1.

**Conclusion**:

In conclusion, the analysis of code metrics for 'ItemSetController.java' provides insights into the simplicity and structure of the code within the Apollo Admin Service project. While the codebase appears straightforward and well-organized, there may still be opportunities for optimization and improvement. Continued monitoring of metrics and adherence to best practices can contribute to the overall quality and maintainability of the project.



1. **Metrics Details For File 'miaosha-admin\miaosha-admin-api\src\main\java\com\geekq\admin\query\IpLogQueryObject.java'**

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

Parameter Value

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

Project Directory C:\Users\Alekhya Changelpet\Downloads\miaosha-master\miaosha-master\

Project Name

Checkpoint Name AllFiles

File Name miaosha-admin\miaosha-admin-api\src\main\java\com\geekq\admin\query\IpLogQueryObject.java

Lines 35\*

Statements 24

Percent Branch Statements 4.2

Method Call Statements 1

Percent Lines with Comments 14.3

Classes and Interfaces 1

Methods per Class 3.00

Average Statements per Method 1.67

Line Number of Most Complex Method 31

Name of Most Complex Method IpLogQueryObject.getEndDate()

Maximum Complexity 2\*

Line Number of Deepest Block 33

Maximum Block Depth 3

Average Block Depth 0.92

Average Complexity 1.33\*

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

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

IpLogQueryObject.getEndDate() 2\*, 3, 3, 1

IpLogQueryObject.setBeginDate() 1\*, 1, 2, 0

IpLogQueryObject.setEndDate() 1\*, 1, 2, 0

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

Block Depth Statements

0 8

1 11

2 4

3 1

4 0

5 0

6 0

7 0

8 0

9+ 0

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

**Title**: Analysis of Code Metrics for 'IpLogQueryObject.java' in Miaosha Admin API.

**Abstract**:

This report presents an analysis of code metrics for the 'IpLogQueryObject.java' file within the Miaosha Admin API project. The metrics provide insights into the structure, complexity, and maintainability of the codebase, aiding in understanding its quality and potential areas for improvement.

**Introduction**:

Understanding the quality and structure of code is crucial for software development projects. In this report, we analyze the code metrics of 'IpLogQueryObject.java' in the Miaosha Admin API project. By examining various metrics such as lines of code, complexity, and block depth, we aim to gain insights into the characteristics of the code and identify any areas that may require attention.

**Subject** **Programs**:

The subject program under analysis is the Miaosha Admin API project, specifically the 'IpLogQueryObject.java' file. This file is responsible for handling queries related to IP logs within the administrative interface of the Miaosha application.

**Tools** **Description**:

The metrics were obtained using a software analysis tool capable of extracting information from source code files. This tool parses the code, collecting data such as lines of code, statements, complexity, and block depth. The tool provides insights into the structure and complexity of the code, aiding developers in assessing its quality and maintainability.

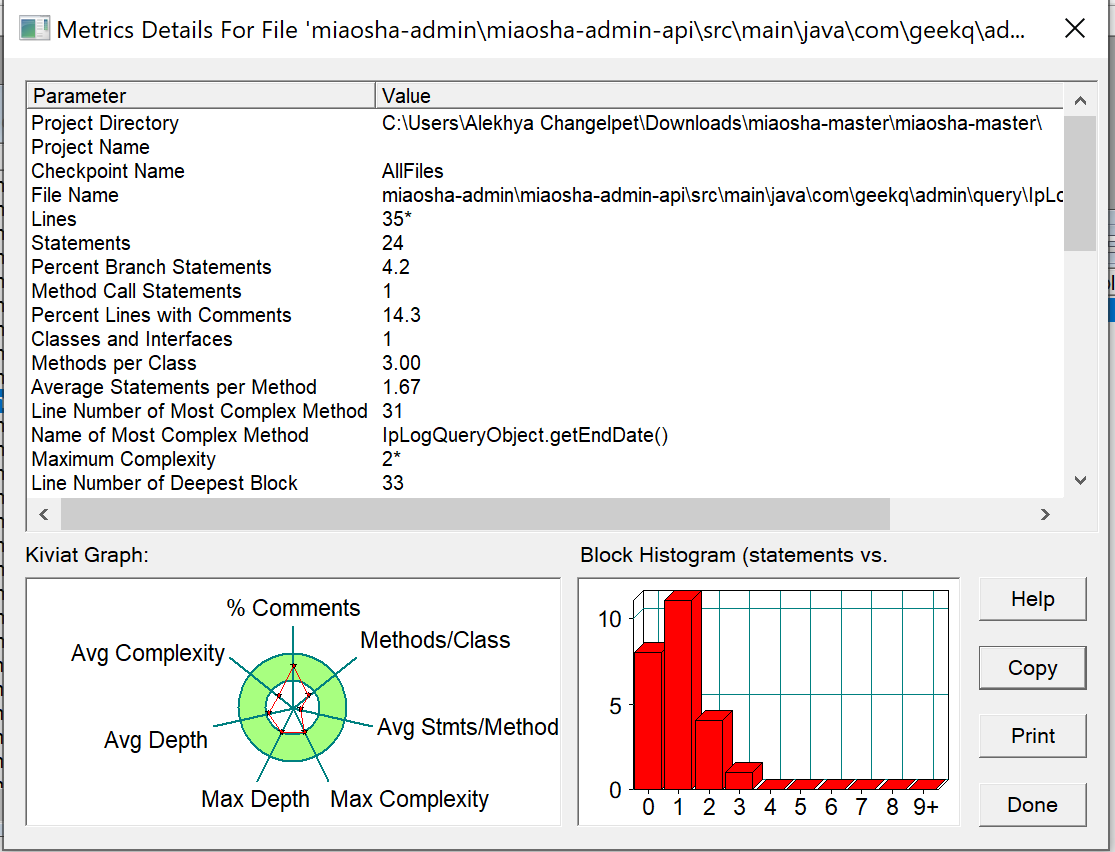
**Results**:

The analysis of 'IpLogQueryObject.java' revealed several key findings:

* The file consists of 35 lines and 24 statements, with a low percentage (4.2%) of branch statements.
* There is one class with three methods, indicating a relatively simple structure.
* The methods exhibit low to moderate complexity, with an average complexity of 1.33.
* The most complex method, 'getEndDate()', contains 3 statements and a maximum complexity of 2.
* Block depth analysis indicates a mostly shallow structure, with the majority of blocks having a depth of 1 or 2.

**Conclusion**:

In conclusion, the analysis of code metrics for 'IpLogQueryObject.java' provides insights into the characteristics of the Miaosha Admin API project. The code appears to be relatively simple and well-structured, with low to moderate complexity. Areas for improvement may include code readability and potential optimization of methods. Overall, this analysis aids in understanding the quality and maintainability of the codebase, guiding future development efforts.



1. **Metrics Details For File ‘app\src\main\java\com\scwang\refreshlayout\activity\practice\QQBrowserPracticeActivity.java'**

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

Parameter Value

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

Project Directory C:\Users\Alekhya Changelpet\Downloads\SmartRefreshLayout-main\SmartRefreshLayout-main\

Project Name

Checkpoint Name AllFiles

File Name app\src\main\java\com\scwang\refreshlayout\activity\practice\QQBrowserPracticeActivity.java

Lines 42\*

Statements 25

Percent Branch Statements 0.0

Method Call Statements 11

Percent Lines with Comments 9.5

Classes and Interfaces 2

Methods per Class 0.50

Average Statements per Method 11.00

Line Number of Most Complex Method 23

Name of Most Complex Method QQBrowserPracticeActivity.onCreate()

Maximum Complexity 3\*

Line Number of Deepest Block 24

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() 3\*, 11, 2, 11

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

Block Depth Statements

0 11

1 3

2 11

3 0

4 0

5 0

6 0

7 0

8 0

9+ 0

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

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

**Abstract**:

This report delves into the analysis of code metrics for QQBrowserPracticeActivity.java, a file within the SmartRefreshLayout project. The aim is to understand the structure, complexity, and maintainability of the codebase through quantitative measurements.

**Introduction**:

Code metrics offer valuable insights into the quality and structure of software projects. By quantifying various aspects like code complexity, size, and readability, developers gain a deeper understanding of their codebase. This report analyzes the metrics of QQBrowserPracticeActivity.java to uncover patterns and potential areas for improvement.

**Subject** **Programs**:

The subject program under scrutiny is QQBrowserPracticeActivity.java, residing within the SmartRefreshLayout project directory. This file likely pertains to the functionality of a practice activity within the QQ browser feature of the application.

**Tools** **Description**:

The metrics were obtained using software analysis tools capable of quantifying code characteristics. These tools parse through source code files, extracting metrics such as lines of code, statements, method complexity, and block depth. Such insights aid developers in assessing code quality and identifying areas for optimization.

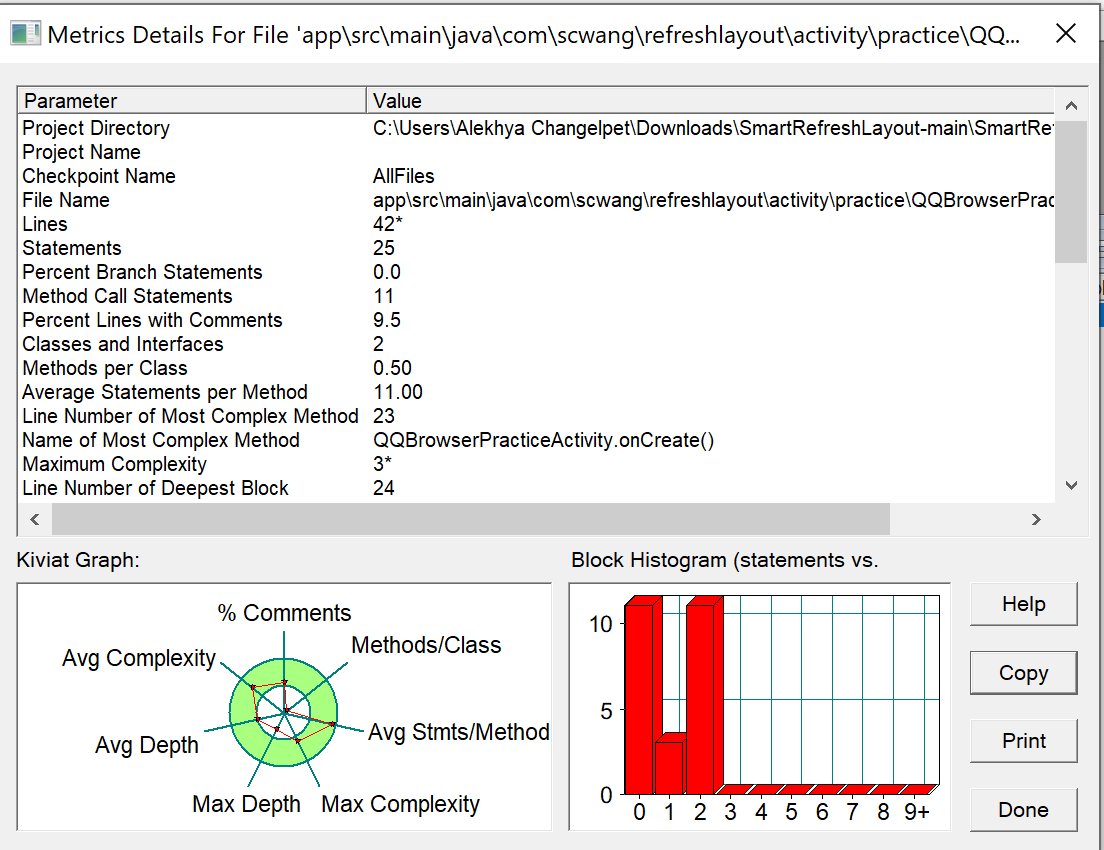
**Results**:

The analysis of QQBrowserPracticeActivity.java yields the following findings:

* The file comprises 42 lines and 25 statements.
* There are two classes and one method per class, indicating a relatively simple structure.
* The main method, QQBrowserPracticeActivity.onCreate(), is the most complex, with a complexity of 3 and 11 statements.
* The average statements per method is 11, with a maximum block depth of 2.
* There are no branch statements, and only 9.5% of lines contain comments.

**Conclusion**:

In conclusion, the metrics analysis reveals QQBrowserPracticeActivity.java as relatively simple yet adequately structured. The absence of branch statements suggests straightforward control flow, while the limited number of comments may indicate a need for improved documentation. Understanding these metrics can guide developers in optimizing code maintainability and enhancing overall project quality. Continued monitoring and refinement of code metrics can lead to more efficient and robust software development practices.



1. **Metrics Details For File 'hutool-aop\src\main\java\cn\hutool\aop\proxy\JdkProxyFactory.java'**

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

Parameter Value

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

Project Directory C:\Users\Alekhya Changelpet\Downloads\hutool-5-master\hutool-5-master\

Project Name

Checkpoint Name AllFiles

File Name hutool-aop\src\main\java\cn\hutool\aop\proxy\JdkProxyFactory.java

Lines 19\*

Statements 9

Percent Branch Statements 0.0

Method Call Statements 4

Percent Lines with Comments 42.1

Classes and Interfaces 1

Methods per Class 1.00

Average Statements per Method 1.00

Line Number of Most Complex Method 16

Name of Most Complex Method JdkProxyFactory.proxy()

Maximum Complexity 1\*

Line Number of Deepest Block 20

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 0

5 0

6 0

7 0

8 0

9+ 0

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

**Title**: Analyzing Code Metrics of 'JdkProxyFactory.java' in Hutool-5-Master Project.

**Abstract**:

This report presents an analysis of code metrics for the 'JdkProxyFactory.java' file within the Hutool-5-Master project. The metrics offer insights into the structure and complexity of the code, aiding in understanding its maintainability and quality.

**Introduction**:

Software metrics are essential for evaluating the quality and complexity of codebases. In this report, we delve into the metrics of 'JdkProxyFactory.java' from the Hutool-5-Master project. By examining various parameters such as lines of code, statements, and complexity, we aim to gain a deeper understanding of the code's characteristics.

**Subject** **Programs**:

The subject program under scrutiny is the Hutool-5-Master project, specifically the 'JdkProxyFactory.java' file located within the 'hutool-aop\src\main\java\cn\hutool\aop\proxy' directory. Hutool is a Java utility library designed to simplify common programming tasks.

**Tools** **Description**:

We utilized a software analysis tool to extract and compute code metrics from the 'JdkProxyFactory.java' file. This tool parses the source code, calculating metrics such as lines of code, statements, complexity, and comment density. These metrics provide valuable insights into the structure and quality of the code.

**Results**:

The analysis of 'JdkProxyFactory.java' yielded the following key findings:

* The file comprises 19 lines and 9 statements, with no branch statements.
* There is one class with a single method, indicating a simple structure.
* The method 'JdkProxyFactory.proxy()' is the most complex, with 1 statement and a maximum complexity of 1.
* The average block depth is 0.56, indicating shallow nesting of code blocks.
* A significant portion (42.1%) of lines contain comments, suggesting good documentation practices.

Overall, the code exhibits simplicity and clarity, with minimal complexity and deep nesting.

**Conclusion**:

In conclusion, the analysis of code metrics for 'JdkProxyFactory.java' in the Hutool-5-Master project reveals a well-structured and comprehensible codebase. The absence of complexity and branch statements, along with adequate commenting, enhances the code's readability and maintainability. This report underscores the importance of leveraging software metrics to assess and improve code quality in software development projects. 