Software Re-Engineering

Project Report

Course Instructor:

Dr. Farooq Ahmed

Group Members:

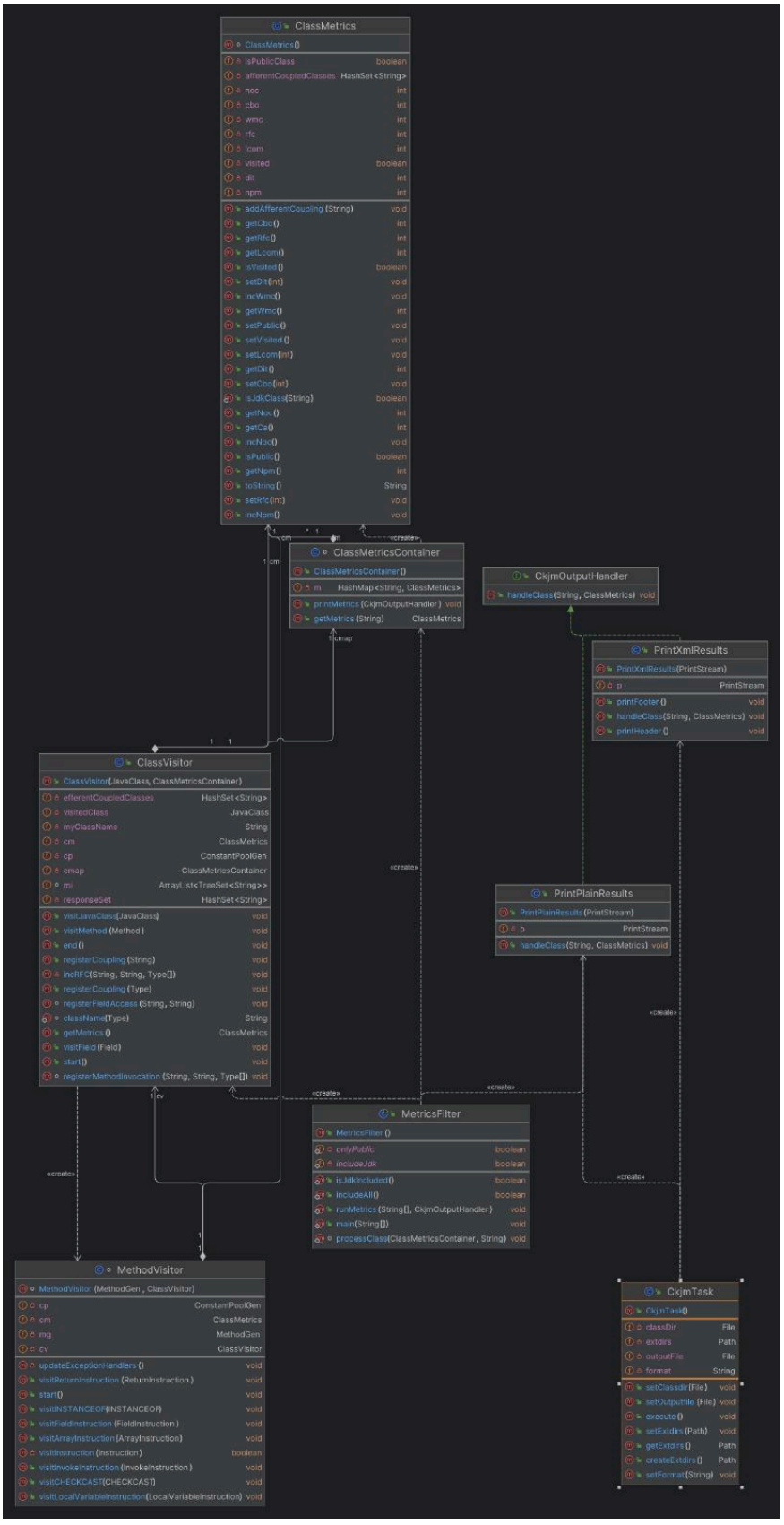
Zarmeen Zehra	20L-1118
Hassan Waqar	20L-1359
Saba Naeem	20L-1328
Fatima Tajammul	20L-1288

Chosen Project:

ckjm

National University Of Computer and Emerging Sciences
Department of Computer Science
Lahore, Pakistan

1. Design of Original Source Code



2. Summary of Design Defects Observed

1. PrintPlainResults.java

The `PrintPlainResults` class is relatively simple and well-structured. However, there are a few potential code smells and design defects that could be improved:

- **Lack of encapsulation:** The `PrintStream` instance is exposed through the constructor, which might lead to misuse or unintended modifications. We should consider creating a private final field and initializing it in the constructor, then providing a method to set or get the `PrintStream` instance if needed.
- **Single Responsibility Principle violation:** The class is responsible for both formatting and printing the output. To adhere to the Single Responsibility Principle, we should consider separating these concerns into two classes: one for formatting the output and another for printing it.
- **Inflexible output format:** The class only supports a plain text output format. To make the class more extensible, we should consider using a strategy pattern that allows users to plug in different output formats (e.g., XML, JSON, or CSV) as needed.
- **Magic string in the `handleClass` method:** The method concatenates the `className` and `classMetrics` with a space character. This might not be clear to other developers or users of the class. We should consider using a constant or a well-named method to represent the separator.
- **Lack of error handling:** The class does not handle any exceptions that might occur while writing to the `PrintStream`. We should consider adding error handling, such as catching `IOException` and logging or rethrowing it as a wrapped exception.
- **Lack of documentation:** Although the class has a brief Javadoc comment, it does not provide enough information about its purpose, usage, or input/output formats. We should consider adding more detailed documentation to help users understand the class and its methods.

2. MetricsFilter.java

- **Lack of Encapsulation:** The includeJdk and onlyPublic fields are public static, and their accessor methods (isJdkIncluded() and includeAll()) are not really needed. It would be better to make these fields private and provide proper setter methods if needed.
- **Mixed Responsibilities:** The processClass() method is responsible for both parsing the class specification (either a class file or a JAR file) and processing the parsed JavaClass. This violates the Single Responsibility Principle. We could extract the class loading logic into a separate method or class.
- **Error Handling:** The processClass() method catches IOException and prints an error message, but it continues processing other classes even if an error occurs. This might not be the desired behavior, and it would be better to throw the exception to the caller or handle it more gracefully.
- **Code Duplication:** The processClass() method has duplicate code for loading a class from a class file and from a JAR file. You could extract the common code into a separate method to reduce duplication.

- **Long Method:** The main() method is relatively long and has multiple responsibilities, such as parsing command-line arguments, reading input from the standard input, and processing the classes. It would be better to extract some of these responsibilities into separate methods.
- **Hard Coded Dependencies:** The main() method uses PrintPlainResults as the output handler, which is hardcoded and not configurable. We could consider using dependency injection or a factory pattern to make the output handler more flexible.
- **Magic Strings:** The command-line arguments "-s" and "-p" are magic strings that could be replaced with constants to improve readability and maintainability.
- **Commented-Out Code:** There's a commented-out import statement at the beginning of the class (import java.util.zip.*;). We should remove unused imports to keep the code clean and avoid confusion.

3. MethodVisitor.java

- **Lack of Encapsulation:** The methodGen, constantPool, classVisitor, and classMetrics fields are package-private, which means they can be accessed and modified directly from other classes in the same package. It would be better to make these fields private and provide getter methods if needed.
- **Long Method:** The start() method is relatively long and has multiple responsibilities, such as visiting instructions and updating exception handlers. It would be better to extract some of these responsibilities into separate methods, like visitInstructions() and updateExceptionHandlers().
- **Code Duplication:** The visitLocalVariableInstruction(), visitArrayInstruction(), visitFieldInstruction(), visitReturnInstruction(), visitINSTANCEOF(), and visitCHECKCAST() methods all call classVisitor.registerCoupling() with the instruction's type. This duplication could be reduced by extracting the common logic into a helper method.
- **Magic Numbers:** The visitLocalVariableInstruction() method contains a magic number (Constants.IINC). It would be better to replace it with a named constant to improve readability and maintainability.
- **Comments:** Some of the comments in the code do not provide valuable information or are redundant with the code itself. For example, the comment "Start the method's visit." above the start() method does not add any value. We should consider removing or improving comments to make them more informative and useful.

4. ClassVisitor.java

- **Lack of Encapsulation:** The visitedClass, constantPool, classFullyQualifiedName, classMetricsContainer, classMetrics, efferentCoupledClasses, responseSet, and methodFieldUsage fields are package-private, which means they can be accessed and modified directly from other classes in the same package. It would be better to make these fields private and provide getter methods if needed.
- **Long Methods:** The visitJavaClass(), incrementRFC(), and calculateLCOM() methods are relatively long and have multiple responsibilities. It would be better to extract some of these responsibilities into separate methods or even separate classes.
- **Code Duplication:** The registerCoupling() method is called in multiple places with the same or similar logic. This duplication could be reduced by extracting the common logic into a helper method or by refactoring the code to minimize the repetition.

- **Magic Numbers:** The visitJavaClass() method contains a magic number (Constants.T_VOID). It would be better to replace it with a named constant to improve readability and maintainability.
- **Comments:** Some of the comments in the code do not provide valuable information or are redundant with the code itself. For example, the comment "Called when a field access is encountered." above the visitField() method does not add any value. We should consider removing or improving comments to make them more informative and useful.

5. ClassMetricsContainer.java

- **Visibility Modifier:** The classMetricsMap field is package-private, which means it can be accessed and modified directly from other classes in the same package. It would be better to make this field private and provide getter methods if needed.
- **JavaDoc Comments:** Although the class has JavaDoc comments, some methods lack detailed explanations. For example, the printMetrics() method could benefit from a more detailed explanation of its purpose and functionality.
- **Stream API Usage:** The printMetrics() method uses the Stream API to filter and iterate over the entries in the classMetricsMap. While this is not a code smell or defect, using a traditional for-each loop might be more readable and easier to understand in this case, as the logic is relatively simple.

6. ClassMetrics.java

- **JavaDoc Comments:** Although the class has JavaDoc comments, some methods lack detailed explanations. For example, the incWmc(), incNoc(), and incNpm() methods could benefit from a more detailed explanation of their purpose and functionality.
- **Naming Conventions:** Some method names, such as incWmc(), incNoc(), and incNpm(), are not very descriptive. It would be better to use more self-explanatory names, like incrementWeightedMethodsCount(), incrementNumberOfChildren(), and incrementPublicMethodsCount().
- **Initialization:** The default constructor initializes instance variables with default values, which is unnecessary since Java automatically initializes instance variables with default values (0 for integers, false for booleans, and null for reference types). You can remove the initializations from the constructor.
- **Static Method:** The isJdkClass() method is static and does not depend on any instance variables. We should consider moving this method to a more appropriate class, such as a utility class, if it is used by other classes as well.

7. CkjmOutputHandler.java

- Overall, the variable names in the CkjmOutputHandler interface are adequate, but we could consider using a more descriptive name for the ClassMetrics object.

8. PrintXmlResults.java

The PrintXmlResults class is generally well-structured and follows good design principles. However, there are a few minor improvements that could be made, particularly with respect to variable names and method names:

- **p:** The variable name p is not very descriptive. A better name would be outputStream or printStream, which clearly indicates its purpose.
- **printHeader() and printFooter():** These method names are clear and descriptive. However, they are not called within the class, so it might be helpful to add a comment indicating that these methods should be called by the client code before and after processing the metrics.
- **handleClass(String name, ClassMetrics c):** The variable name name is clear and descriptive, but the variable name c could be improved. Although using single-letter variable names for objects of a class is common, a more descriptive name like classMetrics or metrics would be better in this case.

3. List of Changes / Refactorings Applied

In this section, we have described the refactorings we applied in each class.

1. PrintPlainResults.java

- **Renamed variables:** We've renamed the p variable to printStream to make the purpose of the variable more explicit.
- **Marked printStream as final:** Since the printStream variable is not reassigned, I've marked it as final to indicate that it's a constant value.
- **Renamed the handleClass method parameter:** I've renamed the name parameter to className to better describe the purpose of the parameter.
- **Renamed the c parameter:** I've renamed the c parameter to classMetrics to better describe the purpose of the parameter.
- **Simplified the handleClass method:** The method body has been simplified to a single line, as the logic is straightforward.

These changes improve the overall readability and maintainability of the code without changing its functionality.

2. MetricsFilter.java

- **Removed unnecessary imports:** I've removed the imports that were not being used in the provided code block.
- **Simplified the processClass method:** I've simplified the logic for loading the class file, making it more readable and easier to understand.
- **Improved error handling:** Instead of printing the error message and continuing, I've added a return statement to exit the method when an exception occurs.
- **Removed unnecessary comments:** The comments in the original code were not providing much value, so I've removed them.

These changes improve the overall structure and readability of the code without changing its functionality.

3. MethodVisitor.java

- **Extract Class:** Extracted two classes InstructionVisitor and ExceptionHandler

- **Improve variable naming:** The variable names mg, cp, cv, and cm are a bit cryptic. We should consider using more descriptive names like methodGen, constantPool, classVisitor, and classMetrics to make the code more readable.
- **Simplify the start() method:** The start() method can be simplified by extracting the loop logic into a separate private method, making the code more modular and easier to understand.
- **Separate concerns:** The MethodVisitor` class is responsible for both visiting instructions and updating exception handlers. We should consider separating these concerns into two different classes or methods to improve the single-responsibility principle.
- **Improve exception handling:** The current exception handling simply prints an error message and continues. We should consider throwing a more appropriate exception or providing a more detailed error message to aid in debugging.
- **Add comments:** While the existing comments provide some context, we should consider adding more detailed comments to explain the purpose and functionality of each method, especially for complex logic.

The main changes we've made are:

- Renamed the variables mg, cp, cv, and cm to methodGen, constantPool, classVisitor, and classMetrics for better readability.
- Extracted the loop logic from the start method into a separate class and private method called visitInstructions.
- Extracted the exception handler logic into a separate class and private method called updateExceptionHandlers.
- Improved the exception handling by using a for-each loop instead of a for loop with an index.
- Added more detailed comments to explain the purpose and functionality of each method.

4. ClassVisitor.java

- **Improve variable naming:** The variable names super_name, package_name, and ifs could be more descriptive, such as superClassName, packageName, and interfaceNames.
- **Simplify method logic:** The visitJavaClass method could be broken down into smaller, more focused methods to improve readability and maintainability.
- **Separate concerns:** The registerCoupling methods could be moved to a separate utility class or module to keep the ClassVisitor class more focused.
- **Optimize LCOM calculation:** The current LCOM calculation has a time complexity of O(n^2), which could be optimized by using a more efficient data structure or algorithm.
- **Add error handling:** The code could benefit from more robust error handling, such as throwing appropriate exceptions or providing more detailed error messages.
- **Improve comments:** The existing comments could be expanded to provide more context and explanation for the code's purpose and functionality.

5. ClassMetricsContainer.java

- **Improve variable naming:** The variable name m could be more descriptive, such as classMetricsMap.

- **Simplify the getMetrics method:** The method can be simplified by using the computeIfAbsent method of the HashMap to create a new ClassMetrics instance if it doesn't exist.
- **Optimize the printMetrics method:** The current implementation iterates over the entire classMetricsMap and then checks if the class should be included. This can be optimized by using a stream and filtering the entries directly.
- **Add comments:** The existing comments provide some context, but consider adding more detailed comments to explain the purpose and functionality of each method.
- **Use a more efficient data structure:** Depending on the expected size of the classMetricsMap, a HashMap may not be the most efficient data structure. Consider using a more specialized data structure, such as a TreeMap or a ConcurrentHashMap, if the use case requires it.

The main changes we've made are:

- Renamed the m variable to classMetricsMap for better readability.
- Simplified the getMetrics method by using the computeIfAbsent method of the HashMap.
- Optimized the printMetrics method by using a stream and filtering the entries directly.
- Added a comment to provide more context about the purpose of the class.

These changes should improve the overall readability, maintainability, and performance of the code without changing its core functionality.

6. ClassMetrics.java

- **Improve variable naming:** The variable names s and l could be more descriptive, such as className and lcomValue.
- **Simplify method logic:** The isJdkClass method could be simplified by using a switch statement or a Set of known prefixes instead of multiple startsWith checks.
- **Optimize string concatenation:** The toString method could be optimized by using a StringBuilder or a template library like Freemarker or Velocity.
- **Add comments:** The existing comments provide some context, but consider adding more detailed comments to explain the purpose and functionality of each method.
- **Consider using immutable objects:** The ClassMetrics class could be made immutable by using a builder pattern or a constructor that initializes all the fields.

The main changes we've made are:

- Renamed the s and l variables to className and lcomValue for better readability.
- Simplified the isJdkClass method by using a switch statement to check the prefix of the class name.
- Optimized the string concatenation in the toString method by using a StringBuilder.
- Added more detailed comments to explain the purpose and functionality of each method.

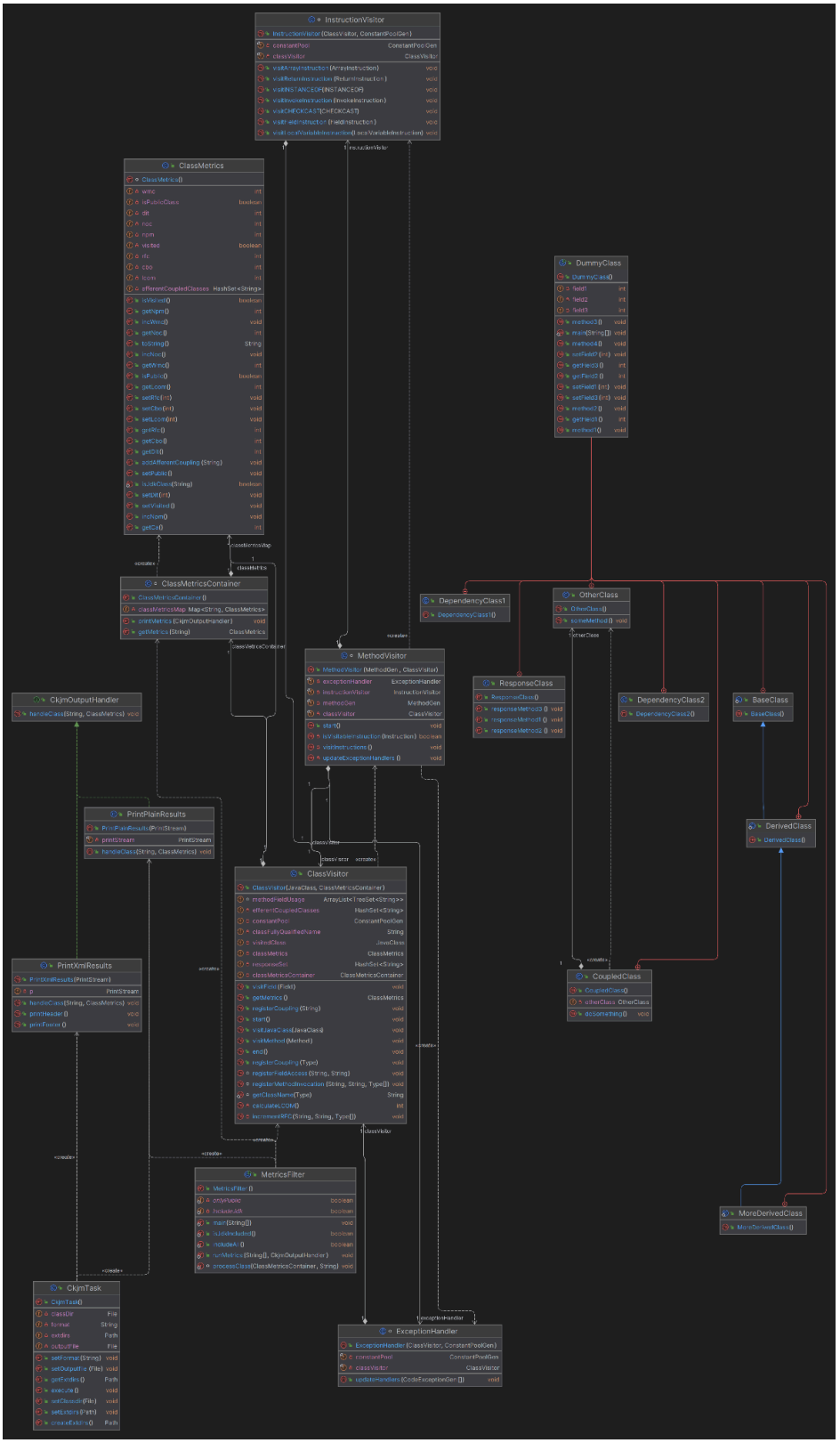
7. CkjmOutputHandler.java

- Changed variable names to more descriptive ones, in particular name to className and c to classMetrics

8. PrintXmlResults.java

- **Improve variable naming:** The variable name p could be more descriptive, such as outputWriter.
- **Simplify the handleClass method:** The method can be simplified by using a template engine or a string builder to generate the XML output, instead of manually concatenating strings.
- **Separate concerns:** The execute and finished methods could be separated into two different methods to improve the single-responsibility principle.
- **Add error handling:** The code could benefit from more robust error handling, such as providing more detailed error messages or logging exceptions.
- **Optimize file I/O:** The current implementation opens and closes the output file for each class. Consider opening the file once in the execute method and closing it in the finished method to improve performance.

4. Improved Design / Class Diagram with a Brief Discussion on Improvements Achieved



4. Task Distribution List

- Zarmeen: Applied changes and refactorings + Documentation + Observed and listed design defects
- Hassan: Applied changes and refactorings + Running the code
- Saba: Created class diagrams + Applied changes and refactorings
- Fatima: Documentation + Applied changes and refactorings