

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Adikaram	Student ID:	18875678
Other name(s):	Kaluthara Korallalage Sandakelum Tharindu		
Unit name:	Software Metrics-(ISAD3002)	Unit ID:	ISAD3002
Lecturer / unit coordinator:	Ms.Asanthika Imbulpitiya	Tutor:	-
Date of submission:	2017-05-19	Which assignment?	SM Assignment 2 Document

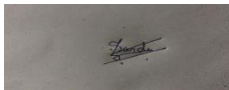
I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:



Date of
signature:

2017-05-18

(By submitting this form, you indicate that you agree with all the above text.)

Figure Table

Figure No	Description	Page No
Figure 1	NetBeans project open	03
Figure 2	NetBeans project run	04
Figure 3	Boundary Test1 output	13
Figure 4	Boundary Test2 output	15
Figure 5	Complex Test1 output	17
Figure 6	Complex Test2 output	19
Figure 7	Complex Test3 output	21
Figure 8	Complex Test4 output	23
Figure 9	Complex Test5 output	25
Figure 10	Complex Test6 output	27
Figure 11	Complex Test7 output	29
Figure 12	Real-World Test 1 output	31
Figure 13	Real-World Test 2 output	32,33
Figure 14	RSM installed	34
Figure 15	RSM locate file path	35
Figure 16	RSM run command	35
Figure 17	SMAssignment2.java quality issue	36
Figure 18	SMAssignment2.java quality issue	36
Figure 19	CodeAttribute.java quality issue	37
Figure 20	MethodInfo.java quality issue	38
Figure 21	FindNextClass.java quality issue	38
Figure 22	Test case execution path	5

Introduction

This document was created for full filling the requirements of the Software metrics second assignment. The requirement was to write a program. The program can be written in any language.

The purpose of writing the program was to extract specific information from a byte code which is being created after compiling a java source code. The information contains the details of caller and callee of the methods and the parameter types as well as the type of the method such as abstract method, recursive method. At last the program should display a call tree of a specific method which is selected by the user.

Java language was used as the program language. And java reflection as well as the ClassFile.java file which is available at the Blackboard and provided by the unit coordinator of the software metrics unit is used to continue the program. Several java class files have to be created for completing the program.

RSM tool which is commercially available tool for quality analysis is being used for analysing the quality of the program. Some identified quality issues were fixed.

Number of test cases are created for testing purposes. Each test case contains a specific scenario to test the code and see how the code is going to react to the different kind of inputs provided by the user.

To determine the expected metric results javap tool was used. Javap tool extract the information of the java Virtual Machine.

The total program was implemented using the NetBeans IDE. The version of the IDE id 8.2. And the for creating the test cases simple IDE is used such as Sublime Text. And the test cases were executed using the Command Prompt.

Technology Requirements

The Java version which the code was implemented - Java Version jdk1.8.2_77, jre1.8.2_66

Operating System that used - Windows 10

The main IDE used for coding – NetBeans IDE

The execution Environments – NetBeans IDE (Version 8.2) and Command Prompt

Test case development IDE – Sublime Text and Visual Code

Execute the source code in NetBeans

Open the project in the NetBeans

Open the NetBeans ide -> Go to 'file' tab -> got to 'Open Project' menu

Select the required project

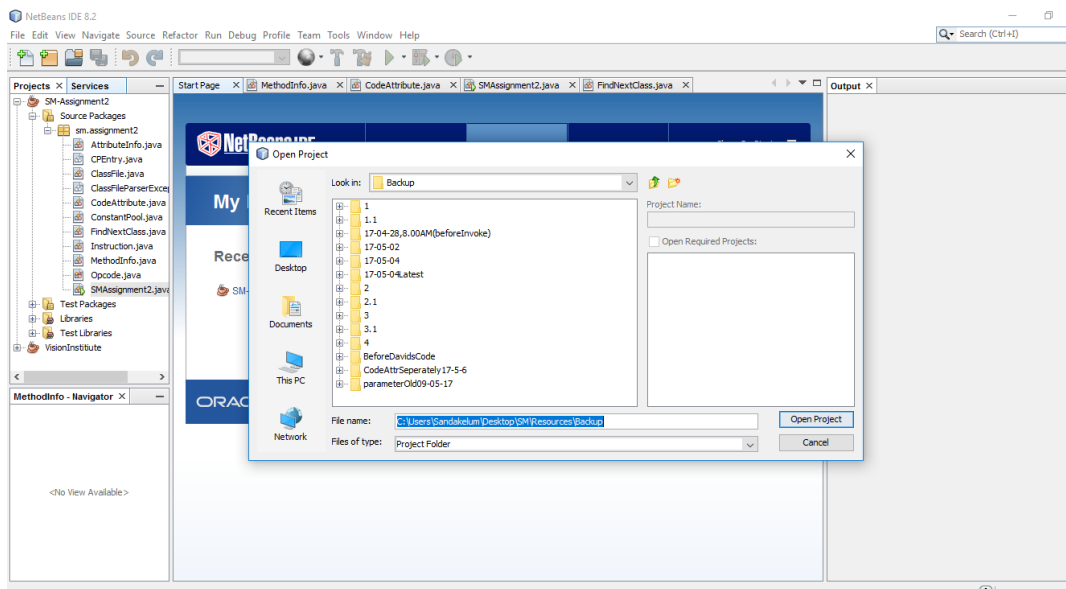


Figure 1

Run the Main class file

Right click on the project -> Run File

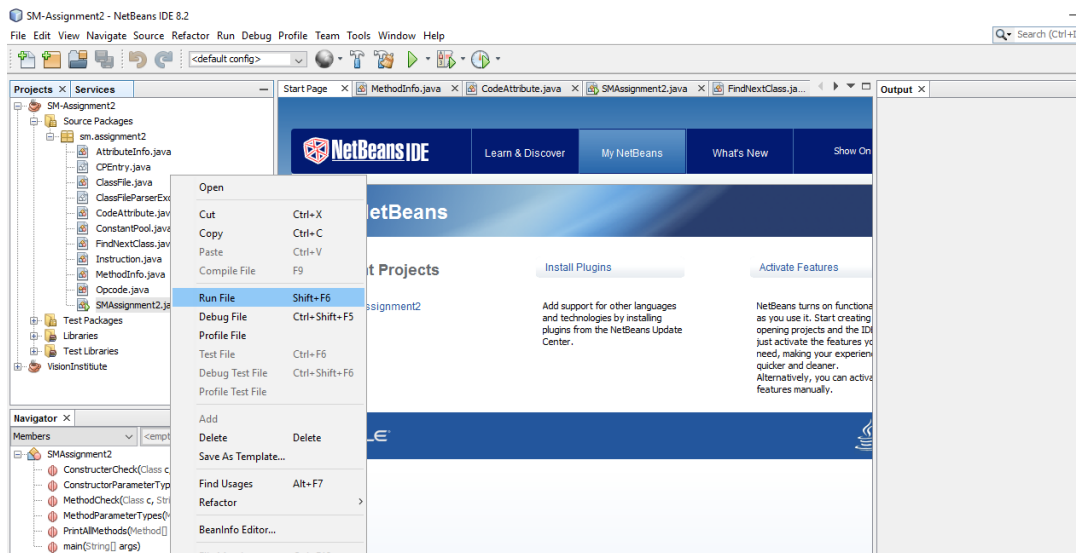


Figure 2

How to execute the test classes

Copy and paste the class files in to the SMAssignment2 folder before run the program.

Because the path has been set to the SMAssignment2 folder.

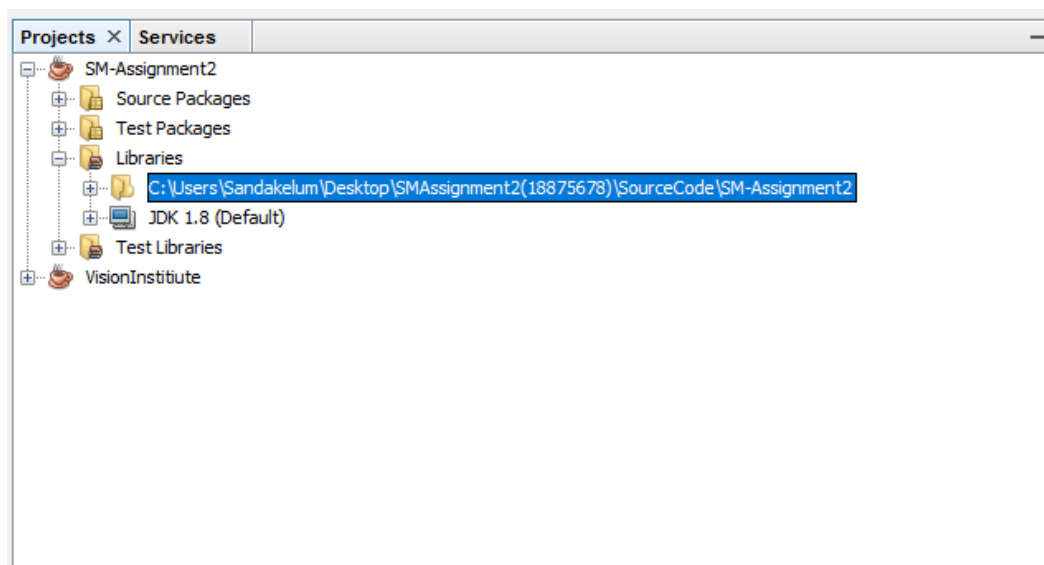


Figure 22

Functionalities

According to the requirements of the Software Metrics Assignment2 there are several functionalities that needed to be completed.

The main functionalities identified are

- User should be able to select a class file using command line arguments or standard inputs.
- User should be able to select a particular method or a constructor using command line arguments or standard inputs.
- Determine the call tree for the selected method or constructor.
- Print the total number of unique methods and constructors called, and the total number of classes involved.
- Handle overloaded methods and constructors those having the same name but different parameters.
- Indicate recursion.
- Indicate abstract method calls.
- Indicate and skip the missing classes. Some classes cannot be found because they are being created by the JVM itself.
- If a method or constructor makes multiple calls to a particular callee, ignore all but the first call.

From the above identified functionalities, I have completed the below functionalities. To the best of my knowledge the below functionalities are fully completed.

- User can select a class file using standard inputs.
- User can select a particular method or a constructor from the list displayed to user using standard inputs.
- Determine the call tree for the selected method or constructor.
- Handled overloaded methods and constructors those having the same name but different parameters.
- Indicated abstract method calls. (partially completed)
- Indicated and skip the missing classes.

Structure of the java class files

The code was implemented using a partially implemented class file which is ClassFile.java. The byte code is started reading from the ClassFile.java. and ClassFile.java calls for several other class files to perform the byte code reading.

The structure of the class files used to implement the whole program to full fill the requirements of the Software Metrics Assignment 2.

- SMAssignment2.java
- ClassFileParserException.java
- ClassFile.java
- MethodInfo.java
- AttributeInfo.java
- CodeAttribute.java
- FindNextClass.java
- ConstantPool.java
- CPEntry.java
- Instruction.java
- Opcode.java

The java class files which are provided by the Curtin university.

- ClassFile.java (Partially Edited)
- ClassFileParserException.java
- ConstantPool.java
- CPEntry.java
- Instruction.java
- Opcode.java

The java class files which are implemented by me.

- SMAssignment2.java
- ClassFile.java (Partially Edited)
- MethodInfo.java
- AttributeInfo.java
- CodeAttribute.java
- FindNextClass.java

The structure of the class files which have been implemented by myself

The main purpose of the source code is to identify a call tree for a given method. User can input any class file from the beginning of the execution. Then the program will display all the methods inside the provided class file. And user will have to enter a method name with the parameters by his choice. And the program will retrieve the methods which has been called inside the provided method. If that method has called to another method inside another class file then that class file should be loaded again to find out the details of the identified method. Such a way the program needed to be continued until it prints the call tree for the initial method that user has been entered.

SMAssignment2.java

The purpose of the SMAssignment2.java file is to accept the user inputs from the user. First the class will accept a .class file from the user. Then after it will display all the methods inside the .class file. And user can select a method from the list displayed in the console with the parameters. Parameter handling was done using a separate method. After the selection was done the method will be passed to the ClassFile.java.

ClassFile.java

ClassFile.java is used to extract the byte code and retrieve information. Inside the ClassFile.java several class files are being called. Up to Constant Pool the code was given. Inside the ClassFile.java all the fields will be skipped until the Method Info fields. To retrieve the method info the MethodInfo.java file is called.

MethodInfo.java

MethodInfo.java file is used to extract the data of the methods inside the .class file. The Method name, Method signatures and the attribute type which the method is belonging to can be identified from the MethodInfo.java file. Inside the MethodInfo.java Attribute.java file is being called.

AttributeInfo.java

Attribute.java file is used to extract the details of the attribute type and the length of the bytes that needed to be allocated to the specified attribute. According to the provided requirements of the assignment 2 the AttributeInfo.java file is mainly focused on the CODE attribute.

Therefore, inside the AttributeInfo.java file A class file called CodeAttribute.java is called to retrieve the details of the code attribute.

CodeAttribute.java

CodeAttribute.java class file is used to implements the code attribute. The main logic of selecting specific method has been implemented inside the CodeAttribute.java. Inside CodeAttribute.java the details of the caller and callee as well as the class name of the defined method and the methods which have been called inside the identified method can be gathered. And additionally, the parameter types and the return types can be gathered. Inside the CodeAttribute.java a class called `FindNextClass.java has been called.

FindNextClass.java

The purpose of using find next class is to call for the ClassFile.java with a new method to get the opcode details for the that method. If a method was found which is being called by another method the FindNextClass.java is called. It will call the ClassFile.java and the ClassFile.java will again execute with the new method name and the class file which has been passed from the FindNextClass.java file.

High Level architecture of the class files which have been implemented by myself

SMAssignment2.java

Inside SMAssignment2.java file 6 methods have been implemented.

Main() - Main method is used to get the class file name and the method name from the user as a input parameter.

MethodCheck() – Check the availability of the method that user has entered from the methods of the class file. And display it to the user that the method is a valid method.

ConstructorCheck() - Check the availability of the constructor that user has entered from the constructors of the class file. And display it to the user that the constructor is a valid constructor.

PrintAllMethods() – Print all the methods of the class file to the user.

MethodParameterTypes() – Identify the parameters for each method of the class file. And return the parameters in the string format.

ConstructorParameterTypes() – Identify the parameters for each constructor of the class file. And return the parameters in the string format.

ClassFile.java

Inside ClassFile.java file 1 method has been implemented.

ClassFile() - Inside the constructor read the bytes to extract the details of the class file.

MethodInfo.java

Inside MethodInfo.java file 2 methods have been implemented.

MethodInfo() – Extract the details of the each method. Such as method name and the parameters. Handle the constructors specifically.

ParameterTypes() – Handle the parameters which retrieved from the bytecode and return a string value.

AttributeInfo.java

Inside ClassFile.java file 1 method has been implemented.

AttributeInfo() - Inside Attribute.java file it will specifically consider about the CODE attribute. If another attribute found it will be skipped.

CodeAttribute.java

Inside ClassFile.java file 1 method has been implemented.

CodeAttribute() – Extract the details inside the code attribute of each method. Using the CodeAttribute method all the methods which have been called inside another method can be gathered.

FindNextClass.java

Inside ClassFile.java file 2 methods have been implemented.

CompareMethods() – pass the identified method to the class file again to retrieve the data inside that method.

CompareConstructors() – pass the identified constructor to the class file again to retrieve the data inside that constructor.

Test Case Analysis

Test case analysis is done to make sure that the source code is working properly. Therefore, several test cases have to be created to check the different scenarios of the source code.

Each requirement should have been tested with a proper test case. Infinite number of test cases can be created for a source code. So, when creating test cases main scenarios should be focused.

There are several test cases have been implemented for the source code which was implemented to full fill the requirements of the Software Metrics Assignment2. And each test case has a different scenario.

Test cases include the scenarios to cover the requirements specified in the Assignment2.

All the test cases that has been used to test the program are contained in the folder which is named as TestClases. Inside the TestClases folder there is a folder higher achy for each test case type. Boundary test cases are cases are contained in the BoundaryTestCases folder. Complex Test cases are contained in the complexTestCases folder. And the real-world test cases are contained in the RealworldTestCases folder. Inside the test case type folder there is another folder higher achy with the number which is assigned for the test case.

E.g.: Test01 of the complex test cases is contained in the folder which is TestClases\complexTestCases\Test01.

Test Cases for boundary cases

Boundary cases includes the test cases to check the error handlings. While developing the program there need to be proper error handling mechanism to continue the program with no errors. Therefore, to check that the error handling mechanisms are working properly.

Test Case 1

The test case 1 is mainly used to test that when the class name is incorrect the program should print an error message.

Test case scenario - If the program runs successfully the program should print an error message.

Preconditions - Class file should not be in the correct file path.

Input test data -

Class name with .class extension.

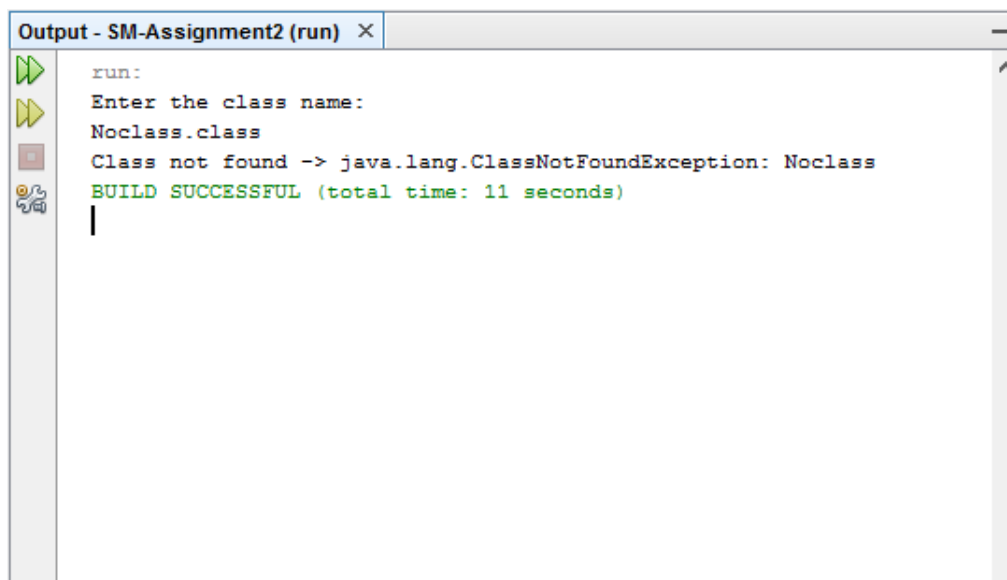
- For the test 1 the class file name is Noclass.class.

Steps to be executed - Enter the class name.

Expected results - Error message saying that class is not found.

Actual results - Error message saying that class is not found.

Pass/fail status - Pass



The screenshot shows an IDE output window titled "Output - SM-Assignment2 (run)". The output text is as follows:

```
run:
Enter the class name:
Noclass.class
Class not found -> java.lang.ClassNotFoundException: Noclass
BUILD SUCCESSFUL (total time: 11 seconds)
```

Figure 3

Test Case 2

The test case 2 is mainly used to after printing the call tree whether the program can identify the number of unique methods, number of unique constructors as well as number of unique classes involves.

Test case scenario - If the program runs successfully the program should print unique methods, constructors and classes.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 1 the class file name is A.class.

Steps to be executed - Enter the class name.

Expected results - print the number of unique methods, constructors and classes. And also print the identified unique methods, constructors and classes.

Actual results - displayed the number of unique methods, constructors and classes. And also, displayed the identified unique methods, constructors and classes.

Pass/fail status - Pass

```
Output - SM-Assignment2 (run) X
*****
Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
method1()
The method you have selected is -> method1()
*****

The first method is -> A.method1()

Inside method1() - > A.method2() has been called

Inside method2() - > A.method3(int) has been called

Inside method2() - > A.method3(String, int) has been called

Inside method1() - > B.B(double) has been called

Inside B(double) - > Object.Object()[Missing] has been called

Inside B(double) - > B.method5() has been called

Inside method1() - > B.method4() has been called

Inside method4() - > B.method5() has been called

Inside method4() - > B.method6() has been called

Number of unique METHODS are: 7
[method1(), method2(), method3(int), method3(String, int), method5(), method4(), method6()]

Number of unique CONSTRUCTORS are: 1
[B(double)]

Number of unique CLASSES are: 3
[A, B, Object]

BUILD SUCCESSFUL (total time: 7 seconds)
|
```

Figure 4

Test Cases for complex cases

Complex cases include the test scenarios for checking the main requirements in the Software Metrics Assignment 2. The scenarios checked and the output status of the test cases have been described below.

Test Case 1

The test case 1 is mainly used to test the call tree when there are no method calls inside another method. When the class file and the method name is entered, there is only one method should display which is the method user has selected.

Test case scenario - If the program runs successfully in the call tree only one method should display.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 1 the class file name is NoCallTree.class

Select a method name from the list.

- For the test 1 the method name is addition(int).

Steps to be executed - Enter the class name and method name.

Expected results - Only the method the user has selected should display as the call tree.

- For the test 1, addition(int) method should be displayed.

Actual results - only the addition(int) method is displayed.

Pass/fail status - Pass


```
run:
Enter the class name:
NoCallTree.class
*****

<--The methods and the parameters of each method of the selected class-->
NoCallTree()
addition(int)
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
addition(int)
The method you have selected is -> addition(int)
*****

The first method is -> NoCallTree.addition(int)

BUILD SUCCESSFUL (total time: 17 seconds)
|
```

Figure 5

Test Case 2

The test case 2 is mainly used to test the call tree. When the class file and the method name is entered, the call tree needed to be displayed in the console.

Test case scenario - If the program runs successfully the call tree should be displayed.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 2 the class file name is A.class

Select a method name from the list.

- For the test 2 the method name is method1().

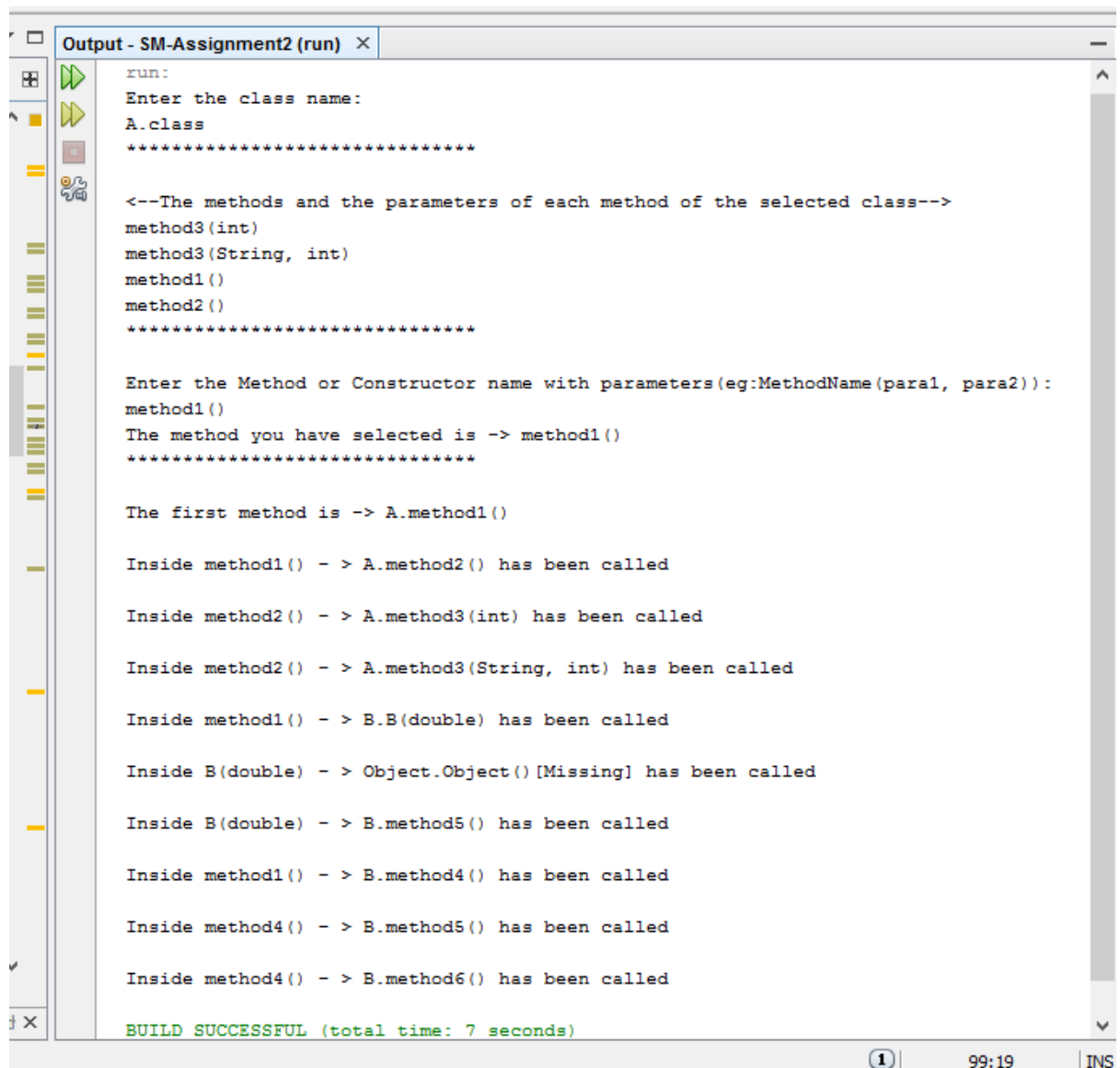
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 1 call tree should be displayed for the method1().

Actual results - Call tree for the method1().

Pass/fail status - Pass



```
Output - SM-Assignment2 (run) X
run:
Enter the class name:
A.class
*****

<--The methods and the parameters of each method of the selected class-->
method3(int)
method3(String, int)
method1()
method2()
*****

Enter the Method or Constructor name with parameters (eg:MethodName(para1, para2)):
method1()
The method you have selected is -> method1()
*****

The first method is -> A.method1()

Inside method1() - > A.method2() has been called

Inside method2() - > A.method3(int) has been called

Inside method2() - > A.method3(String, int) has been called

Inside method1() - > B.B(double) has been called

Inside B(double) - > Object.Object() [Missing] has been called

Inside B(double) - > B.method5() has been called

Inside method1() - > B.method4() has been called

Inside method4() - > B.method5() has been called

Inside method4() - > B.method6() has been called

BUILD SUCCESSFUL (total time: 7 seconds)
```

Figure 6

Test Case 3

The test case 3 is mainly used to test the recursion methods. When the class file and the method name is entered, the call tree needed to be displayed in the console. And when a recursion method found that method should specifically displayed with recursion keyword.

Test case scenario - If the program runs successfully in the call tree the recursion methods should be displayed with the recursion keyword.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 3 the class file name is Recursion.class

Select a method name from the list.

- For the test 3 the method name is FIndfactorial(int).

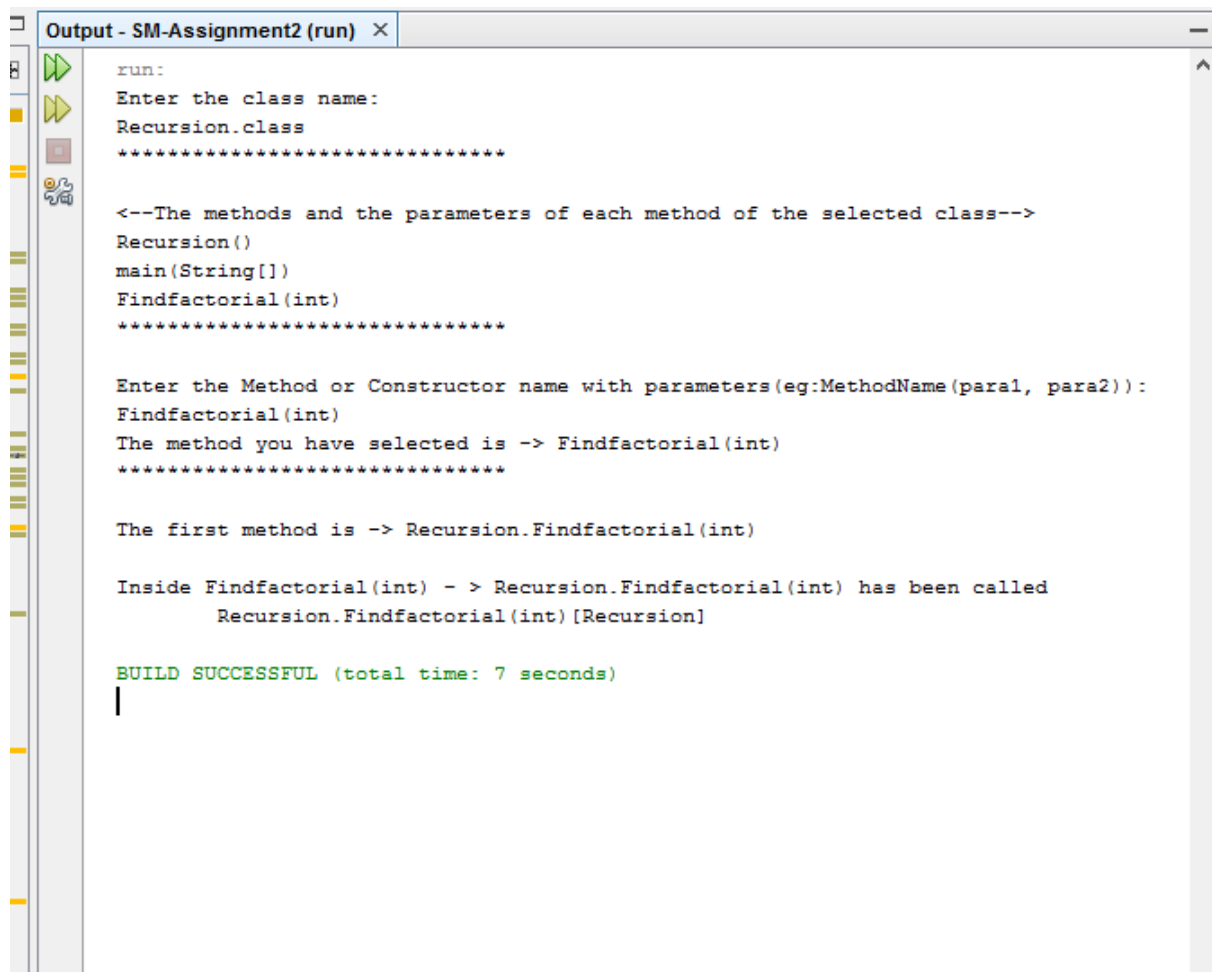
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 3, in the call tree the FIndfactorial(int) method should be displayed only one time with the recursive keyword.

Actual results - In the call tree the FIndfactorial(int) method is displayed only one time with the recursive keyword.

Pass/fail status - Pass



```
run:
Enter the class name:
Recursion.class
*****

<--The methods and the parameters of each method of the selected class-->
Recursion()
main(String[])
Findfactorial(int)
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
Findfactorial(int)
The method you have selected is -> Findfactorial(int)
*****

The first method is -> Recursion.Findfactorial(int)

Inside Findfactorial(int) - > Recursion.Findfactorial(int) has been called
    Recursion.Findfactorial(int) [Recursion]

BUILD SUCCESSFUL (total time: 7 seconds)
|
```

Figure 7

Test Case 4

The test case 4 is mainly used to test the methods which have been called several times inside the same method. In that situation, the method which has been called several times should print only one time.

Test case scenario - If the program runs successfully in the call tree the method which has been called several times should print only one time.

Preconditions - Class file should be in the correct file path.

Input test data —

Class name with .class extension.

- For the test 4 the class file name is CallingMethodsInSameClass.class

Select a method name from the list.

- For the test 4 the method name is FirstMethod ().

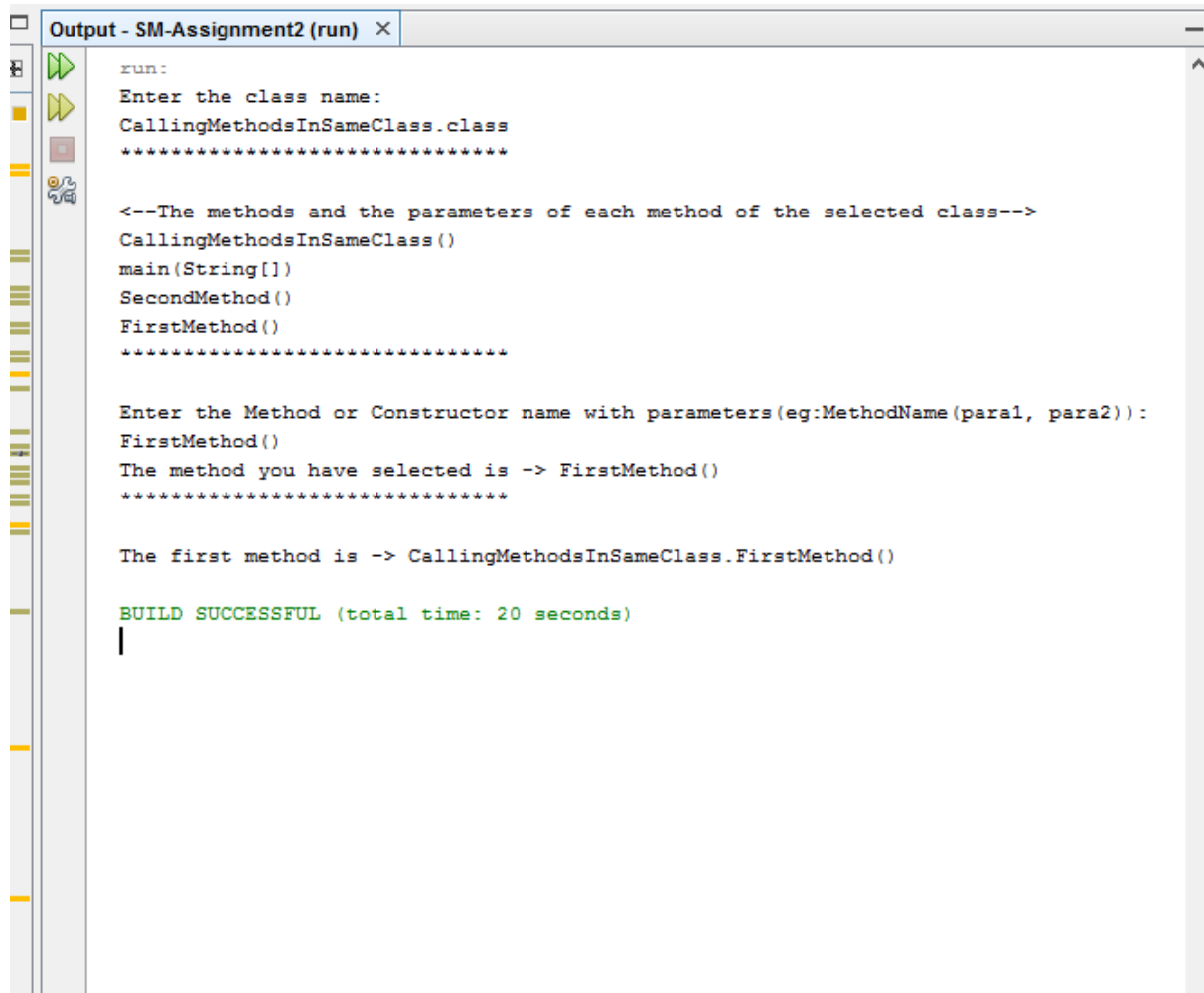
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 4, in the call tree the method which has been called several times inside a same method should print only one time.

Actual results - In the call tree the method which has been called several times inside a same method is displayed only one time.

Pass/fail status — Pass



```
run:
Enter the class name:
CallingMethodsInSameClass.class
*****

<--The methods and the parameters of each method of the selected class-->
CallingMethodsInSameClass()
main(String[])
SecondMethod()
FirstMethod()
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
FirstMethod()
The method you have selected is -> FirstMethod()
*****

The first method is -> CallingMethodsInSameClass.FirstMethod()

BUILD SUCCESSFUL (total time: 20 seconds)
|
```

Figure 8

Test Case 5

The test case 5 is mainly used to test the missing classes. When a missing class found, the program should print the missing keyword in front of the missing class.

Test case scenario - If the program runs successfully the missing keyword should print.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 5 the class file name is MissingMethods.class

Select a method name from the list.

- For the test 5 the method name is missingCheck (int).

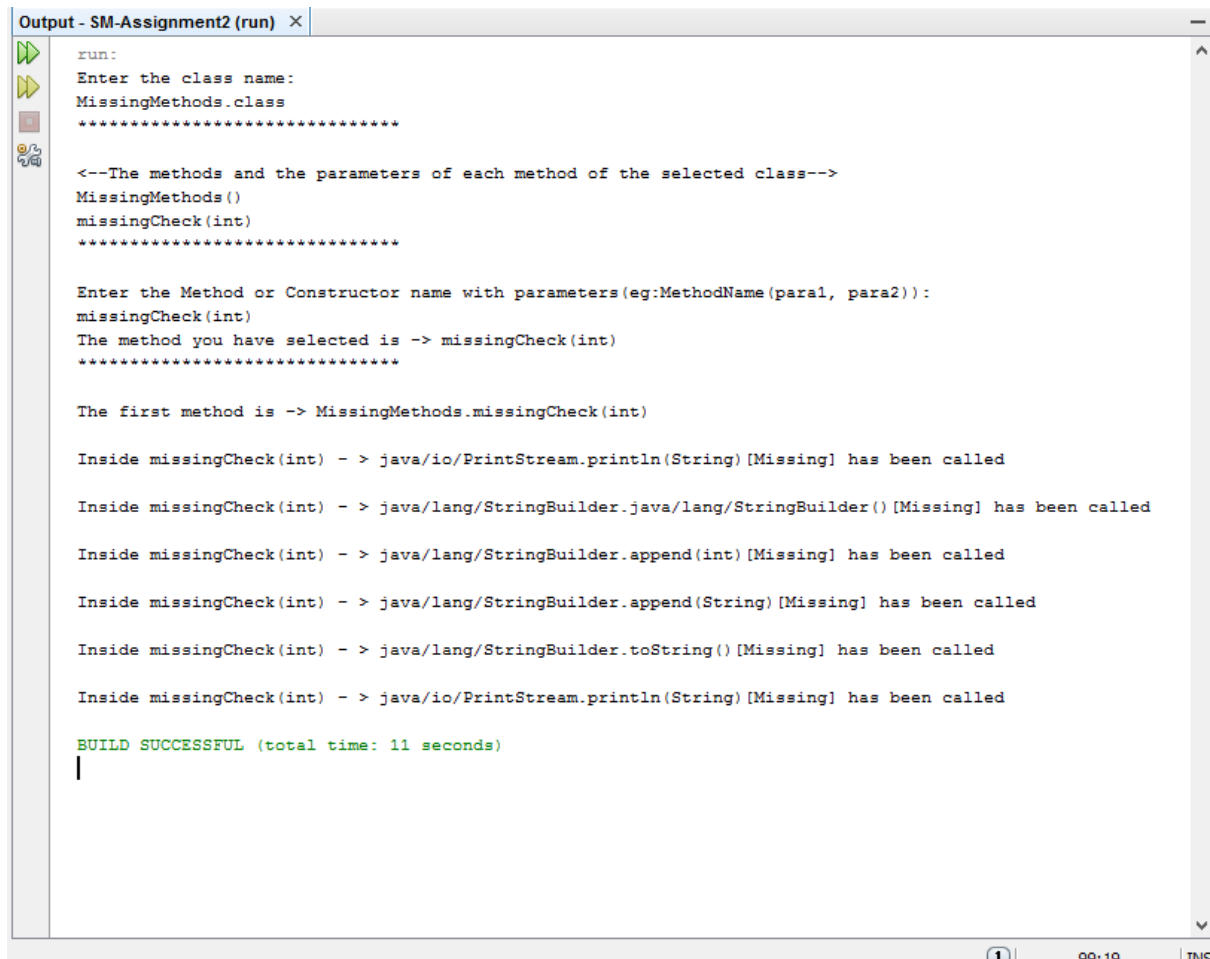
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 5 the missing keyword should print for the classes that cannot be found.

Actual results - The missing keyword is printed for the classes that cannot be found.

Pass/fail status - Pass



```
run:
Enter the class name:
MissingMethods.class
*****

<--The methods and the parameters of each method of the selected class-->
MissingMethods()
missingCheck(int)
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
missingCheck(int)
The method you have selected is -> missingCheck(int)
*****

The first method is -> MissingMethods.missingCheck(int)

Inside missingCheck(int) - > java/io/PrintStream.println(String) [Missing] has been called
Inside missingCheck(int) - > java/lang/StringBuilder.java/lang/StringBuilder() [Missing] has been called
Inside missingCheck(int) - > java/lang/StringBuilder.append(int) [Missing] has been called
Inside missingCheck(int) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside missingCheck(int) - > java/lang/StringBuilder.toString() [Missing] has been called
Inside missingCheck(int) - > java/io/PrintStream.println(String) [Missing] has been called

BUILD SUCCESSFUL (total time: 11 seconds)
|
```

Figure 9

Test Case 6

The test case 6 is mainly used to test the methods which have been called several times inside the same method in a for loop. In that situation, the method which has been called several times should print only one time.

Test case scenario - If the program runs successfully in the call tree the method which has been called several times should print only one time.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 6 the class file name is MethodInForLoop.class

Select a method name from the list.

- For the test 6 the method name is main (String []).

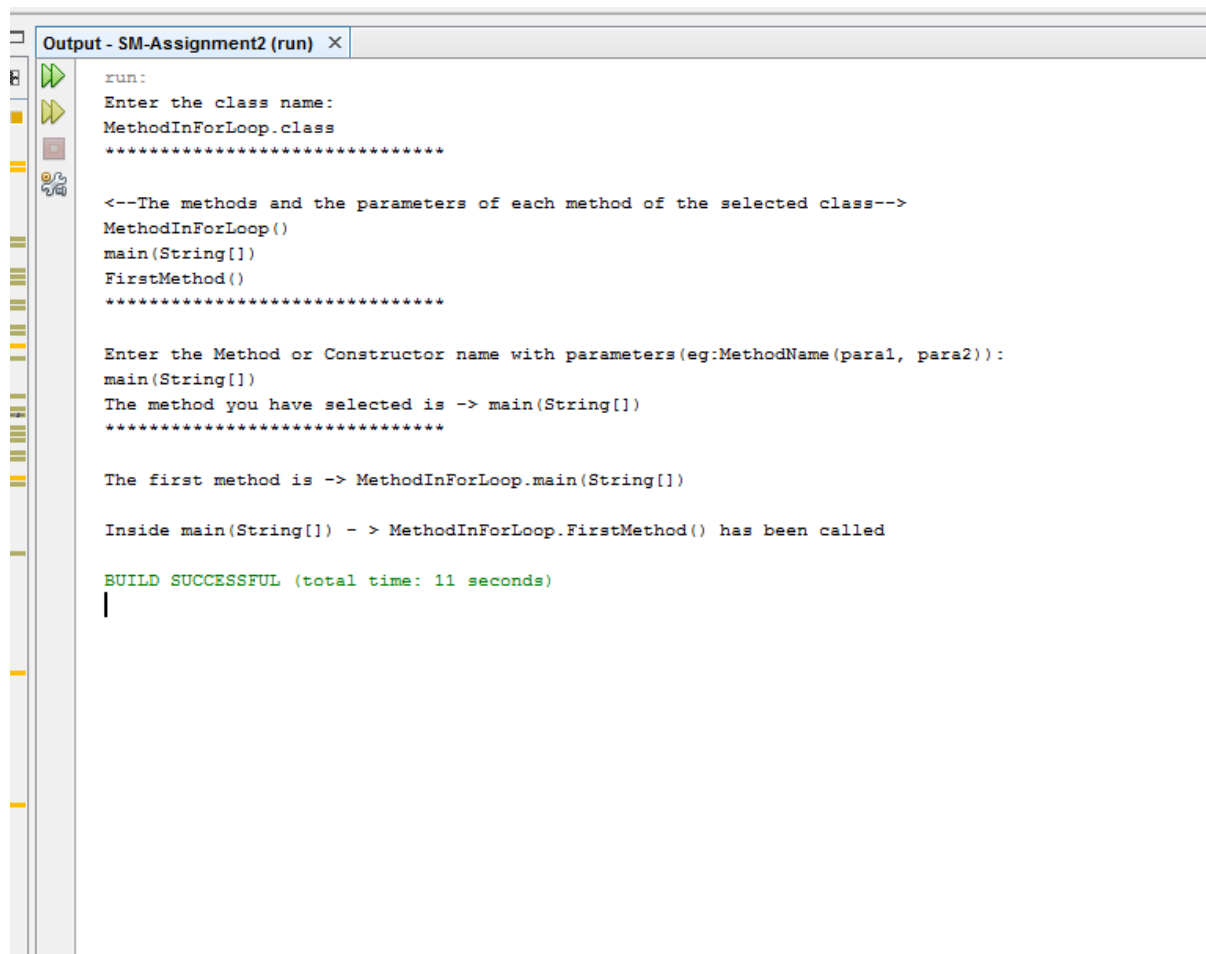
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 6, in the call tree the method which has been called several times should print only one time.

Actual results - In the call tree the method which has been called several times is printed only one time.

Pass/fail status - Pass



```
run:
Enter the class name:
MethodInForLoop.class
*****

<--The methods and the parameters of each method of the selected class-->
MethodInForLoop()
main(String[])
FirstMethod()
*****

Enter the Method or Constructor name with parameters (eg:MethodName(para1, para2)):
main(String[])
The method you have selected is -> main(String[])
*****

The first method is -> MethodInForLoop.main(String[])

Inside main(String[]) - > MethodInForLoop.FirstMethod() has been called

BUILD SUCCESSFUL (total time: 11 seconds)
|
```

Figure 10

Test Case 7

The test case 7 is mainly used to test the abstraction methods. If an abstract method has been identified, it should be displayed with the abstraction keyword.

Test case scenario- If the program runs successfully, abstract methods should be displayed with the abstraction keyword.

Preconditions - Class file should be in the correct file path.

Input test data –

Class name with .class extension.

- For the test 7, the class file name is `abstractTest2.class`

Select a method name from the list.

- For the test 7, the method name is `main(String args[])`.

Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 7, in the call tree, the abstract method should print with the abstract keyword.
- Actual results - In the call tree, the abstract methods printed have been printed without the abstract keyword.

Pass/fail status– Fail

```

Output - SM-Assignment2 (run) X
run:
Enter the class name:
abstractTest2.class
*****

<---The methods and the parameters of each method of the selected class--->
main(String[])
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
main(String[])
The method you have selected is -> main(String[])
*****

The first method is -> abstractTest2.main(String[])

Inside main(String[]) - > MainAbstract.MainAbstract() has been called

Inside main(String[]) - > abstractTest.method1() has been called

Inside main(String[]) - > abstractTest.mathod2() has been called

Inside mathod2() - > java/io/PrintStream.println(String) [Missing] has been called

Number of unique METHODS are: 3
[main(String[]), method1(), mathod2()]

Number of unique CONSTRUCTORS are: 0
[]

Number of unique CLASSES are: 3
[abstractTest2, abstractTest, java/io/PrintStream]

BUILD SUCCESSFUL (total time: 8 seconds)
|

```

Test Cases for real-world cases

Real world cases include the test scenarios for checking the main requirements in the Software Metrics Assignment 2 using real world examples. The scenarios checked and the output status of the test cases have been described below.

Test Case 1

The test case 1 is mainly used to test the recursion methods. When the class file and the method name is entered, the call tree needed to be displayed in the console. And when a recursion method found that method should specifically displayed with recursion keyword.

Test case scenario - If the program runs successfully in the call tree the recursion methods should be displayed with the recursion keyword.

Preconditions - Class file should be in the correct file path.

Input test data -

Class name with .class extension.

- For the test 3 the class file name is RecursionExampleDirectory.class

Select a method name from the list.

- For the test 3 the method name is pow(int, int).

Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 3, in the call tree the pow(int, int) method should be displayed only one time with the recursive keyword.

Actual results - In the call tree the pow(int, int) method is displayed only one time with the recursive keyword.

Pass/fail status - Pass

```
Enter the class name:
RecursionExampleDirectory.class
*****

<--The methods and the parameters of each method of the selected class-->
RecursionExampleDirectory()
main(String[])
pow(int, int)
getSize(class Directory)
fib(int)
minWasted(class [I, int, int)
fact(int)
*****

Enter the Method or Constructor name with parameters(eg:MethodName(para1, para2)):
main(String[])
The method you have selected is -> main(String[])
*****

The first method is -> RecursionExampleDirectory.main(String[])

Inside main(String[]) - > RecursionExampleDirectory.RecursionExampleDirectory() has been called

Inside RecursionExampleDirectory() - > Object.Object() [Missing] has been called

Inside main(String[]) - > Directory.Directory() has been called

Inside Directory() - > Object.Object() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > File.File(int) has been called

Inside File(int) - > Object.Object() [Missing] has been called

Inside Directory() - > Object.Object() [Missing] has been called

Inside Directory() - > Directory.Directory() has been called

Inside Directory() - > Object.Object() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > java/lang/Math.random() [Missing] has been called

Inside Directory() - > Directory.Directory() [Recursion]

Inside main(String[]) - > RecursionExampleDirectory.getSize(L, doubleirectory;) has been called

Inside main(String[]) - > java/io/PrintStream.println(int) [Missing] has been called

Number of unique METHODS are: 1
[main(String[])]

Number of unique CONSTRUCTORS are: 3
[RecursionExampleDirectory(), Directory(), File(int)]

Number of unique CLASSES are: 6
[RecursionExampleDirectory, Object, Directory, java/lang/Math, File, java/io/PrintStream]

BUILD SUCCESSFUL (total time: 9 seconds)
```

Figure 12

Test Case 2

The test case 2 is mainly used to test the call tree. When the class file and the method name is entered, the call tree needed to be displayed in the console.

Test case scenario - If the program runs successfully the call tree should be displayed.

Preconditions - Class file should be in the correct file path.

Input test data —

Class name with .class extension.

- For the test 2 the class file name is SmartCard.class

Select a method name from the list.

- For the test 2 the method name is main(String[]).

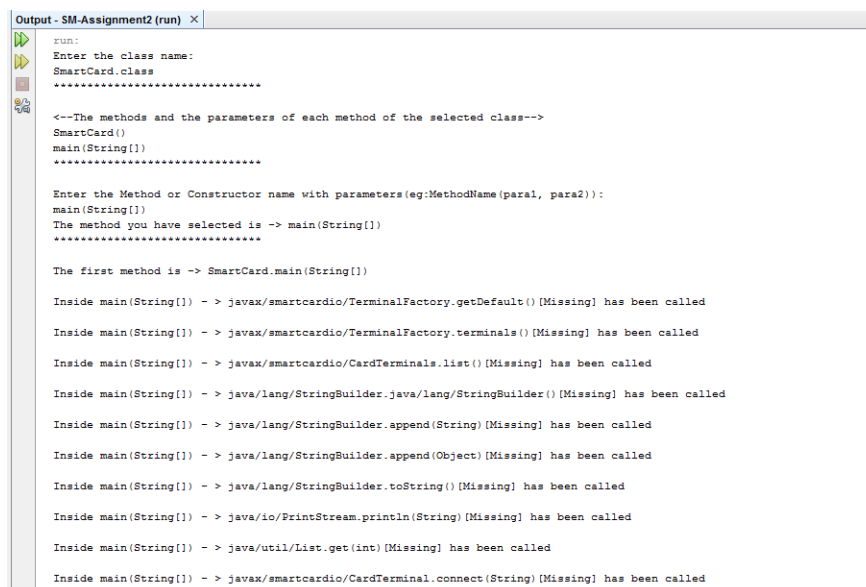
Steps to be executed - Enter the class name and method name.

Expected results - Call tree for the entered method.

- For the test 1 call tree should be displayed for the main(String[]).

Actual results - Call tree for the main(String[]).

Pass/fail status — Pass



```
run:
Enter the class name:
SmartCard.class
.....

<--The methods and the parameters of each method of the selected class-->
SmartCard()
main(String[])
.....

Enter the Method or Constructor name with parameters (eg:MethodName(para1, para2)):
main(String[])
The method you have selected is -> main(String[])
.....

The first method is -> SmartCard.main(String[])

Inside main(String[]) - > javax/smartcardio/TerminalFactory.getDefault() [Missing] has been called
Inside main(String[]) - > javax/smartcardio/TerminalFactory.terminals() [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CardTerminals.list() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.java/lang/StringBuilder() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(Object) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.toString() [Missing] has been called
Inside main(String[]) - > java/io/PrintStream.println(String) [Missing] has been called
Inside main(String[]) - > java/util/List.get(int) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CardTerminal.connect(String) [Missing] has been called
```



```
Output - SM-Assignment2 (run) X
Inside main(String[]) - > java/lang/StringBuilder.java/lang/StringBuilder() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(Object) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.toString() [Missing] has been called
Inside main(String[]) - > java/io/PrintStream.println(String) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/Card.getBasicChannel() [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CommandAPDU(javax/smartcardio/CommandAPDU(int, int, int, int, byte[]) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CardChannel.transmit(Ljavax/smartcardio/, charommandAP, doubleU;) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CommandAPDU(javax/smartcardio/CommandAPDU(int, int, int, int) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/CardChannel.transmit(Ljavax/smartcardio/, charommandAP, doubleU;) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.java/lang/StringBuilder() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/ResponseAPDU.toString() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.toString() [Missing] has been called
Inside main(String[]) - > java/io/PrintStream.println(String) [Missing] has been called
Inside main(String[]) - > javax/smartcardio/ResponseAPDU.getData() [Missing] has been called
Inside main(String[]) - > java/io/PrintStream.print(char) [Missing] has been called

-----
Inside main(String[]) - > java/io/PrintStream.println() [Missing] has been called
Inside main(String[]) - > javax/smartcardio/Card.disconnect(boolean) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.java/lang/StringBuilder() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > java/lang/Exception.toString() [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.append(String) [Missing] has been called
Inside main(String[]) - > java/lang/StringBuilder.toString() [Missing] has been called
Inside main(String[]) - > java/io/PrintStream.println(String) [Missing] has been called

Number of unique METHODS are: 1
[main(String[])]

Number of unique CONSTRUCTORS are: 0
[]

Number of unique CLASSES are: 12
[SmartCard, javax/smartcardio/TerminalFactory, javax/smartcardio/CardTerminals, java/lang/StringBuilder, java/io/PrintStream, jav

BUILD SUCCESSFUL (total time: 5 seconds)
```

Figure 13

Quality Analysis

The quality analysis was done using the RSM tool. After performing quality analysis several quality issues were identified and some issues were fixed. The quality analysis of a source code will provide the details of what standards are being followed how adaptive the source code is and the details of readability, maintainability. There are number of predefined metrics that are used to measure the quality of a product.

The issues identified from a quality analysis are not code issues but the issues of how the quality is maintained. Therefore, some quality issues may or may not be removed.

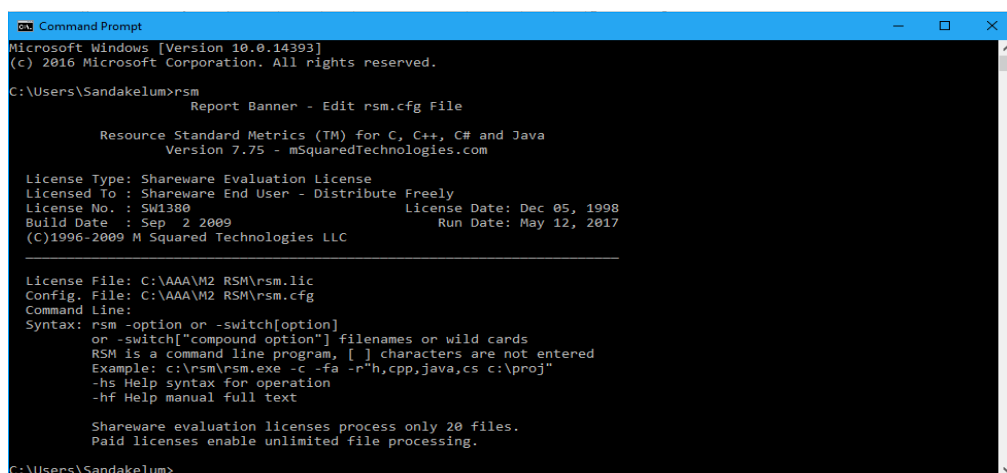
In the source code that I have developed there were several quality issues identified. And some issues have been removed.

What is RSM

RSM is a commercially available tool which is used to measure the code quality of a source code. RSM provides standard methods for analysing C, ANSI C++, C# and Java source code across operating systems.

In this assignment, the RSM tool is used to measure the code quality of a JAVA source code. Quality issues, heap usage, caller and callee method details, Cyclometric Complexity etc. can be gathered from the RSM tool.

RSM Installed



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Sandakelum>rsm
Report Banner - Edit rsm.cfg File

Resource Standard Metrics (TM) 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: May 12, 2017
(C)1996-2009 M Squared Technologies LLC

License File: C:\AAA\M2 RSM\rsm.lic
Config. File: C:\AAA\M2 RSM\rsm.cfg
Command Line:
Syntax: rsm -option or -switch[option]
or -switch["compound option"] filenames or wild cards
RSM is a command line program, [ ] characters are not entered
Example: c:\rsm\rsm.exe -c -fa -r"h,cpp,java,cs c:\proj"
-hs Help syntax for operation
-hf Help manual full text

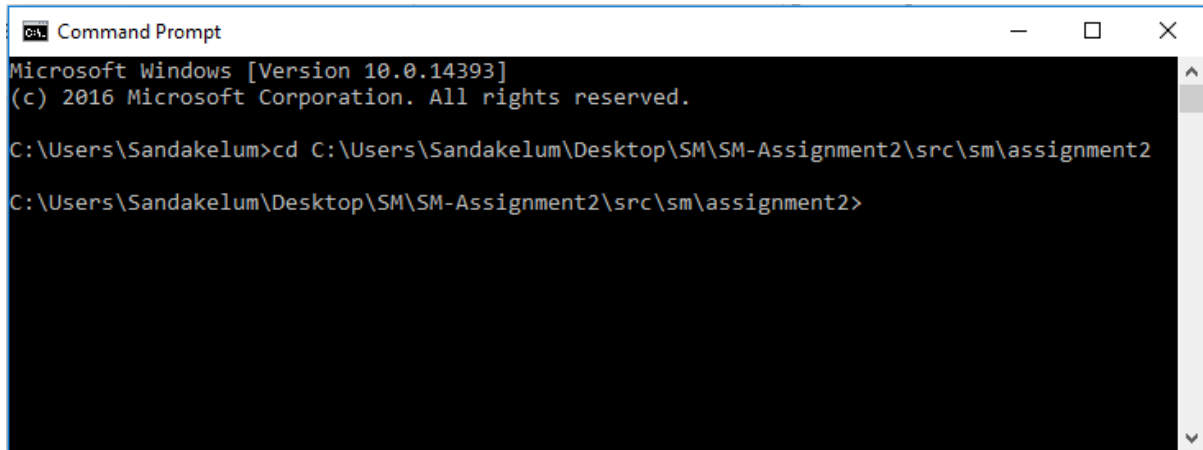
Shareware evaluation licenses process only 20 files.
Paid licenses enable unlimited file processing.

C:\Users\Sandakelum>
```

Figure 14

Steps to be followed to create the RSM quality report

- Globally install the RSM tool as showed in the above figures.
- Locate the file path where the java file is existed.

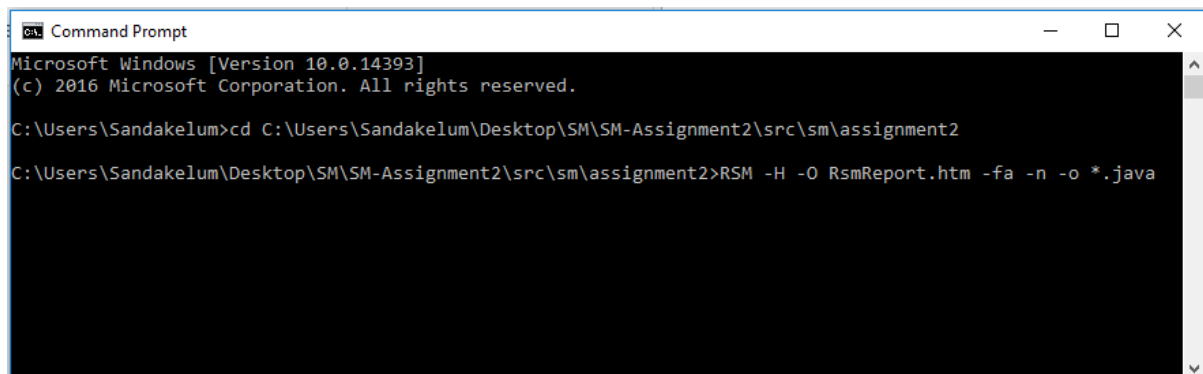


```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Sandakelum>cd C:\Users\Sandakelum\Desktop\SM\SM-Assignment2\src\sm\assignment2
C:\Users\Sandakelum\Desktop\SM\SM-Assignment2\src\sm\assignment2>
```

Figure 15

- Run the command to get the RSM quality report.



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Sandakelum>cd C:\Users\Sandakelum\Desktop\SM\SM-Assignment2\src\sm\assignment2
C:\Users\Sandakelum\Desktop\SM\SM-Assignment2\src\sm\assignment2>RSM -H -O RsmReport.htm -fa -n -o *.java
```

Figure 16

- For each and every java file there is a separate quality report created. The quality report is saved at the directory path which

C:\Users\Sandakelum\Desktop\SM\RSMReports

The folder names are created with the java file names. Each folder contains the quality report of the java file which the folder name represents.

Code quality issues

A quality issue raised from SMAssignment2.java file

```
----- Class Begin Line: 21 -----  
Class: sm.assignment2.SMAssignment2  
  
Notice #52: Line 21: 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 #35: Line 23: Public data has been identified with a class  
specification. This design breaks object encapsulation  
allowing free access from other objects and functions.  
  
Notice #35: Line 24: Public data has been identified with a class  
specification. This design breaks object encapsulation  
allowing free access from other objects and functions.  
  
Notice #35: Line 25: Public data has been identified with a class  
specification. This design breaks object encapsulation  
allowing free access from other objects and functions.  
  
Notice #35: Line 26: Public data has been identified with a class  
specification. This design breaks object encapsulation  
allowing free access from other objects and functions.  
  
----- Function Begin Line: 31 -----  
Function: sm.assignment2.SMAssignment2.main
```

Figure 17

SMAssignment2 two act as a primary class and because it contains public data. In a primary class data should not become unprotected.

But because I have to use those public data all over the entire project it is being declared with public access modifier.

A quality issue raised from SMAssignment2.java file

```
----- Function Begin Line: 31 -----  
Function: sm.assignment2.SMAssignment2.main  
Parameters: (String[] args)  
  
Notice #38: Line 44: 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 56: Line character length = 117. This width  
exceeds the standard terminal width of 80 characters.  
  
Notice #38: Line 61: 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.
```

Figure 18

Try catch is used to handle the exceptions. But in best practice exception handling should do in the design of the program not in the implementation.

But there are situations that that exception handling is required even in the implementation. Therefore, above quality issues cannot be removed.

A quality issue raised from CodeAttribute.java file

```
exceeds the standard terminal width of 80 characters.

----- Function Begin Line: 42 -----
Function: sm.assignment2.CodeAttribute.CodeAttribute
Parameters: (DataInputStream dis, ConstantPool constantPool, Class c, boolean abstractType, String methName)

Notice #51: Line 42: 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 45: Line character length = 83. This width exceeds the standard terminal width of 80 characters.

Notice #50: Line 47: A variable has been identified which is assigned to a literal number. Symbolic constants should be used to enhance maintainability.

Notice #50: Line 51: A variable has been identified which is assigned to a literal number. Symbolic constants should be used to enhance maintainability.

Notice #38: Line 52: 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.
```

Figure 19

here i have used a for loop and inside the for loop the starting point has been defined by assigning a literal number to a variable.

without directly assigning a literal number to a variable it can be done by using Symbolic constants.

A quality issue raised from MethodInfo.java file

```
Notice #1: Line 130: Line character length = 112. This width
exceeds the standard terminal width of 80 characters.

Notice #1: Line 133: Line character length = 113. This width
exceeds the standard terminal width of 80 characters.

Notice #1: Line 136: Line character length = 116. This width
exceeds the standard terminal width of 80 characters.

Notice #55: Line 139
The depth of scope '6' exceeds the notice limit of 5.

Notice #55: Line 141
The depth of scope '6' exceeds the notice limit of 5.

Notice #1: Line 142: Line character length = 118. This width
exceeds the standard terminal width of 80 characters.

Notice #17: Function comments, 0.9% are less than 10.0%.

Notice #46: Function blank line percent, 0.9% is less than 10.0%.

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

Figure 20

here the quality notice 55 is occurred, which is saying that the depth of scope '6' exceeds the notice limit of 5. because the method ParameterTypes() contains nested if-else statements.

But those statements cannot be removed due to the reason of checking parameter types.

A quality issue raised from FindNextClass.java file

```
----- Function Begin Line: 35 -----
Function: sm.assignment2.FindNextClass.FindNextClass
Parameters: ()

Notice #51: Line 35: 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.

Notice #48: The number of logical lines of code or statements
0 is <= to the specified minimum of 0. This could
indicate a null function with no operational purpose.

Function: sm.assignment2.FindNextClass.FindNextClass
```

Figure 21

here the quality notice 48 is occurred, which is saying that the number of logical lines of code or statements 0 is \leq to the specified minimum of 0. This could indicate a null function with no operational purpose.

And it is occurred because I have defined constructor(FindNextClass()) with no code segment inside it.

The identified bugs

- The program cannot identify the abstract methods.

Even though I have implement the code to identify the abstract methods. While printing there is a method mismatch occurred. The problem occurs while printing the methods because the printing is done before the abstract method is identified. Therefore, it is impossible to print abstract methods after identified.

- One scenario of recursion cannot be identified

Recursion programs can be identified only when a method call the same method inside itself. But if a method calls another method and that method again call the previous method then the recursion cannot be identified. But it will execute only one time. Because it is identified as same method call again and again.

Eg : TheClass.methodA()

TheClass.methodA() [recursive]

This scenario can be identified.

Eg: TheClass.methodB()

TheClass.methodC()

TheClass.methodD()

TheClass.methodB() [recursive]

This scenario cannot be identified.

- The code recognized many kind of parameters but when a custom parameter found other than primitive types there can be conflicts occurred.

Eg: parameter type - class Directory

Reference

Reflection functionalities

<https://docs.oracle.com/javase/8/docs/api/index.html>

Decimal to Hexadecimal converter

<http://www.binaryhexconverter.com/decimal-to-hex-converter>

Java decompile online

<http://www.javadecompilers.com/>

Java class file data decompile online

http://www.mobilefish.com/services/java_decompiler/java_decompiler.php

class file structure

<https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html>

RSM Tool

<http://msquaredtechnologies.com/m2rsm/>

http://msquaredtechnologies.com/m2rsm/docs/rsm_analysis.htm

Test case scenarios

SM lab sessions in Sri Lanka Institute of information technology

Real world test cases: -

Test02 - <https://github.com/code-lover/java/blob/master/SmartCardDemo/src/SmartCard.java>