Université de Montpellier

HAI913I - Évolution et restructuration des logiciels

Static Code Analysis Report

TP N°01

Prepared by:

ABDELKADER-KHAROUBI Mohamed Yassine

Supervised by:

M. Djamel-Abdelhak Seriai
M. Bachar Rima

Academic Year: 2024/2025

Contents

1 Introduction

This report presents the implementation of a static code analyzer applied to an object-oriented application. Using the Eclipse JDT Abstract Syntax Tree (AST), we computed various statistics and insights related to class structures, methods, packages, and attributes. Additionally, we generated a call graph, providing a visual representation of method invocations across the application.

2 Theoretical Foundations

2.1 Abstract Syntax Tree (AST) with Eclipse JDT

2.1.1 Abstract Syntax Tree (AST)

An Abstract Syntax Tree (AST) is a hierarchical representation of the syntactic structure of source code. ASTs are widely used in compilers and static code analyzers to abstract the relationships between various code constructs.

2.1.2 Eclipse JDT

The Eclipse Java Development Tools (JDT) provides a sophisticated AST API that allows for parsing and manipulation of Java source code. This enables us to extract meaningful data from the code for analysis.

2.2 Visitor Design Pattern

The Visitor design pattern is integral to traversing AST nodes. It separates algorithmic operations from the data structures they operate on, allowing us to implement operations like counting classes, methods, and attributes without altering the AST structure.

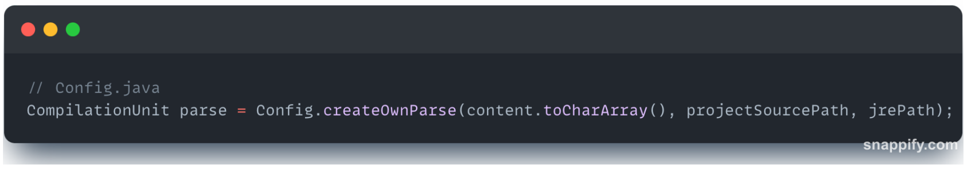
3 Implementation

3.1 Core Functionalities

3.1.1 AST Generation

The analyzer uses Eclipse JDT to parse Java source files and generate Abstract Syntax Trees. These ASTs serve as the foundation for all subsequent analysis.

Implementation:



```java
// Config.java
CompilationUnit parse = Config.createOwnParse(content.toCharArray(), projectSourcePath, jrePath);
```

3.1.2 Package Analysis

The **PackageVisitor** traverses the AST to collect information about packages and the classes they contain.

Key functionalities:

1. Identifying all non-interface classes
2. Mapping classes to their respective packages

3.1.3 Class Analysis

The **ClassVisitor** examines each class declaration to gather information about its structure.

Key functionalities:

1. Counting methods and attributes per class
2. Identifying class relationships (e.g., inheritance, implementation)

3.1.4 Method Analysis

The **ClassVisitor** and **MethodInvocationVisitor** work together to analyze individual methods.

Key functionalities:

1. Counting lines of code per method
2. Identifying method parameters and return types
3. Analyzing method complexity

3.1.5 Call Graph Generation

The **GUI** is responsible for drawing the graphic interface and it uses **MethodInvocationVisitor** for identifying method invocations and constructing a call graph.

Key functionalities:

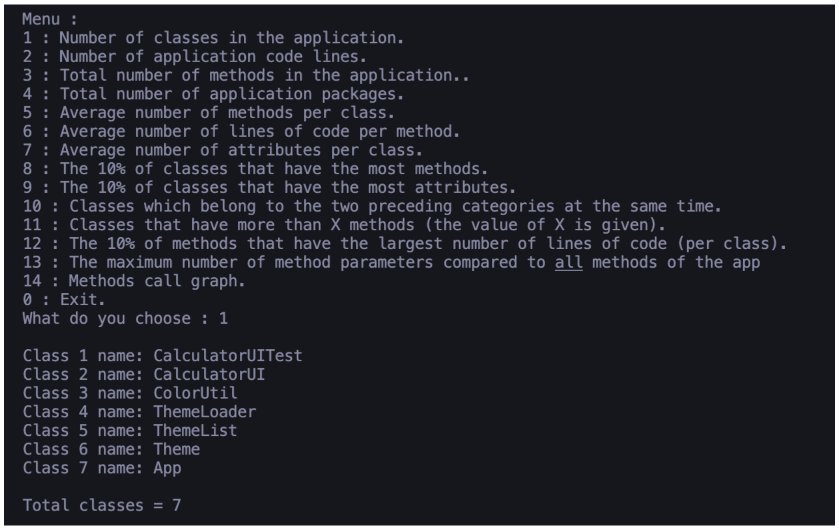
1. Tracking method invocations within each method
2. Building a graph representation of method calls

3.2 Statistical Analysis of Object-Oriented Applications

In this section, we will demonstrate how to perform statistical analysis on object-oriented applications using our analyzer on a simple calculator app developed in Java. This app includes basic functionalities like addition, subtraction, multiplication, and division. The source code for the calculator is available in the directory: **/resources/calculator-app/src**.

3.2.1 Number of Classes

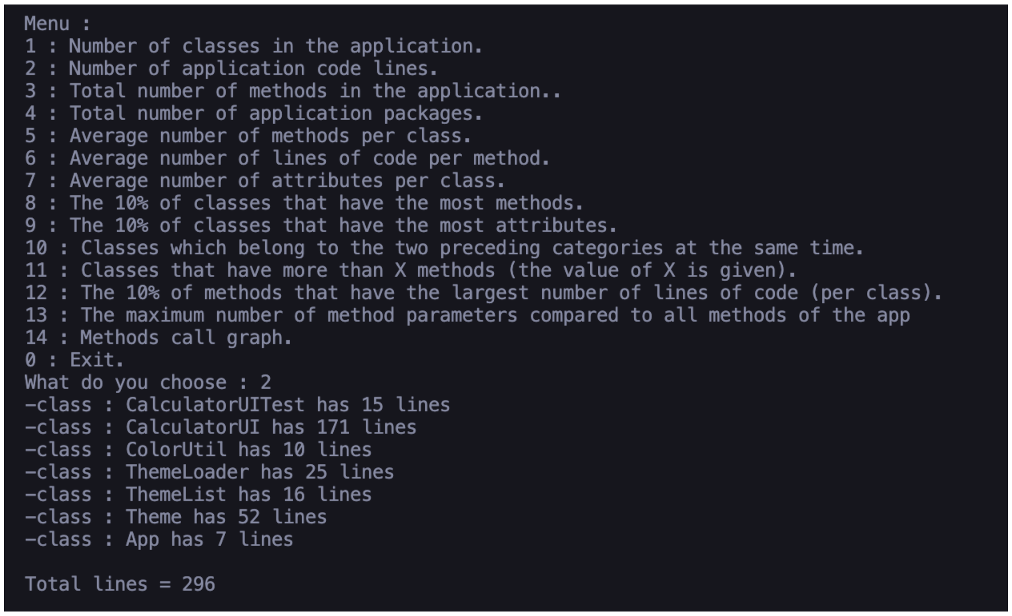
The total number of classes found in the application.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 1

Class 1 name: CalculatorUITest
Class 2 name: CalculatorUI
Class 3 name: ColorUtil
Class 4 name: ThemeLoader
Class 5 name: ThemeList
Class 6 name: Theme
Class 7 name: App

Total classes = 7
```

3.2.2 Number of Lines of Code

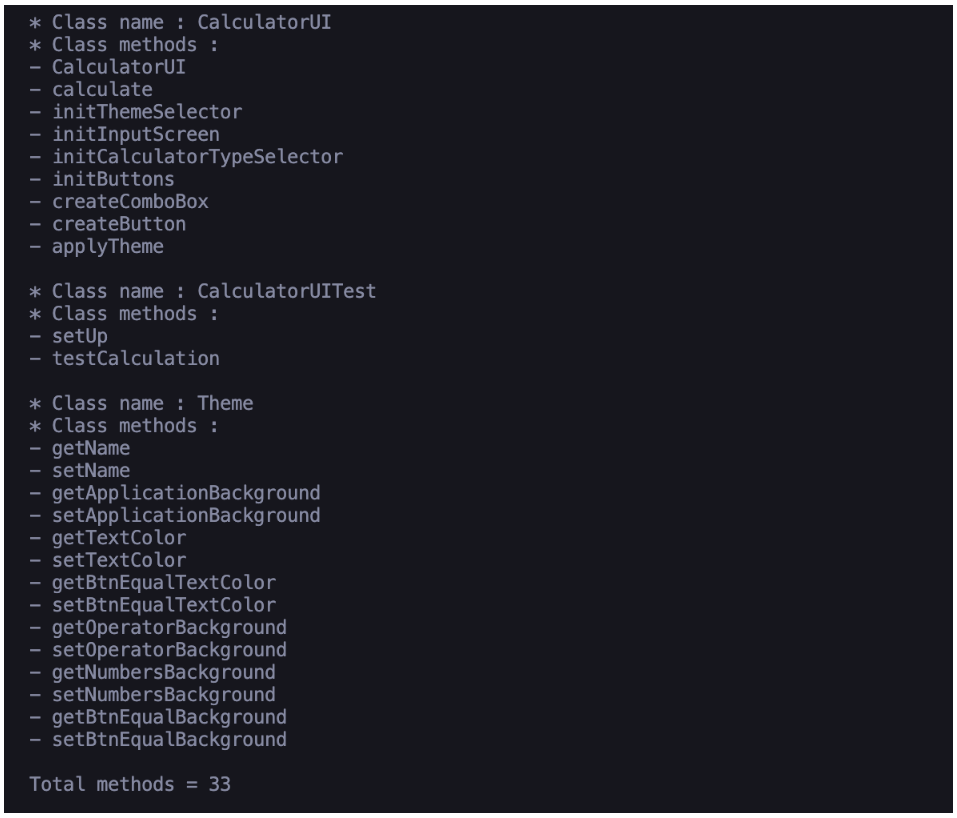
The total lines of code across the application.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 2
-class : CalculatorUITest has 15 lines
-class : CalculatorUI has 171 lines
-class : ColorUtil has 10 lines
-class : ThemeLoader has 25 lines
-class : ThemeList has 16 lines
-class : Theme has 52 lines
-class : App has 7 lines

Total lines = 296
```

3.2.3 Number of Methods

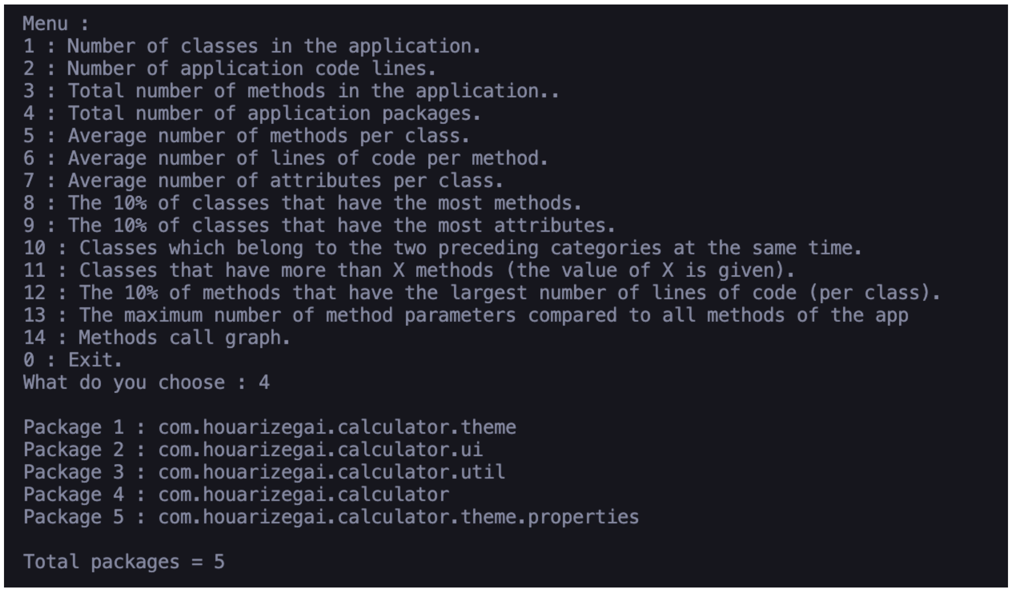
The total number of methods found in the classes.



```
* Class name : CalculatorUI
* Class methods :
- CalculatorUI
- calculate
- initThemeSelector
- initInputScreen
- initCalculatorTypeSelector
- initButtons
- createComboBox
- createButton
- applyTheme

* Class name : CalculatorUITest
* Class methods :
- setUp
- testCalculation

* Class name : Theme
* Class methods :
- getName
- setName
- getApplicationBackground
- setApplicationBackground
- getTextColor
- setTextColor
- getBtnEqualTextColor
- setBtnEqualTextColor
- getOperatorBackground
- setOperatorBackground
- getNumbersBackground
- setNumbersBackground
- getBtnEqualBackground
- setBtnEqualBackground

Total methods = 33
```

3.2.4 Number of Packages

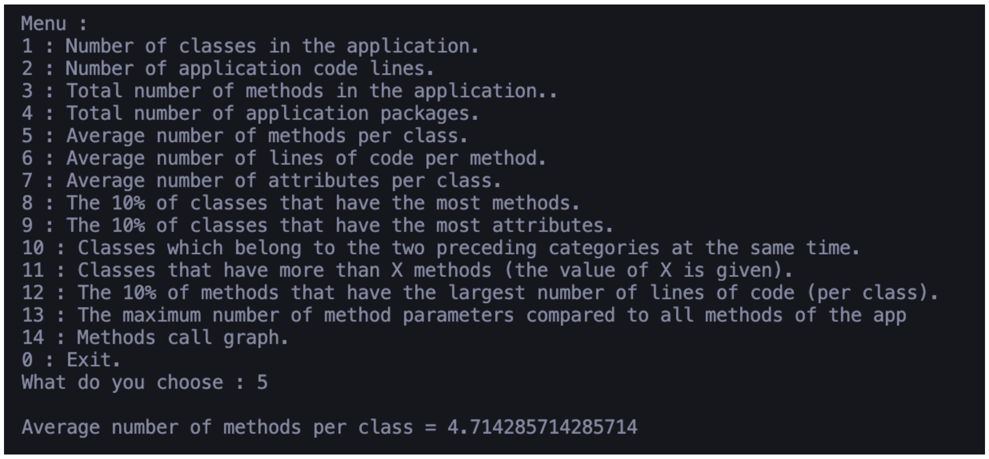
The total number of packages.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 4

Package 1 : com.houarizegai.calculator.theme
Package 2 : com.houarizegai.calculator.ui
Package 3 : com.houarizegai.calculator.util
Package 4 : com.houarizegai.calculator
Package 5 : com.houarizegai.calculator.theme.properties

Total packages = 5
```

3.2.5 Average Methods per Class

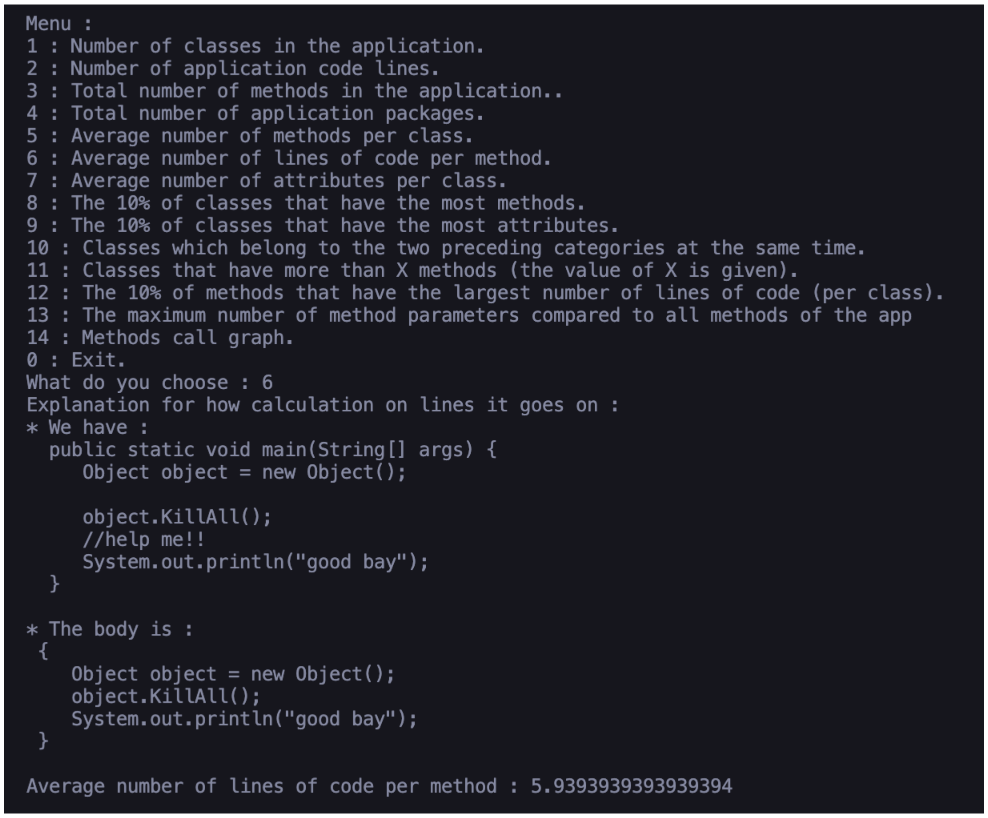
The average number of methods per class is calculated by dividing the total methods by the total classes.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 5

Average number of methods per class = 4.714285714285714
```

3.2.6 Average Lines per Method

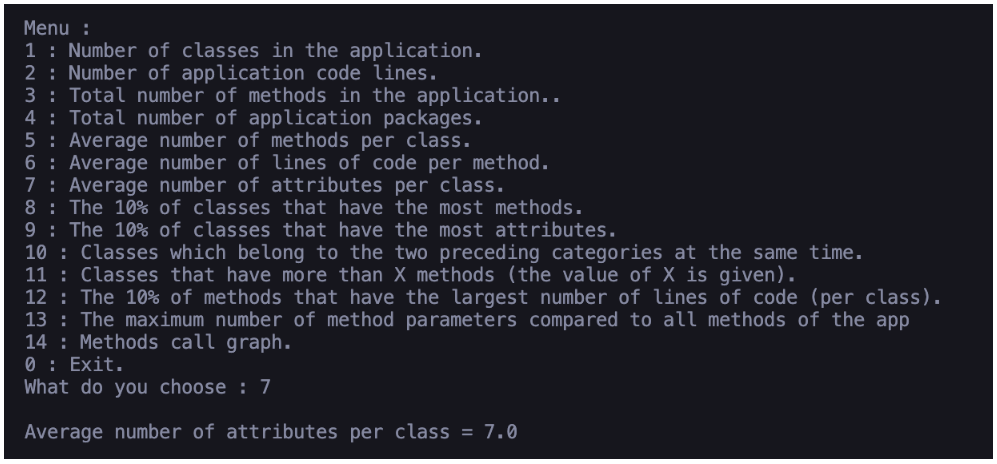
The average number of lines of code per method.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 6
Explanation for how calculation on lines it goes on :
* We have :
  public static void main(String[] args) {
     Object object = new Object();

     object.KillAll();
     //help me!!
     System.out.println("good bay");
  }

* The body is :
 {
    Object object = new Object();
    object.KillAll();
    System.out.println("good bay");
 }

Average number of lines of code per method : 5.9393939393939394
```

3.2.7 Average Attributes per Class

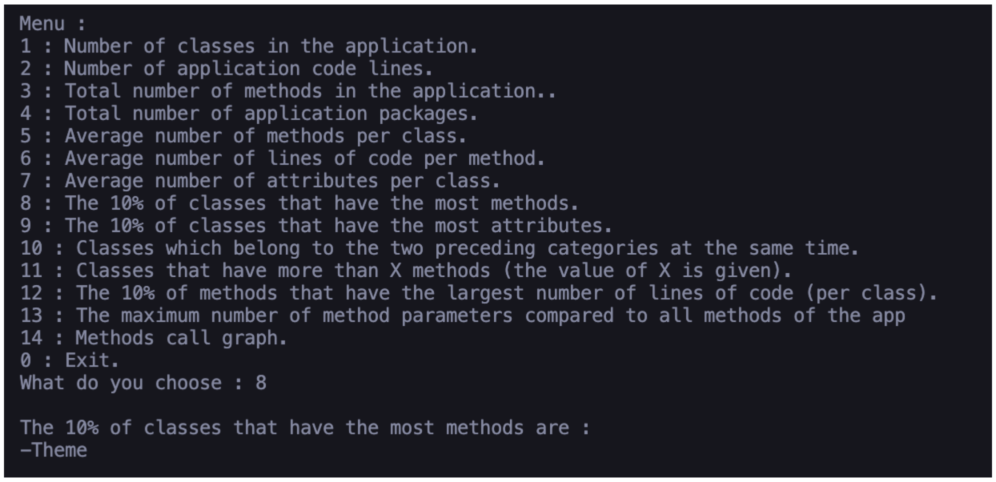
The average number of attributes per class.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 7

Average number of attributes per class = 7.0
```

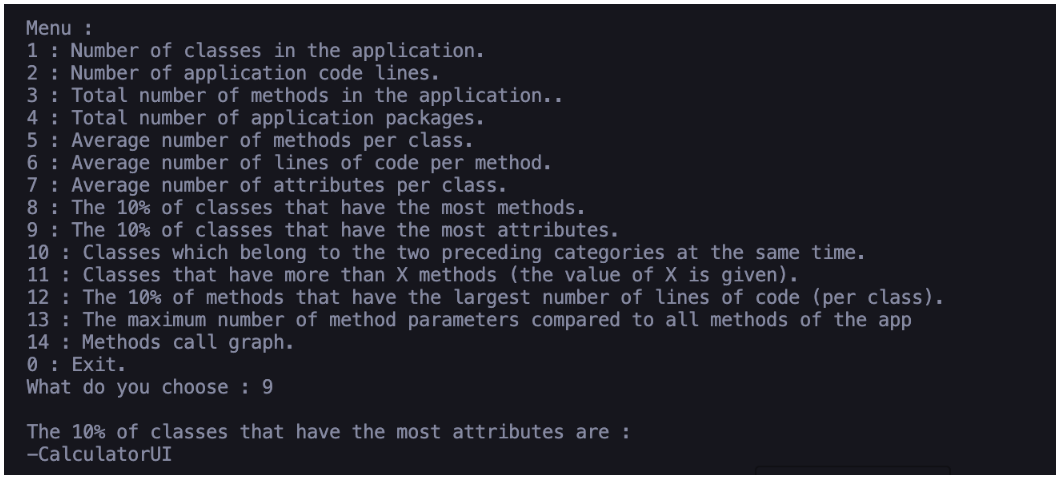
3.2.8 Top 10% Classes by Methods

Identifies the top 10% of classes with the most methods.


```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 8

The 10% of classes that have the most methods are :
-Theme
```

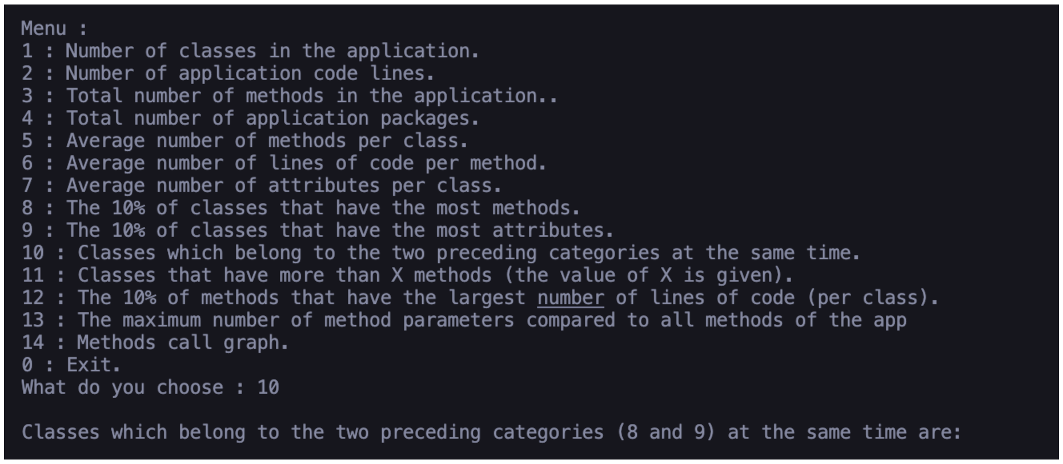
3.2.9 Top 10% Classes by Attributes

Identifies the top 10% of classes with the most attributes.


```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 9

The 10% of classes that have the most attributes are :
-CalculatorUI
```

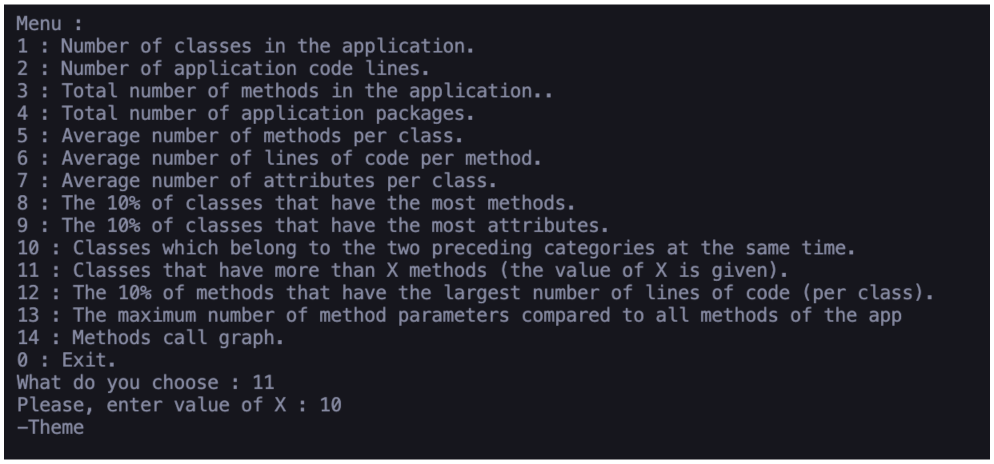
3.2.10 Overlapping Classes in Top Methods and Attributes

Classes that appear in both the top 10% of methods and top 10% of attributes.


```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 10

Classes which belong to the two preceding categories (8 and 9) at the same time are:
```

3.2.11 Classes with More than X Methods

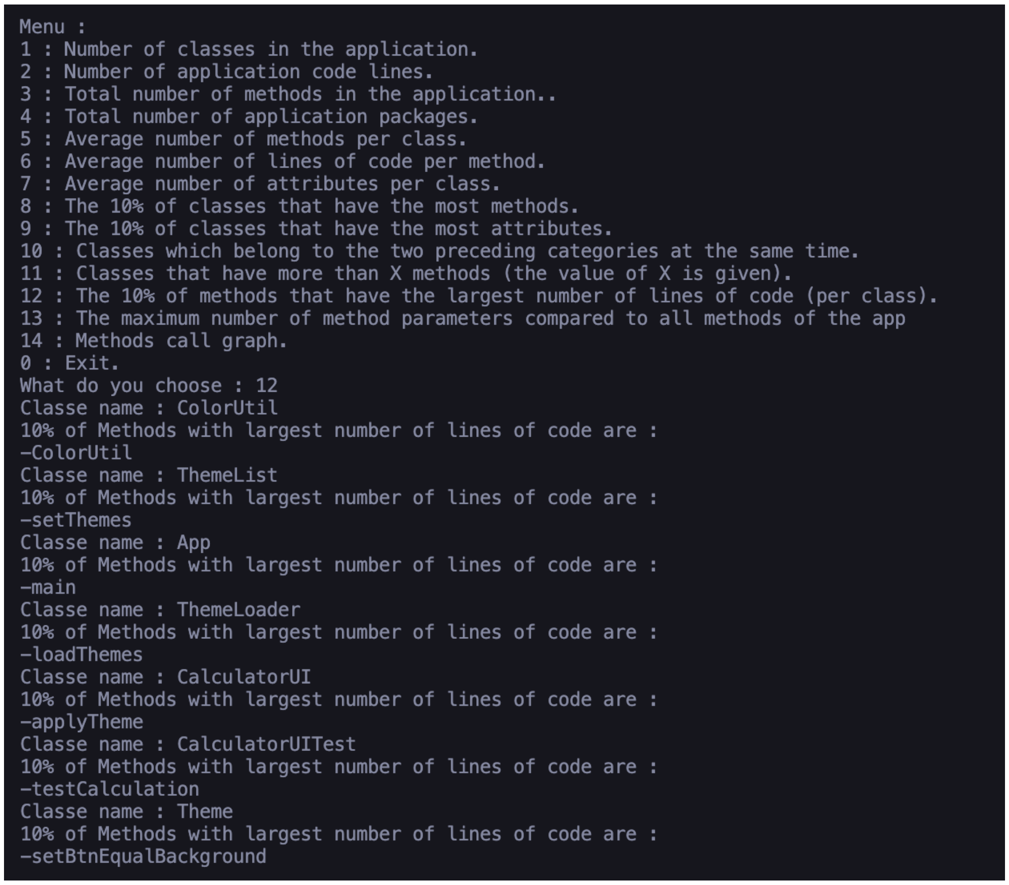
Lists classes that have more than a user-defined number of methods (X).



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 11
Please, enter value of X : 10
-Theme
```

3.2.12 Top 10% Methods by Lines of Code

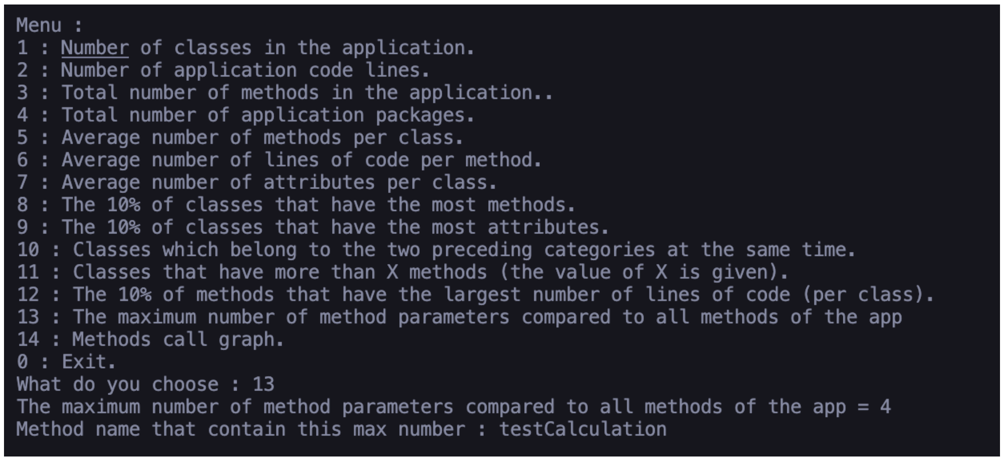
Lists the methods with the highest number of lines of code, ranked within their respective classes.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 12
Classe name : ColorUtil
10% of Methods with largest number of lines of code are :
-ColorUtil
Classe name : ThemeList
10% of Methods with largest number of lines of code are :
-setThemes
Classe name : App
10% of Methods with largest number of lines of code are :
-main
Classe name : ThemeLoader
10% of Methods with largest number of lines of code are :
-loadThemes
Classe name : CalculatorUI
10% of Methods with largest number of lines of code are :
-applyTheme
Classe name : CalculatorUITest
10% of Methods with largest number of lines of code are :
-testCalculation
Classe name : Theme
10% of Methods with largest number of lines of code are :
-setBtnEqualBackground
```

3.2.13 Maximum Parameters Across Methods

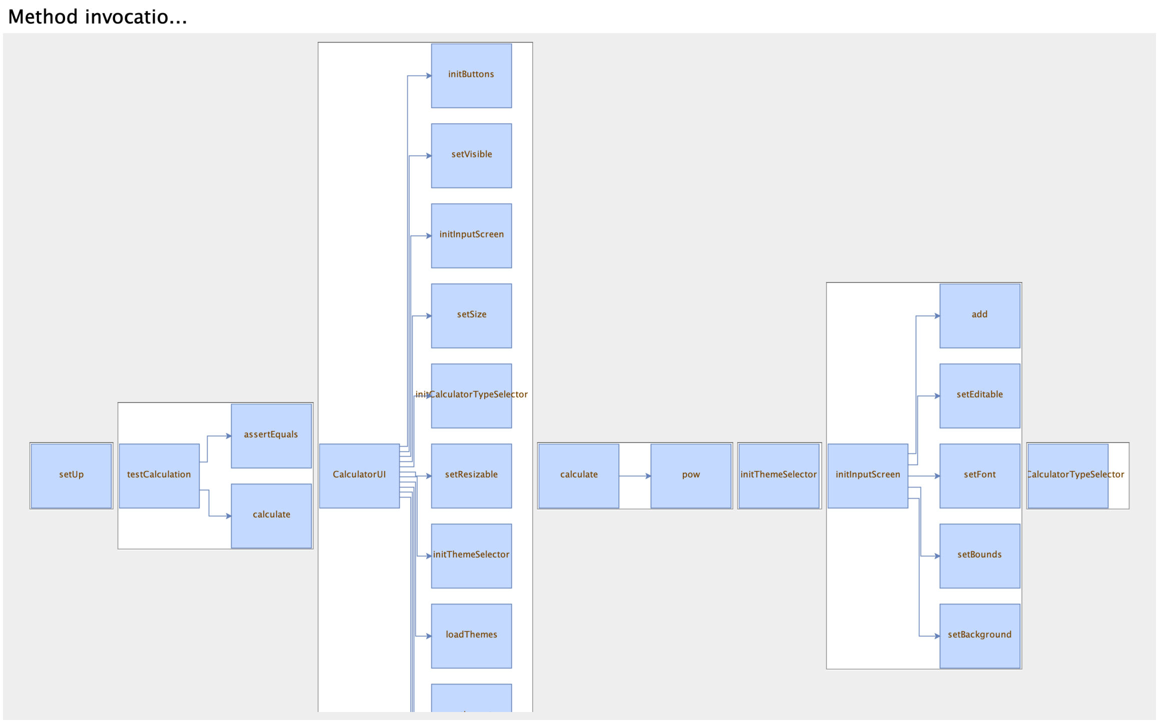
The method with the maximum number of parameters in the application.



```
Menu :
1 : Number of classes in the application.
2 : Number of application code lines.
3 : Total number of methods in the application..
4 : Total number of application packages.
5 : Average number of methods per class.
6 : Average number of lines of code per method.
7 : Average number of attributes per class.
8 : The 10% of classes that have the most methods.
9 : The 10% of classes that have the most attributes.
10 : Classes which belong to the two preceding categories at the same time.
11 : Classes that have more than X methods (the value of X is given).
12 : The 10% of methods that have the largest number of lines of code (per class).
13 : The maximum number of method parameters compared to all methods of the app
14 : Methods call graph.
0 : Exit.
What do you choose : 13
The maximum number of method parameters compared to all methods of the app = 4
Method name that contain this max number : testCalculation
```

3.3 Call Graph Construction

To create a call graph, we analyzed method invocations across the application. This provides insight into how methods interact with each other. We first generated a structural representation of the call graph and then visualized it using graphing tools.



4 Conclusion

The static code analysis leveraged the AST from Eclipse JDT to extract detailed insights about object-oriented Java applications. We computed critical metrics such as the number of classes, methods, lines of code, and constructed a call graph. This exercise showcased the utility of static code analysis in understanding and optimizing codebases.