/\*\*

\*

\* File: CMSC 350 project 4

\* Author: Bedemariam Degef

\* Date: December 13, 2019

\* Purpose: This program builds a directed graph from a \* list of classes from a file and mimics a

\* java compiler and output the recommended \* list of class to execute in a topological \* order

\*

\*/

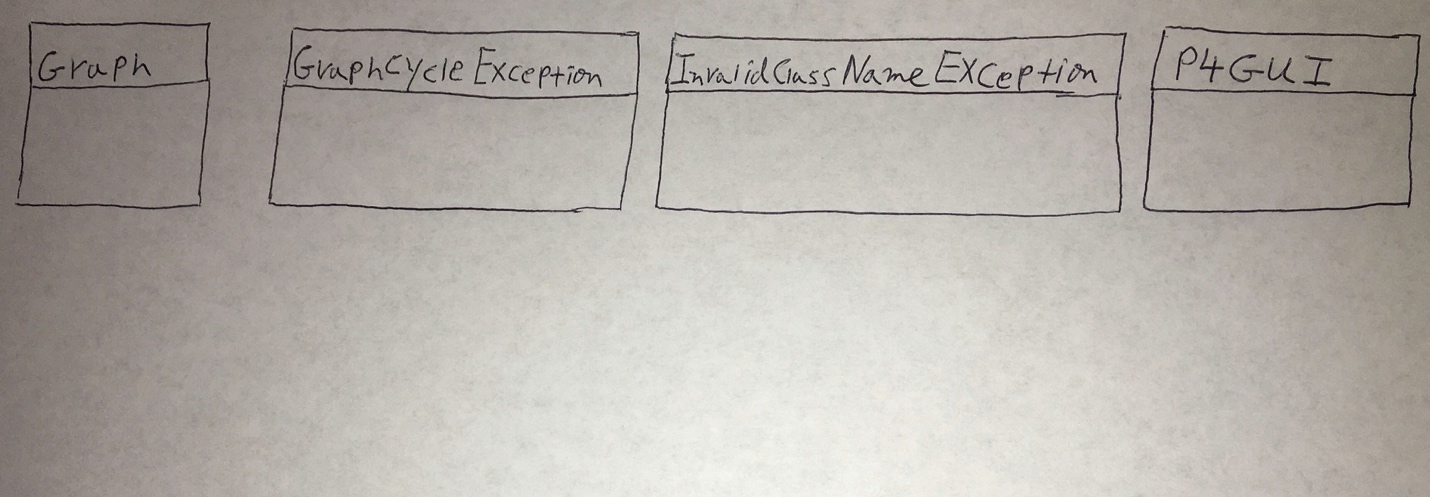
1. **Assumptions, main design decisions, error handling;**

My assumption after reading the instruction was that this project is implemented using directed graph data structure. It will consist two class to build the GUI and the directed graph. There will be two more classes for user defined exceptions. The graph is going to be constructed by reading the Strings needed from a file. Topological order algorithm is used to generate recommendation list in the output.

There are going to be four classes. The first class is P4GUI.java which will define and implement the GUI. In addition, it will also handle exceptions that are thrown. The second class is Graph.java, and this class will build a directed graph from a file and preform tropological order sort in order get the appropriate output values. In the graph, integer values will be used to represent the vertices instead of Strings. The graph will represent as an array list of integer value vertices that contain a linked list of their associated adjacency lists. The adjacency lists will be the lists of integers that represent the index rather than vertex name itself. A hash map will be used to associate vertex names with their index in the list of vertices. The rest of two classes are GraphCycleException.java for when the graph contains cycles and InvalidClassNameException.java for invalid class names.

When exceptions are thrown by Graph.java class, they are caught and handled in the P4GUI.java class. If there are cycles in the graph GraphCycleException is thrown and handled. For invalid class name InvalidClassNameException is thrown and handled. If the file didn’t open, IOException is thrown and handled.

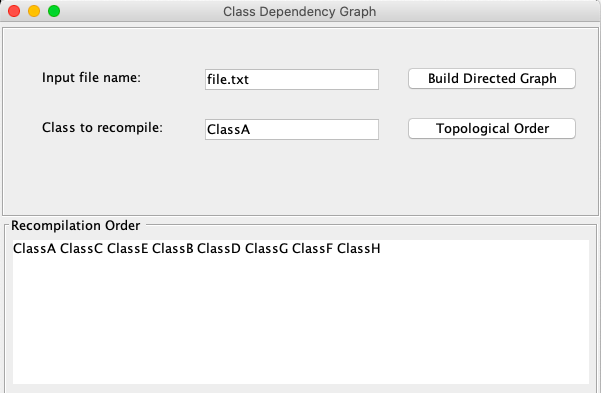
1. **A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods;**

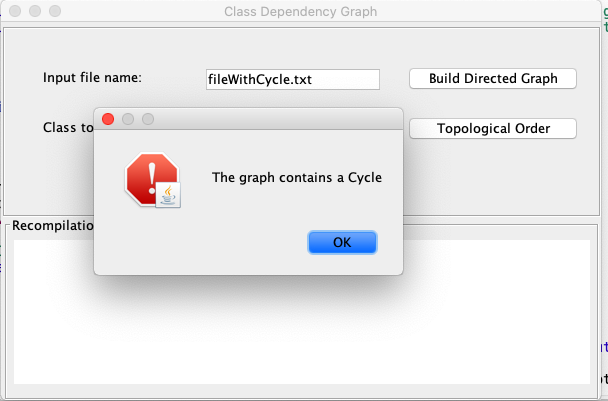
****

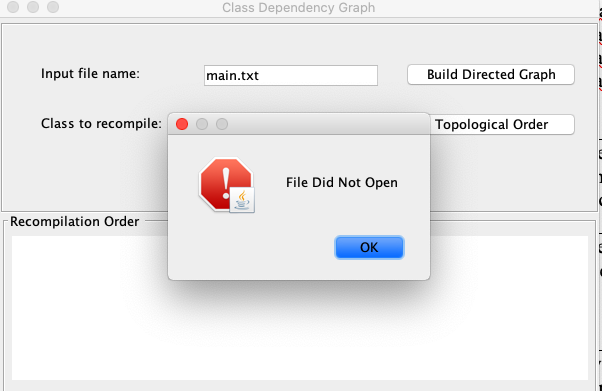
1. **A table of test cases including the test cases that you have created to test the program. The table should have 5 columns indicating (i) what aspect is tested, (ii) the input values, (iii) the expected output, (iv) the actual output and (v) if the test case passed or failed. Each test case will be defined in a table row.**

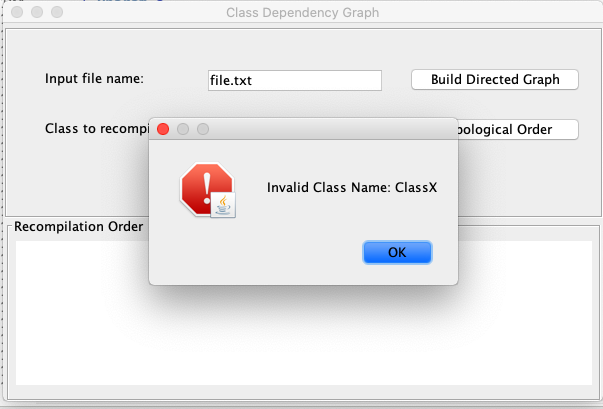
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What aspect is tested** | **The input values** | **The expected output** | **The actual output** | **Pass or fail** |
| Correct topological order for all cases without cycles | Filename: file.txt  Class: ClassA | ClassA ClassC ClassE ClassB ClassD ClassG ClassF ClassH | ClassA ClassC ClassE ClassB ClassD ClassG ClassF ClassH | Passed |
| Error message for all cases with cycles | Filename: fileWithCycle.txt  Class: ClassE | The graph contains a Cycle | The graph contains a Cycle | Passed |
| Error message when file does not open | Filename: main.txt  Class: ClassC | File Did Not Open | File Did Not Open | Passed |
| Error message when invalid class name is entered | Filename: file.txt  Class: ClassX | Invalid Class Name: ClassX | Invalid Class Name: ClassX | Passed |
| Message confirming that graph has been built | Filename: file.txt  Class: ClassD | Graph Built Successfully | Graph Built Successfully | Passed |

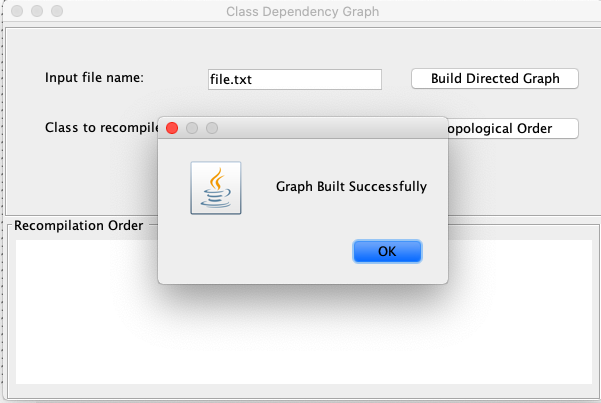
1. **Relevant screenshots of program execution;**

****



****

****

****

**(5) Lessons learned from the project;**

I have gained a better understanding of graph data structures. I have learned how they are implemented and how we can use depth first search to check for cycles in a graph and use topological order algorithm to obtain a linear list form a graph. I have also learned how java compiler class execution works at a higher level.