

CSE 566 Software Project Process and Quality Management

ASSIGNMENT 5

Kedar Sai Nadh Reddy Kanchi – 1225297164 – kkachi@asu.edu

Part 1

Source Code

For this assignment I have chosen Database Management System that developed during my bachelor's as part of the CSE 207 Object Oriented Programming in Java course.

Overview of Java Database Management System (DBMS)

The Java Database Management System (DBMS) is a sophisticated software application engineered to optimize database interactions specifically for managing user data and roles through a MySQL database interface. Developed using Java programming language and leveraging the robust Java Database Connectivity (JDBC).

Code Functionalities

The Java DBMS is equipped with an array of functionalities that significantly enhance user and role management within organizations. At its core, our system supports comprehensive CRUD (Create, Read, Update, Delete) operations for user management. This functionality allows us as administrators to efficiently add new users, update existing user details, delete users from the system, and retrieve comprehensive lists of user data. These capabilities are complemented by robust role management features that enable the addition of new roles, the assignment of specific roles to users, and the removal of roles, facilitating dynamic and flexible access control.

Further enhancing our capabilities, our system incorporates advanced search functionality that enables us to quickly find specific users by name or email, streamlining user management and improving operational efficiency. We also implement meticulous transaction management protocols to ensure the integrity and consistency of data. We carefully manage each transaction to handle commits and rollbacks effectively, thereby safeguarding against data corruption and ensuring reliable database operations.

Moreover, our system's error handling and connection management frameworks are meticulously designed to handle SQL exceptions and manage database connections securely. This not only enhances the stability and reliability of the system but also ensures that we adhere to best practices in database security and operations management.

Non-Default Metric

I have chosen the M2-Functional Quality Metrics report to get the non-default metrics. This is because the generation of the M2-Functional Quality Metrics report is highly recommended as it facilitates a detailed evaluation of the software's code quality. This analytical tool focuses on essential aspects such as commenting practices, code structure, complexity, and adherence to organizational standards. By systematically identifying deviations from best practices, the report illuminates areas that could negatively impact the maintainability, scalability, and readability of the software. Such insights are invaluable as they guide developers in implementing targeted refactoring strategies to address specific deficiencies in the codebase. Consequently, leveraging this report can significantly enhance software quality, reduce error rates, and simplify the process of future modifications, thereby providing a robust foundation for ongoing software development and maintenance in an academic or professional setting.

Metric Analysis and Interpretation

The Resource Standard Metrics (RSM) tool was employed on the Java based Database Manager System with an emphasis on Complexity and Functional Quality metrics. The Database Manager System has only one java file.

Complexity Metric Analysis and Interpretation

Based on the complexity metrics report generated for the DatabaseManager.java file, we can draw several insights about the quality and design of the software. Here's a detailed interpretation of these metrics:

Overview of Code Metrics

Summary

- Total Lines of Code (LOC): 239
- Effective Lines of Code (eLOC): 207
- Logical Lines of Code (ILOC): 163
- Total Lines including comments and blanks: 253.
- Function Count: 13 functions

Interpretation

The DatabaseManager.java program contains 239 lines of code, indicating a moderate size that should be manageable for a single developer. The effective(eLOC) and logical lines of code are 207 and 163 respectively, which shows a good ratio of lines that directly contribute to functionality versus overall code size. This density suggests that the code is relatively straightforward and not overly padded with inefficiencies or redundancies. The presence of 13 functions indicates that the program has distributed its functionality across multiple, smaller blocks of code, which is beneficial for isolation of responsibilities and easier debugging.

Cyclomatic Complexity Analysis

Summary

- Average Cyclomatic Complexity: 2.31
- Maximum Cyclomatic Complexity: 13 (in `DatabaseManager.main`)

Interpretation

The average cyclomatic complexity[1] of 2.31 across the program is indicative of a straightforward logic with a simple control flow with few branching paths, which simplifies understanding and maintenance. The complexity peaks at 13 in the main function, suggesting a concentration of decision-making in this area. This could be a concern for potential errors and may require careful testing and possible refactoring to ensure reliability and maintainability. Lowering this complexity in main could distribute decision points more evenly across other functions or modularize the code better.

Function-Specific Metrics

Summary

- `DatabaseManager.main`: Highest complexity at 13, LOC 58
- Other Functions: Complexity generally 1 or 2, LOC ranges from 10 to 15

The software comprises 13 functions, reflecting a potentially well-structured approach to breaking down the program's functionality.

Interpretation

The main function stands out for its higher complexity and longer code length, which could indicate multiple responsibilities or a central control flow. This suggests that the function might be managing several core operations and could benefit from decomposing into smaller, more focused methods. Whereas the utility functions like the `getConnection`, `executeUpdate` etc, maintain low complexity and shorter lengths pointing to specific, single-

purpose roles which is ideal for unit testing and maintenance. However, the sparse comments across these functions could make it difficult to quickly grasp their intended functionalities and edge cases.

Code Maintainability and Scalability

Summary

- Comment Lines: Only 5 lines throughout the code.
- Comment to Code Ratio: Very Low
- Interface Complexity: Average of 1.85

Interpretation

The scant number of comments in the entire codebase could either indicate that the code is straightforward enough not to require extensive documentation or that there is an area for improvement in terms of code documentation. But still, the low comment-to-code ratio raises concerns regarding the future maintainability of the code. Without sufficient documentation, scaling the software or onboarding new developers could be challenging. Interface complexity being moderate suggests that the method interfaces are not overly complicated, which is positive, but there's room for improvement in documenting these interfaces especially by enhancing comments, particularly in complex sections like the main function, could aid future developers in understanding the business logic faster and facilitate smoother updates and maintenance.

Conclusion of the Complexity Metric Analysis and Interpretation

The use of the Resource Standard Metrics (RSM) tool has provided valuable insights into the Database Management System. The analysis revealed that the DatabaseManager.java program shows a promising codebase structure with generally low complexity across most functions and balanced code distribution, facilitating easier testing and maintenance. However, the significant complexity in the main function, combined with a lack of sufficient documentation, poses risks to maintainability and understandability. Addressing these issues through targeted refactoring and improved documentation can enhance the overall quality and sustainability of the software.

Non-Default Metric Analysis and Interpretation

The report generated by the Resource Standard Metrics tool provides insights into several aspects of the code quality for the DatabaseManager.java. The key findings focus on function and class metrics, complexity, and coding style issues. Below, I interpret these findings in terms of their implications for the quality and design of the code:

1. **Commenting Practices: Insufficient Commenting:** The code lacks adequate comments both at the class level and across various function blocks. Specifically, no preceding comment is present for the class and

multiple functions, which could hinder understandability and maintainability. Most functions lack preceding comments (17.33% quality notices). The overall comments in the code only account for a small fraction of the total lines i.e. 2.1%, which is significantly below the 10% threshold suggesting that the code might not be well documented. This can make the code harder to understand and maintain, especially for new developers or when modifications are needed.

2. **Code Structure: Excessive Line Lengths:** There are numerous instances where the length of the lines exceeds the standard terminal width of 80 characters (22.67% quality notices). Long lines can make the code less readable and complicate code reviews and edits in environments that do not support line wrapping.
3. **Blank Line Usage: Sparse Use of Blank Lines:** The report highlights an underuse of blank lines both within functions and classes (17.33% quality notices). The low percentage of blank lines (less than 10%) might suggest dense coding practices that could affect the readability and ease of navigation through the code.
4. **Exception Handling: Heavy Use of Try-Catch Blocks:** The frequent use of try-catch for exception handling is noted as a potential issue (16.00% quality notices), suggesting that it may be used as a control structure which can lead to "spaghetti code" which makes the logic harder to follow and could lead to hidden bugs.
5. **Function and Class Metrics: High Function Count per File:** With 13 functions defined in a single file, the DatabaseManager class might be doing too much, potentially violating the Single Responsibility Principle.

To address the challenges of maintaining and scaling the `DatabaseManager.java` codebase, it is recommended to enhance the commenting practices by providing comprehensive documentation before each class and function. Refactoring the code to maintain standard line lengths and organizing content with appropriate use of blank lines will further improve readability and structure. Additionally, scrutinizing the use of exception handling to ensure it aligns with best practices for error conditions will help avoid overly complex and convoluted code. Implementing these changes will significantly improve the readability and maintainability of the code, thus enhancing the overall software quality.

Part 2

Introduction to Code Portability

Code portability[2] refers to the ability of software to run across different environments, such as operating systems, hardware platforms, or devices, without requiring significant modifications. This characteristic is crucial[3] because it enhances the flexibility and scalability of software, enabling developers to reach a broader audience without needing to rewrite or heavily adapt their code for each new platform. Portability is particularly important in today's diverse technological landscape, where applications might need to operate seamlessly on desktops, mobile devices, and cloud-based systems. By designing software with portability in mind, developers can reduce development time and costs, improve software quality, and simplify maintenance and updates, making their applications more competitive and adaptable to evolving technologies.

Literature Review of Metrics for Code Portability

Assessing the portability of code is crucial for developing software that can be easily deployed and executed across a variety of hardware and software platforms. However, quantifying code portability is not a trivial task, as there are no universally accepted standards or metrics. Researchers have proposed several techniques that focus on different aspects of code portability. The main approaches fall into a few broad categories:

1. Measuring the characteristics of the code itself, such as the percentage of platform-independent code, adherence to coding standards, and modular design.
2. Evaluating the performance of the portable code compared to optimized implementations or theoretical performance ceilings.
3. Calculating composite metrics that combine measures of both code characteristics and performance across multiple target platforms.

Basing on this, the key techniques that can be used for measuring code portability are:

1. **Portability Metric:** Although this metric is critical, there isn't a single universal formula to calculate this. Instead, it is about the proportion of these dependencies in comparison to the entire codebase. Therefore, the portability metric measures the percentage of platform-independent code compared to the total code in the application. It is calculated by taking the amount of code that is specific to a particular platform or environment and dividing it by the total amount of code. A higher percentage of platform-independent code indicates better overall portability of the codebase. Overall, we can classify that the degree of the

platform-dependent code is inversely proportional to the portability of the code. Platform dependencies is typically assessed by:

- a. Conducting **static code analysis** using various tools in the market to identify and fix defects, faults, and other issues in the portable code helps ensure it functions properly across different platforms. This includes checking for things like memory leaks, race conditions, and undefined behaviors that could impact portability.
- b. **Interdependency Measurement:** Quantifying the number of interdependencies, or code elements that rely on each other to function properly, can indicate the modularity and reusability of the portable code. Fewer interdependencies generally mean more portable, maintainable code. Static code analysis can be used to identify these relationships between code components. However, the specific details of what counts as "platform specific" can vary widely depending on the context and needs to be defined based on the platforms considered.

2. **Comparing Performance to Optimized Implementations:** This technique^[6] involves measuring the performance of the portable version of the code and comparing it to a well-recognized, highly optimized implementation of the same application on each target platform. The portable code's performance is then expressed as a fraction or percentage of the optimized implementation's performance. This provides a way to assess the relative performance portability compared to the best-known versions.

- a. The portable code's performance can be increased by following some strict practices like by following some strict **language standard compliance** which evaluates how closely the code adheres to the standard specifications of the programming language used. This can be done by using multiple **compilers** and **setting the warning levels as high** as possible helps ensure the portable code adheres to coding standards and can be compiled successfully across different platforms. Identifying and fixing any compiler warnings helps improve the overall portability of the codebase.

3. **Roofline Performance Model:** The roofline performance model^[6] establishes theoretical performance ceilings or "roofs" for an application based on factors like memory bandwidth and instruction-level parallelism. By analyzing where the portable code's actual performance falls relative to these roofs, you can identify the key hardware characteristics that are limiting its performance portability. The roofline model plots performance as a function of the application's arithmetic intensity (FLOPs per byte of memory access).

Relation of General Code Portability Metrics to the Resource Standard Metrics (RSM) tool

The Resource Standard Metrics (RSM) tool is designed to analyse the compiled source code across the following languages: C, C++, C# and Java. The RSM metrics and analysis capabilities cover a wide range of code elements, including function-level, class-level, and namespace/package-level metrics, which are applicable across the supported programming languages. Therefore, if not directly, the Resource Standard Metrics (RSM) tool[5] offers various metrics[4] like Cyclomatic Complexity, Lines of Code, and Comment Weight etc which indirectly relate to code portability.

1. **Portability Metric:** This metric focuses on quantifying the proportion of platform-independent code within the codebase. The Macro LOC metric and the White Space Percentage Metric of the RSM tool can inform this analysis.
 - a. The size and complexity of macros, as measured by the Macro LOC Metric, can indicate the degree of platform-specific code within the system. Larger, more complex macros are more likely to contain platform-dependent functionality, negatively impacting portability.
 - b. Whereas the White Space Percentage Metric measures the amount of whitespace in the code, can be an indirect indicator of code readability and maintainability. A higher white space percentage may suggest a more modular, reusable codebase, which is generally more portable across platforms.
2. **Comparing Performance to Optimized Implementations:** The second portability measure outlined in the prompt involves comparing the performance of the portable code to highly optimized implementations on each target platform. The RSM tool's metrics that can inform this analysis include:
 - a. **Function Complexity Metric:** The RSM Function Complexity Metric, which combines cyclomatic complexity and interface complexity, can highlight areas of the code that may be less performant and require further optimization. Lower complexity may indicate simpler control structures, which are typically easier to adapt and modify across different platforms. Thus, higher function complexity is often associated with less portable, more tightly coupled code and a lower cyclomatic complexity supports better portability.
 - b. **Comment Line and Comment Percent Metric:** The RSM tool's Comment Line and Comment Percent Metric can provide insights into the readability and maintainability of the code. Well-commented, readable code is often easier to optimize for performance across different platforms.

Conclusion

Effective analysis of code portability necessitates a comprehensive evaluation of aspects such as dependencies on platform-specific features, utilization of external libraries, and adherence to programming language standards. While the RSM tool predominantly evaluates metrics related to complexity and maintainability, these metrics can serve as supplementary indicators in portability assessments by identifying areas of code that are overly complex or extensively documented, which may complicate the porting process. Therefore, by leveraging the insights provided by RSM metrics alongside direct evaluations of portability, developers can formulate more effective strategies for creating software that is easily adaptable, requiring minimal modifications to operate across diverse computing environment and compare its performance to the best-known implementations on each target platform.

References

1. [Code metrics - Cyclomatic complexity.](#)
2. [Software Portability](#)
3. [The importance of Software Portability](#)
4. [Metrics Definitions](#)
5. [What's Inside Your Source Code?](#)
6. [Measuring Performance Portability](#)

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.util.Scanner;
7
8 public class DatabaseManager {
9     private static final String DB_URL = "jdbc:mysql://localhost:3306/mydatabase";
10    private static final String USER = "user";
11    private static final String PASS = "password";
12
13    public static void main(String[] args) {
14        Scanner scanner = new Scanner(System.in);
15        int option;
16        do {
17            System.out.println("\n--- Database Management System ---");
18            System.out.println("1. Add User");
19            System.out.println("2. Update User");
20            System.out.println("3. Delete User");
21            System.out.println("4. View All Users");
22            System.out.println("5. Add Role");
23            System.out.println("6. Assign Role to User");
24            System.out.println("7. Remove Role from User");
25            System.out.println("8. List All Roles");
26            System.out.println("9. Search User by Name");
27            System.out.println("10. Search User by Email");
28            System.out.println("11. Exit");
29            System.out.print("Choose an option: ");
30            option = scanner.nextInt();
31            switch (option) {
32                case 1:
33                    addUser(scanner);
34                    break;
35                case 2:
36                    updateUser(scanner);
37                    break;
38                case 3:
39                    deleteUser(scanner);
40                    break;
41                case 4:
42                    viewUsers();
43                    break;
44                case 5:
45                    addRole(scanner);
46                    break;
47                case 6:
48                    assignRole(scanner);
49                    break;
50                case 7:
51                    removeRole(scanner);
52                    break;
53                case 8:
54                    listRoles();
55                    break;
56                case 9:
```

```
57             searchUserByName(scanner);
58             break;
59         case 10:
60             searchUserByEmail(scanner);
61             break;
62         case 11:
63             System.out.println("Exiting the program...");
64             break;
65         default:
66             System.out.println("Invalid option. Please enter a valid
number.");
67         }
68     } while (option != 11);
69     scanner.close();
70 }
71
72 private static Connection getConnection() throws SQLException {
73     return DriverManager.getConnection(DB_URL, USER, PASS);
74 }
75
76 private static void addUser(Scanner scanner) {
77     System.out.print("Enter user name: ");
78     scanner.nextLine(); // clear buffer
79     String name = scanner.nextLine();
80     System.out.print("Enter email: ");
81     String email = scanner.next();
82     String sql = "INSERT INTO users (name, email) VALUES (?, ?)";
83     try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql)) {
84         conn.setAutoCommit(false);
85         pstmt.setString(1, name);
86         pstmt.setString(2, email);
87         executeUpdate(pstmt, "User added successfully!");
88         conn.commit();
89     } catch (SQLException e) {
90         System.out.println("Error adding user: " + e.getMessage());
91     }
92 }
93
94 private static void updateUser(Scanner scanner) {
95     System.out.print("Enter user id to update: ");
96     int id = scanner.nextInt();
97     scanner.nextLine(); // clear buffer
98     System.out.print("Enter new email: ");
99     String email = scanner.nextLine();
100    String sql = "UPDATE users SET email = ? WHERE id = ?";
101   try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql)) {
102       conn.setAutoCommit(false);
103       pstmt.setString(1, email);
104       pstmt.setInt(2, id);
105       executeUpdate(pstmt, "User updated successfully!");
106       conn.commit();
107   } catch (SQLException e) {
108       System.out.println("Error updating user: " + e.getMessage());
109   }
}
```

```
110 }
111
112 private static void deleteUser(Scanner scanner) {
113     System.out.print("Enter user id to delete: ");
114     int id = scanner.nextInt();
115     String sql = "DELETE FROM users WHERE id = ?";
116     try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql)) {
117         conn.setAutoCommit(false);
118         pstmt.setInt(1, id);
119         executeUpdate(pstmt, "User deleted successfully!");
120         conn.commit();
121     } catch (SQLException e) {
122         System.out.println("Error deleting user: " + e.getMessage());
123     }
124 }
125
126 private static void viewUsers() {
127     String sql = "SELECT * FROM users";
128     try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql); ResultSet rs = pstmt.executeQuery()) {
129         System.out.println("List of all users:");
130         while (rs.next()) {
131             int id = rs.getInt("id");
132             String name = rs.getString("name");
133             String email = rs.getString("email");
134             System.out.printf("ID: %d, Name: %s, Email: %s\n", id, name, email);
135         }
136     } catch (SQLException e) {
137         System.out.println("Error retrieving users: " + e.getMessage());
138     }
139 }
140
141 private static void addRole(Scanner scanner) {
142     System.out.print("Enter role name: ");
143     scanner.nextLine(); // clear buffer
144     String roleName = scanner.nextLine();
145     String sql = "INSERT INTO roles (role_name) VALUES (?)";
146     try (Connection conn = getConnection(); PreparedStatement pstmt =
conn.prepareStatement(sql)) {
147         conn.setAutoCommit(false);
148         pstmt.setString(1, roleName);
149         executeUpdate(pstmt, "Role added successfully!");
150         conn.commit();
151     } catch (SQLException e) {
152         System.out.println("Error adding role: " + e.getMessage());
153     }
154 }
155
156 private static void assignRole(Scanner scanner) {
157     System.out.print("Enter user id for role assignment: ");
158     int userId = scanner.nextInt();
159     System.out.print("Enter role id to assign: ");
160     int roleId = scanner.nextInt();
161     String sql = "INSERT INTO user_roles (user_id, role_id) VALUES (?, ?)";
```

```
162     try (Connection conn = getConnection(); PreparedStatement pstmt =
163          conn.prepareStatement(sql)) {
164         conn.setAutoCommit(false);
165         pstmt.setInt(1, userId);
166         pstmt.setInt(2, roleId);
167         executeUpdate(pstmt, "Role assigned to user successfully!");
168         conn.commit();
169     } catch (SQLException e) {
170         System.out.println("Error assigning role: " + e.getMessage());
171     }
172 }
173
174 private static void removeRole(Scanner scanner) {
175     System.out.print("Enter user id for role removal: ");
176     int userId = scanner.nextInt();
177     System.out.print("Enter role id to remove: ");
178     int roleId = scanner.nextInt();
179     String sql = "DELETE FROM user_roles WHERE user_id = ? AND role_id = ?";
180     try (Connection conn = getConnection(); PreparedStatement pstmt =
181          conn.prepareStatement(sql)) {
182         conn.setAutoCommit(false);
183         pstmt.setInt(1, userId);
184         pstmt.setInt(2, roleId);
185         executeUpdate(pstmt, "Role removed from user successfully!");
186         conn.commit();
187     } catch (SQLException e) {
188         System.out.println("Error removing role: " + e.getMessage());
189     }
190 }
191
192 private static void listRoles() {
193     String sql = "SELECT * FROM roles";
194     try (Connection conn = getConnection(); PreparedStatement pstmt =
195          conn.prepareStatement(sql); ResultSet rs = pstmt.executeQuery()) {
196         System.out.println("List of all roles:");
197         while (rs.next()) {
198             int roleId = rs.getInt("role_id");
199             String roleName = rs.getString("role_name");
200             System.out.printf("Role ID: %d, Role Name: %s\n", roleId, roleName);
201         }
202     } catch (SQLException e) {
203         System.out.println("Error listing roles: " + e.getMessage());
204     }
205 }
206
207 private static void searchUserByName(Scanner scanner) {
208     System.out.print("Enter user name to search: ");
209     scanner.nextLine(); // clear buffer
210     String name = scanner.nextLine();
211     String sql = "SELECT * FROM users WHERE name LIKE ?";
212     try (Connection conn = getConnection(); PreparedStatement pstmt =
213          conn.prepareStatement(sql)) {
214         pstmt.setString(1, "%" + name + "%");
215         try (ResultSet rs = pstmt.executeQuery()) {
216             System.out.println("Search results:");
217             while (rs.next()) {
```

```
214             int id = rs.getInt("id");
215             String email = rs.getString("email");
216             System.out.printf("ID: %d, Name: %s, Email: %s\n", id, name,
217 email);
218         }
219     } catch (SQLException e) {
220         System.out.println("Error searching for user: " + e.getMessage());
221     }
222 }
223
224 private static void searchUserByEmail(Scanner scanner) {
225     System.out.print("Enter email to search for: ");
226     scanner.nextLine(); // clear buffer
227     String email = scanner.nextLine();
228     String sql = "SELECT * FROM users WHERE email LIKE ?";
229     try (Connection conn = getConnection(); PreparedStatement pstmt =
230 conn.prepareStatement(sql)) {
231         pstmt.setString(1, "%" + email + "%");
232         try (ResultSet rs = pstmt.executeQuery()) {
233             System.out.println("Search results:");
234             while (rs.next()) {
235                 int id = rs.getInt("id");
236                 String name = rs.getString("name");
237                 System.out.printf("ID: %d, Name: %s, Email: %s\n", id, name,
238 email);
239             }
240         } catch (SQLException e) {
241             System.out.println("Error searching for email: " + e.getMessage());
242         }
243     }
244     private static void executeUpdate(PreparedStatement pstmt, String
245 successMessage) throws SQLException {
246         int affectedRows = pstmt.executeUpdate();
247         if (affectedRows > 0) {
248             System.out.println(successMessage);
249             System.out.println("Operation was successful.");
250         } else {
251             System.out.println("Operation failed, no changes were made.");
252         }
253     }
254 }
```

Report Banner - Edit rsm.cfg File

Resource Standard Metrics™ for C, C++, C# and Java
Version 7.75 - mSquaredTechnologies.com

License Type: Shareware Evaluation License
Licensed To : Shareware End User - Distribute Freely
License No. : SW1380
Build Date : Sep 2 2009
©1996-2009 M Squared Technologies LLC™

License Date: Dec 05, 1998

Run Date: Apr 22, 2024

License File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.lic
Config. File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.cfg
Command Line: -H -OC:\Users\DELL\M2 RSM Wizard\output\complexity_metrics _report_for_databasemanager_java_code.htm -c -fd -FC:\User s\DELL\M2 RSM Wizard\input\rsm_file_list_for_databasemaneg er_java_code.lst
~~ Function Metrics ~~
~~ Complexity Detail Analysis ~~

File: [C:\Program Files \(x86\)\MSquared\M2 RSM Wizard\example_code\DatabaseManager.java](C:\Program Files (x86)\MSquared\M2 RSM Wizard\example_code\DatabaseManager.java)

Function: DatabaseManager.main

Parameters: (String[] args)

Cyclomatic Complexity Vg Detail

Function Base	:	1			
Loops while / do	:	1			
Selection case	:	11			
Complexity LOC 58	Param 1 eLOC 56	Return 1 ILOC 41	Cyclo Vg 13 Comment 0	Total Lines	15 58

Function: DatabaseManager.getConnection

Parameters: ()

Complexity LOC 3	Param 0 eLOC 2	Return 1 ILOC 1	Cyclo Vg 1 Comment 0	Total Lines	2 3
------------------	----------------	-----------------	----------------------	-------------	-----

Function: DatabaseManager.addUser

Parameters: (Scanner scanner)

Complexity LOC 14	Param 1 eLOC 14	Return 1 ILOC 12	Cyclo Vg 1 Comment 1	Total Lines	3 14
-------------------	-----------------	------------------	----------------------	-------------	------

Function: DatabaseManager.updateUser

Parameters: (Scanner scanner)

Complexity LOC 14	Param 1 eLOC 14	Return 1 ILOC 12	Cyclo Vg 1 Comment 1	Total Lines	3 14
-------------------	-----------------	------------------	----------------------	-------------	------

Function: DatabaseManager.deleteUser

Parameters: (Scanner scanner)

Complexity LOC 10	Param 1 eLOC 10	Return 1 ILOC 8	Cyclo Vg 1 Comment 0	Total Lines	3 10
-------------------	-----------------	-----------------	----------------------	-------------	------

Function: DatabaseManager.viewUsers

Parameters: ()

Cyclomatic Complexity Vg Detail

Function Base	:	1			
Loops while / do	:	1			
Complexity LOC 11	Param 0 eLOC 10	Return 1 ILOC 8	Cyclo Vg 2 Comment 0	Total Lines	3 11

Function: DatabaseManager.addRole

Parameters: (Scanner scanner)

Complexity LOC 11	Param 1 eLOC 11	Return 1 ILOC 9	Cyclo Vg 1 Comment 1	Total Lines	3 11
-------------------	-----------------	-----------------	----------------------	-------------	------

Function: DatabaseManager.assignRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
------------	---------	----------	------------	-------	---

LOC 13	eLOC 13	lLOC 11	Comment 0	Lines	13
--------	---------	---------	-----------	-------	----

Function: DatabaseManager.removeRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 13	eLOC 13	lLOC 11	Comment 0	Lines	13

Function: DatabaseManager.listRoles

Parameters: ()

Cyclomatic Complexity	Vg Detail				
Function Base	:	1			
Loops while / do	:	1			
Complexity	Param 0	Return 1	Cyclo Vg 2	Total	3
LOC 10	eLOC 9	lLOC 7	Comment 0	Lines	10

Function: DatabaseManager.searchUserByName

Parameters: (Scanner scanner)

Cyclomatic Complexity	Vg Detail				
Function Base	:	1			
Loops while / do	:	1			
Complexity	Param 1	Return 1	Cyclo Vg 2	Total	4
LOC 15	eLOC 13	lLOC 10	Comment 1	Lines	15

Function: DatabaseManager.searchUserByEmail

Parameters: (Scanner scanner)

Cyclomatic Complexity	Vg Detail				
Function Base	:	1			
Loops while / do	:	1			
Complexity	Param 1	Return 1	Cyclo Vg 2	Total	4
LOC 15	eLOC 13	lLOC 10	Comment 1	Lines	15

Function: DatabaseManager.executeUpdate

Parameters: (PreparedStatement pstmt, String successMessage)

Cyclomatic Complexity	Vg Detail				
Function Base	:	1			
Conditional if / else if: 1					
Complexity	Param 2	Return 1	Cyclo Vg 2	Total	5
LOC 9	eLOC 7	lLOC 4	Comment 0	Lines	9

~~ Total File Summary ~~

LOC 239	eLOC 207	lLOC 163	Comment 5	Lines	253
---------	----------	----------	-----------	-------	-----

~~ File Functional Summary ~~

File Function Count.....:	13				
Total Function LOC.....:	196	Total Function Pts LOC :	4.5		
Total Function eLOC....:	185	Total Function Pts eLOC:	3.9		
Total Function lLOC....:	144	Total Function Pts lLOC:	3.1		
Total Function Params ..:	11	Total Function Return ..:	13		
Total Cyclo Complexity :	30	Total Function Complex.:	54		
-----	-----	-----	-----	-----	-----
Max Function LOC:	58	Average Function LOC ..:	15.08		
Max Function eLOC:	56	Average Function eLOC ..:	14.23		
Max Function lLOC:	41	Average Function lLOC ..:	11.08		
-----	-----	-----	-----	-----	-----
Max Function Parameters:	2	Avg Function Parameters:	0.85		
Max Function Returns ..:	1	Avg Function Returns ..:	1.00		
Max Interface Complex. :	3	Avg Interface Complex. :	1.85		
Max Cyclomatic Complex.:	13	Avg Cyclomatic Complex.:	2.31		
Max Total Complexity ...:	15	Avg Total Complexity ...:	4.15		

End of File: [C:\Program Files \(x86\)\MSquared\M2 RSM Wizard\example code\DatabaseManager\DatabaseManager.java](C:\Program Files (x86)\MSquared\M2 RSM Wizard\example code\DatabaseManager\DatabaseManager.java)

~~ Total Metrics For 1 Files ~~

~~ Project Directory Summary ~~

Files 1 - C:\Program Files (x86)\MSquared\M2 RSM Wizard\example code\DatabaseManager					
LOC 239	eLOC 207	lLOC 163	Comment 5	Lines	253

~~ Total Project Summary ~~

LOC 239	eLOC 207	lLOC 163	Comment 5	Lines	253
Average per File, metric/1 files					
LOC 239	eLOC 207	lLOC 163	Comment 5	Lines	253

~~ Project Functional Metrics ~~

Function: DatabaseManager.main

Parameters: (String[] args)

Complexity	Param 1	Return 1	Cyclo Vg 13	Total	15
LOC 58	eLOC 56	lLOC 41	Comment 0	Lines	58

Function: DatabaseManager.getConnection

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 1	Total	2
LOC 3	eLOC 2	lLOC 1	Comment 0	Lines	3

Function: DatabaseManager.addUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 14	eLOC 14	lLOC 12	Comment 1	Lines	14

Function: DatabaseManager.updateUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 14	eLOC 14	lLOC 12	Comment 1	Lines	14

Function: DatabaseManager.deleteUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 10	eLOC 10	lLOC 8	Comment 0	Lines	10

Function: DatabaseManager.viewUsers

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 2	Total	3
LOC 11	eLOC 10	lLOC 8	Comment 0	Lines	11

Function: DatabaseManager.addRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 11	eLOC 11	lLOC 9	Comment 1	Lines	11

Function: DatabaseManager.assignRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 13	eLOC 13	lLOC 11	Comment 0	Lines	13

Function: DatabaseManager.removeRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 13	eLOC 13	lLOC 11	Comment 0	Lines	13

Function: DatabaseManager.listRoles

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 2	Total	3
LOC 10	eLOC 9	lLOC 7	Comment 0	Lines	10

Function: DatabaseManager.searchUserByName

Parameters: (Scanner scanner)

Complexity LOC 15	Param 1 eLOC 13	Return 1 1LOC 10	Cyclo Vg 2 Comment 1	Total Lines	4 15
----------------------	--------------------	---------------------	-------------------------	-------------	---------

Function: DatabaseManager.searchUserByEmail

Parameters: (Scanner scanner)

Complexity LOC 15	Param 1 eLOC 13	Return 1 1LOC 10	Cyclo Vg 2 Comment 1	Total Lines	4 15
----------------------	--------------------	---------------------	-------------------------	-------------	---------

Function: DatabaseManager.executeUpdate

Parameters: (PreparedStatement pstmt, String successMessage)

Complexity LOC 9	Param 2 eLOC 7	Return 1 1LOC 4	Cyclo Vg 2 Comment 0	Total Lines	5 9
---------------------	-------------------	--------------------	-------------------------	-------------	--------

Total: Functions

LOC 196	eLOC 185	1LOC 144	InCmp 24	CycloCmp	30
---------	----------	----------	----------	----------	----

Function Points	FP(LOC) 3.7	FP(eLOC) 3.5	FP(1LOC)	2.7
-----------------	-------------	--------------	----------	-----

~~ Project Functional Analysis ~~

Total Functions	13	Total Physical Lines ..:	196
Total LOC	196	Total Function Pts LOC :	3.7
Total eLOC	185	Total Function Pts eLOC:	3.5
Total 1LOC.....	144	Total Function Pts 1LOC:	2.7
Total Cyclomatic Comp. :	30	Total Interface Comp. ..:	24
Total Parameters	11	Total Return Points:	13
Total Comment Lines	5	Total Blank Lines	0
<hr/>			
Avg Physical Lines ..:	15.08		
Avg LOC	15.08	Avg eLOC	14.23
Avg 1LOC	11.08	Avg Cyclomatic Comp. ...:	2.31
Avg Interface Comp.	1.85	Avg Parameters	0.85
Avg Return Points	1.00	Avg Comment Lines	0.38
<hr/>			
Max LOC	58		
Max eLOC	56	Max 1LOC	41
Max Cyclomatic Comp. ...:	13	Max Interface Comp.:	3
Max Parameters	2	Max Return Points	1
Max Comment Lines	1	Max Total Lines	58
<hr/>			
Min LOC	3		
Min eLOC	2	Min 1LOC	1
Min Cyclomatic Comp. ...:	1	Min Interface Comp.:	1
Min Parameters	0	Min Return Points	1
Min Comment Lines	0	Min Total Lines	3

~~ File Summary ~~

C Source Files *.c	0	C/C++ Include Files *.h:	0
C++ Source Files *.c* ..:	0	C++ Include Files *.h* :	0
C# Source Files *.cs ...:	0	Java Source File *.jav*:	1
Other Source Files	0		
Total File Count	1		

Shareware evaluation licenses process only **20** files.
Paid licenses enable processing for an unlimited number of files.

Report Banner - Edit rsm.cfg File

Report Banner - Edit rsm.cfg File

Resource Standard Metrics™ for C, C++, C# and Java
Version 7.75 - mSquaredTechnologies.com

License Type: Shareware Evaluation License
Licensed To : Shareware End User - Distribute Freely
License No. : SW1380 License Date: Dec 05, 1998
Build Date : Sep 2 2009 Run Date: Apr 22, 2024
©1996-2009 M Squared Technologies LLC™

License File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.lic
Config. File: C:\Program Files (x86)\MSquared\M2 RSM\rsm.cfg
Command Line: -H -OC:\Users\DELL\M2 RSM Wizard\output\functional_quality
_metrics_report_for_databasemanager_java_code.htm -c -fa -
n -o -SFunctional Quality Metrics -FC:\Users\DELL\M2 RSM W
izard\input\rsm_file_list_for_databasemanager_java_code.ls
t
UDQN File : C:\Program Files (x86)\MSquared\M2 RSM\rsm_udqn.cfg

Functional Quality Metrics

~~ Function Metrics ~~
~~ Class/Struct Metrics ~~
~~ Complexity Analysis ~~
~~ Quality Analysis ~~

File: [C:\Program Files \(x86\)\MSquared\M2 RSM Wizard\example_code\DatabaseManager.java](http://C:\Program Files (x86)\MSquared\M2 RSM Wizard\example_code\DatabaseManager.java)

----- Class Begin Line: 8 -----

Class: DatabaseManager

Notice #52: Line 8: A class has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 9: Line character length = 82. This width exceeds the standard terminal width of 80 characters.

----- Function Begin Line: 13 -----

Function: DatabaseManager.main

Parameters: (String[] args)

Notice #51: Line 13: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 66: Line character length = 87. This width exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Notice #28: The function cyclomatic complexity of 13 exceeds the specified limit of 10.

Function: DatabaseManager.main

Complexity	Param 1	Return 1	Cyclo	Vg	13	Total	15
LOC 58	eLOC 56	1LOC 41	Comment 0			Lines	58

----- Function End Line: 70 -----

----- Function Begin Line: 72 -----

Function: DatabaseManager.getConnection

Parameters: ()

Notice #51: Line 72: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #49: The function contains no input parameters or void. Suggest using explicit parameters for interface clarity.

Function: **DatabaseManager.getConnection**

Complexity	Param 0	Return 1	Cyclo	Vg	1	Total	2
LOC 3	eLOC 2	lLOC 1	Comment 0			Lines	3

----- Function End Line: 74 -----

----- Function Begin Line: 76 -----

Function: **DatabaseManager.addUser**

Parameters: **(Scanner scanner)**

Notice #51: Line 76: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 83: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 83: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 6.7% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.addUser**

Complexity	Param 1	Return 1	Cyclo	Vg	1	Total	3
LOC 14	eLOC 14	lLOC 12	Comment 1			Lines	14

----- Function End Line: 89 -----

----- Function Begin Line: 94 -----

Function: **DatabaseManager.updateUser**

Parameters: **(Scanner scanner)**

Notice #51: Line 94: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 101: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 101: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 6.7% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.updateUser**

Complexity	Param 1	Return 1	Cyclo	Vg	1	Total	3
LOC 14	eLOC 14	lLOC 12	Comment 1			Lines	14

----- Function End Line: 107 -----

----- Function Begin Line: 112 -----

Function: **DatabaseManager.deleteUser**

Parameters: **(Scanner scanner)**

Notice #51: Line 112: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 116: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 116: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.deleteUser**

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 10	eLOC 10	lLOC 8	Comment 0	Lines	10

----- Function End Line: 121 -----

----- Function Begin Line: 126 -----

Function: **DatabaseManager.viewUsers**

Parameters: ()

Notice #51: Line 126: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 128: Line character length = 140. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 128: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #1: Line 134: Line character length = 84. This width exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Notice #49: The function contains no input parameters or void. Suggest using explicit parameters for interface clarity.

Function: **DatabaseManager.viewUsers**

Complexity	Param 0	Return 1	Cyclo Vg 2	Total	3
LOC 11	eLOC 10	lLOC 8	Comment 0	Lines	11

----- Function End Line: 136 -----

----- Function Begin Line: 141 -----

Function: **DatabaseManager.addRole**

Parameters: (Scanner scanner)

Notice #51: Line 141: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 146: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 146: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 8.3% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.addRole**

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 11	eLOC 11	1LOC 9	Comment 1	Lines	11

----- Function End Line: 151 -----

----- Function Begin Line: 156 -----
Function: DatabaseManager.assignRole
Parameters: (Scanner scanner)

Notice #51: Line 156: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 162: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 162: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: DatabaseManager.assignRole
Complexity Param 1 Return 1 Cyclo Vg 1 Total 3
LOC 13 eLOC 13 1LOC 11 Comment 0 Lines 13
----- Function End Line: 168 -----

----- Function Begin Line: 173 -----
Function: DatabaseManager.removeRole
Parameters: (Scanner scanner)

Notice #51: Line 173: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 179: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 179: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: DatabaseManager.removeRole
Complexity Param 1 Return 1 Cyclo Vg 1 Total 3
LOC 13 eLOC 13 1LOC 11 Comment 0 Lines 13
----- Function End Line: 185 -----

----- Function Begin Line: 190 -----
Function: DatabaseManager.listRoles
Parameters: ()

Notice #51: Line 190: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 192: Line character length = 140. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 192: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #1: Line 197: Line character length = 84. This width exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Notice #49: The function contains no input parameters or void. Suggest using explicit parameters for interface clarity.

Function: **DatabaseManager.listRoles**

Complexity	Param 0	Return 1	Cyclo	Vg 2	Total	3
LOC 10	eLOC 9	lLOC 7	Comment 0		Lines	10

----- Function End Line: 199 -----

----- Function Begin Line: 204 -----

Function: **DatabaseManager.searchUserByName**

Parameters: **(Scanner scanner)**

Notice #51: Line 204: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 209: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 209: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #38: Line 211: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #1: Line 216: Line character length = 88. This width exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 6.3% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.searchUserByName**

Complexity	Param 1	Return 1	Cyclo	Vg 2	Total	4
LOC 15	eLOC 13	lLOC 10	Comment 1		Lines	15

----- Function End Line: 218 -----

----- Function Begin Line: 224 -----

Function: **DatabaseManager.searchUserByEmail**

Parameters: **(Scanner scanner)**

Notice #51: Line 224: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #1: Line 229: Line character length = 103. This width exceeds the standard terminal width of 80 characters.

Notice #38: Line 229: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed upon the code.

Notice #38: Line 231: Exception handling 'try' - 'catch' has been identified. Exception handling can be a form indirect logic control similar to a goto. The use of exception handling should be driven by the design and not casually imposed

upon the code.

Notice #1: Line 236: Line character length = 88. This width exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 6.3% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.searchUserByEmail**

Complexity	Param 1	Return 1	Cyclo	Vg 2	Total	4
LOC 15	eLOC 13	1LOC 10	Comment 1		Lines	15

----- Function End Line: 238 -----

Notice #1: Line 244: Line character length = 107. This width exceeds the standard terminal width of 80 characters.

----- Function Begin Line: 244 -----

Function: **DatabaseManager.executeUpdate**

Parameters: **(PreparedStatement pstmt, String successMessage)**

Notice #51: Line 244: A function has been identified which does not have a preceding comment. Comments that detail the purpose, algorithms, and parameter/return definitions are suggested.

Notice #17: Function comments, 0.0% are less than 10.0%.

Notice #46: Function blank line percent, 0.0% is less than 10.0%.

Function: **DatabaseManager.executeUpdate**

Complexity	Param 2	Return 1	Cyclo	Vg 2	Total	5
LOC 9	eLOC 7	1LOC 4	Comment 0		Lines	9

----- Function End Line: 252 -----

Notice #31: Class comments, 2.1% are less than 10.0%.

Notice #46: Class blank line percent, 5.3% is less than 10.0%.

Class: **DatabaseManager**

Attributes	Publ 0	Prot 0	Private 3	Total	3
Methods	Publ 1	Prot 0	Private 22	Total	23
LOC 233	eLOC 201	1LOC 157	Comment 5	Lines	245

----- Class End Line: 253 -----

Notice #47: File blank line percentage, 5.9% is less than 10.0%

Notice #20: File comment line percentage, 2.0% is less than 10.0%

~~ Total File Summary ~~

LOC 239	eLOC 207	1LOC 163	Comment 5	Lines	253
---------	----------	----------	-----------	-------	-----

~~ File Functional Summary ~~

File Function Count....:	13		
Total Function LOC....:	196	Total Function Pts LOC :	4.5
Total Function eLOC....:	185	Total Function Pts eLOC:	3.9
Total Function 1LOC....:	144	Total Function Pts 1LOC:	3.1
Total Function Params ..:	11	Total Function Return ..:	13
Total Cyclo Complexity :	30	Total Function Complex.:	54
-----	-----	-----	-----
Max Function LOC :</td <td>58</td> <td>Average Function LOC ..:</td> <td>15.08</td>	58	Average Function LOC ..:	15.08
Max Function eLOC :</td <td>56</td> <td>Average Function eLOC ..:</td> <td>14.23</td>	56	Average Function eLOC ..:	14.23
Max Function 1LOC	41	Average Function 1LOC ..:	11.08
-----	-----	-----	-----
Max Function Parameters:	2	Avg Function Parameters:	0.85
Max Function Returns ...:	1	Avg Function Returns ...:	1.00
Max Interface Complex. :	3	Avg Interface Complex. :	1.85
Max Cyclomatic Complex.:	13	Avg Cyclomatic Complex.:	2.31
Max Total Complexity ...:	15	Avg Total Complexity ...:	4.15

End of File: C:\Program Files (x86)\MSquared\M2 RSM Wizard\example code\DatabaseManager\DatabaseManager.java

~~ Total Metrics For 1 Files ~~

~~ Project Directory Summary ~~

Files 1 - C:\Program Files (x86)\MSquared\M2 RSM Wizard\example code\DatabaseManager					
LOC 239	eLOC 207	1LOC 163	Comment 5	Lines	253

~~ Total Project Summary ~~

LOC 239	eLOC 207	1LOC 163	Comment 5	Lines	253
Average per File, metric/1 files					
LOC 239	eLOC 207	1LOC 163	Comment 5	Lines	253

~~ Project Functional Metrics ~~

Function: DatabaseManager.main

Parameters: (String[] args)

Complexity	Param 1	Return 1	Cyclo Vg 13	Total	15
LOC 58	eLOC 56	1LOC 41	Comment 0	Lines	58

Function: DatabaseManager.getConnection

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 1	Total	2
LOC 3	eLOC 2	1LOC 1	Comment 0	Lines	3

Function: DatabaseManager.addUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 14	eLOC 14	1LOC 12	Comment 1	Lines	14

Function: DatabaseManager.updateUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 14	eLOC 14	1LOC 12	Comment 1	Lines	14

Function: DatabaseManager.deleteUser

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 10	eLOC 10	1LOC 8	Comment 0	Lines	10

Function: DatabaseManager.viewUsers

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 2	Total	3
LOC 11	eLOC 10	1LOC 8	Comment 0	Lines	11

Function: DatabaseManager.addRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 11	eLOC 11	1LOC 9	Comment 1	Lines	11

Function: DatabaseManager.assignRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 13	eLOC 13	1LOC 11	Comment 0	Lines	13

Function: DatabaseManager.removeRole

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 1	Total	3
LOC 13	eLOC 13	1LOC 11	Comment 0	Lines	13

Function: DatabaseManager.listRoles

Parameters: ()

Complexity	Param 0	Return 1	Cyclo Vg 2	Total Lines	3
LOC 10	eLOC 9	1LOC 7	Comment 0		10

Function: DatabaseManager.searchUserByName

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 2	Total Lines	4
LOC 15	eLOC 13	1LOC 10	Comment 1		15

Function: DatabaseManager.searchUserByEmail

Parameters: (Scanner scanner)

Complexity	Param 1	Return 1	Cyclo Vg 2	Total Lines	4
LOC 15	eLOC 13	1LOC 10	Comment 1		15

Function: DatabaseManager.executeUpdate

Parameters: (PreparedStatement pstmt, String successMessage)

Complexity	Param 2	Return 1	Cyclo Vg 2	Total Lines	5
LOC 9	eLOC 7	1LOC 4	Comment 0		9

Total: Functions

LOC 196	eLOC 185	1LOC 144	InCmp 24	CycloCmp	30
---------	----------	----------	----------	----------	----

Function Points	FP(LOC) 3.7	FP(eLOC) 3.5	FP(1LOC)	2.7
-----------------	-------------	--------------	----------	-----

~~ Project Functional Analysis ~~

Total Functions	13	Total Physical Lines ..:	196
Total LOC	196	Total Function Pts LOC :	3.7
Total eLOC	185	Total Function Pts eLOC:	3.5
Total 1LOC.....	144	Total Function Pts 1LOC:	2.7
Total Cyclomatic Comp. :	30	Total Interface Comp. ..:	24
Total Parameters	11	Total Return Points:	13
Total Comment Lines ...:	5	Total Blank Lines	0

Avg Physical Lines ..:	15.08	Avg eLOC	14.23
Avg LOC	15.08	Avg Cyclomatic Comp. ...:	2.31
Avg 1LOC	11.08	Avg Parameters	0.85
Avg Interface Comp.	1.85	Avg Comment Lines	0.38
Avg Return Points	1.00		

Max LOC	58	Max 1LOC	41
Max eLOC	56	Max Interface Comp.:	3
Max Cyclomatic Comp. ...:	13	Max Return Points	1
Max Parameters	2	Max Total Lines	58
Max Comment Lines	1		

Min LOC	3	Min 1LOC	1
Min eLOC	2	Min Interface Comp.:	1
Min Cyclomatic Comp. ...:	1	Min Return Points	1
Min Parameters	0	Min Total Lines	3
Min Comment Lines	0		

~~ Project Class/Struct Metrics ~~**Class:** DatabaseManager

Attributes	Publ 0	Prot 0	Private 3	Total	3
Methods	Publ 1	Prot 0	Private 22	Total	23
Complexity	Param 11	Return 13	Cyclo Vg 30	Total	54
LOC 233	eLOC 201	1LOC 157	Comment 5	Lines	245

Total: All Classes/Structs

Attributes	Publ 0	Prot 0	Private 3	Total	3
Methods	Publ 1	Prot 0	Private 22	Total	23
Complexity	Param 11	Return 13	Cyclo Vg 30	Total	54
LOC 233	eLOC 201	1LOC 157	Comment 5	Lines	245

~~ Project Class/Struct Analysis ~~

Total Classes/Structs ..:	1	Total Methods	23
Total Public Methods ...:	1	Total Public Attributes:	0
Total Protected Methods:	0	Total Protected Attrib.:	0
Total Private Methods ..:	22	Total Private Attrib. ..:	3
Total Physical Lines ...:	245	Total LOC	233
Total eLOC	201	Total lLOC	157
Total Cyclomatic Comp. :	30	Total Interface Comp. ..:	24
Total Parameters	11	Total Return Points:	13
Total Comment Lines:	5	Total Blank Lines	13
<hr/>			
Avg Physical Lines:	245.00	Avg Methods	23.00
Avg Public Methods:	1.00	Avg Public Attributes ..:	0.00
Avg Protected Methods ..:	0.00	Avg Protected Attrib. ..:	0.00
Avg Private Methods:	22.00	Avg Private Attributes :	3.00
Avg LOC	233.00	Avg eLOC	201.00
Avg lLOC	157.00	Avg Cyclomatic Comp. ...:	30.00
Avg Interface Comp.:	24.00	Avg Parameters	11.00
Avg Return Points	13.00	Avg Comment Lines	5.00
<hr/>			
Max Physical Lines:	245	Max Methods	23
Max Public Methods:	1	Max Public Attributes ..:	0
Max Protected Methods ..:	0	Max Protected Attrib. ..:	0
Max Private Methods:	22	Max Private Attributes :	3
Max LOC	233	Max eLOC	201
Max lLOC	157	Max Cyclomatic Comp. ...:	30
Max Interface Comp.:	24	Max Parameters	11
Max Return Points	13	Max Comment Lines	5
<hr/>			
Min Physical Lines:	245	Min Methods	23
Min Public Methods:	1	Min Public Attributes ..:	0
Min Protected Methods ..:	0	Min Protected Attrib. ..:	0
Min Private Methods:	22	Min Private Attributes :	3
Min LOC	233	Min eLOC	201
Min lLOC	157	Min Cyclomatic Comp. ...:	30
Min Interface Comp.:	24	Min Parameters	11
Min Return Points	13	Min Comment Lines	5

~~ Project Quality Profile ~~

Type	Count	Percent	Quality Notice
------	-------	---------	----------------

1	17	22.67	Physical line length > 80 characters
17	12	16.00	Function comment content less than 10.0%
20	1	1.33	File comment content < 10.0%
28	1	1.33	Cyclomatic complexity > 10
31	1	1.33	Class/Struct comments are < 10.0%
38	12	16.00	Exception Handling "try"- "catch" has been identified
46	13	17.33	Function/Class Blank Line content less < 10.0%
47	1	1.33	File Blank Line content < 10.0%
49	3	4.00	Function appears to have null or blank parameters
51	13	17.33	No comment preceding a function block
52	1	1.33	No comment preceding a class block

75 100.00 Total Quality Notices

~~ Quality Notice Density ~~

Basis: 1000 (K)

Quality Notices/K LOC =	313.8 (31.38%)
Quality Notices/K eLOC =	362.3 (36.23%)
Quality Notices/K lLOC =	460.1 (46.01%)

~~ File Summary ~~

C Source Files *.c	0	C/C++ Include Files *.h:	0
C++ Source Files *.c* ..	0	C++ Include Files *.h* :	0
C# Source Files *.cs ...:	0	Java Source File *.jav*:	1
Other Source Files	0		
Total File Count	1		

Shareware evaluation licenses process only **20** files.
Paid licenses enable processing for an unlimited number of files.

Report Banner - Edit rsm.cfg File