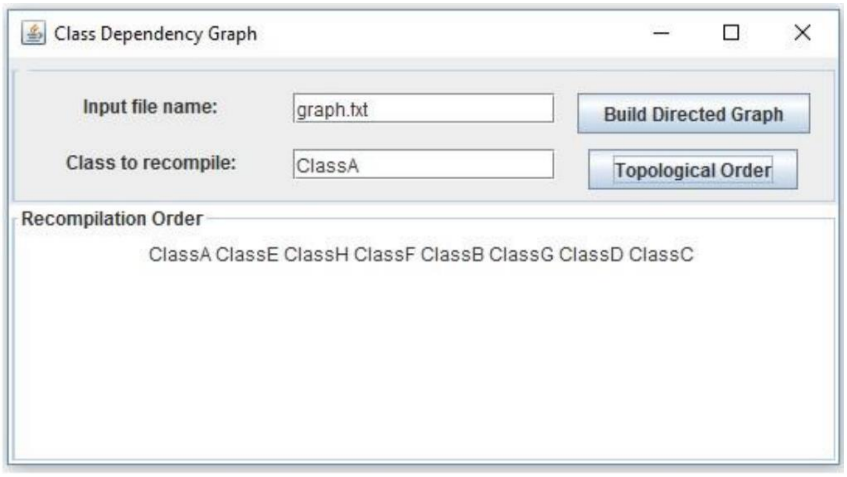
UMUC
CMSC 350 Project 4

1. Specification

The fourth programming project involves designing, writing and testing a program that behaves like the Java command line compiler. Whenever we request that the Java compiler recompile a particular class, it not only recompiles that class but every other class that depends upon it, directly or indirectly, and in a particular order. To make the determination about which classes need recompilation, the Java compiler maintains a directed graph of class dependencies. Any relationship in a UML class diagram of a Java program such as inheritance relationships, composition relationships and aggregations relationships indicate a class dependency.

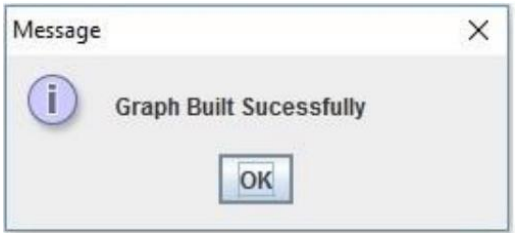
The main class **P4GUI** should create the Swing based GUI shown below:



The GUI must be generated by code that you write. You may not use a drag-and-drop GUI generator.
Pressing the *Build Directed Graph* button should cause the specified input file that contains the class dependency information to be read in and the directed graph represented by those dependencies to be built. The input file should be generated by the students using a simple text editor such as Notepad. The input file associated with the above example is shown below:

```
ClassA ClassC ClassE
ClassB ClassD ClassG
ClassE ClassB ClassF ClassH
ClassI ClassC
```

Each line of this file specifies classes that have other classes that depend upon them. The first line, for example, indicates that `ClassA` has two classes that depend upon it, `ClassC` and `ClassE`. In the context of recompilation, it means when `ClassA` is recompiled, `ClassC` and `ClassE` must be recompiled as well. Using graph terminology, the first name on each line is the name of a vertex and the remaining are its associated adjacency list. Classes that have no dependent classes need not appear at the beginning of a separate line. Notice, for example, that `ClassC` is not the first name on any line of the file.
After pressing the *Build Directed Graph* button, one of following two messages should be generated depending upon whether the specified file name could be opened:



Once the graph has been built, the name of a class to be recompiled can be specified and the *Topological Order* button can be pressed. Provided a valid class name has been supplied, a topological order algorithm should be executed that will generate (in the text area at the bottom of the GUI window) the list of classes in the order they are to be recompiled. The correct recompilation order is any topological order of the subgraph that emanates from the specified vertex.

An invalid class name should generate an exception and an appropriate error message will be displayed in a JOptionPane. If the graph contains a cycle among the Java classes, an exception will be thrown and a message will be displayed in a JOptionPane indicating that a cycle has been detected.

Note. In the real compiling processes, when circular dependencies exist in Java programs, the compiler must make two passes over all the classes in the cycle. For this program, it will be sufficient to display a message indicating that a cycle has been detected.

In addition to the main class that defines the GUI, a second class is needed to define the directed graph. It should be a generic class allowing for a generic type of its vertices. In this application the type of the vertices will be String.

For better processing purposes, integer values will be used to represent the vertices instead of Strings. The graph should be represented as an array list of vertices (as integer values) that contain a linked list of their associated adjacency lists. The adjacency lists should be lists of integers that represent the index rather than vertex name itself. A hash map should be used to associate vertex names with their index in the list of vertices. For the input file shown above the array list of linked lists of integers would be the following:

```
0 [1, 2]
1 []
2 [3, 6, 7]
3 [4, 5]
4 []
5 []
6 []
7 []
8 [1]
```

Storing the vertex indices rather than the names simplifies the topological order algorithm. The hash map would associate index 0 with `ClassA`, index 1 with `ClassC`, index 2 with `ClassE` and so on.

The directed graph class should define methods for initializing the graph each time a new file is read in, for adding a vertex and an edge to the graph and for generating a topological order given a starting index. Other classes / methods could be also defined to achieve the program requirements and for better structuring the code.

Finally, custom checked exception classes should be defined for the cases where a cycle occurs and when an invalid class name is specified.

Your program should compile without errors.

The Google recommended Java style guide (https://google.github.io/styleguide/javaguide.html) should be used to format and document your code. Specifically, the following style guide attributes should be addressed: header comments include filename, author, date and brief purpose of the program; In-line comments used to describe major functionality of the code; the meaning and the role of variables and constants are indicated as code comments; meaningful variable names and prompts applied; class names are written in UpperCamelCase; variable names are written in lowerCamelCase; constant names are in written in All Capitals; braces use K&R style.

In addition the following design constraints should be followed: declare all instance variables private; avoid the duplication of code.

2. Submission requirements

Submit the following to the Project 2 assignment area no later than the due date listed in your LEO classroom.

1. All **.java** source files (**no other file types should be submitted)**. The source code should use Java code conventions and appropriate code layout (white space management and indents) and comments. All submitted files may be included in a .zip file.

2. The solution description document **P3SolutionDescription** (.pdf or .doc / .docx) containing the following:
(1) Assumptions, main design decisions, error handling;
(2) 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;
(3) A table of test cases including the test cases that you have created to test the program. The table has 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.
(4) Relevant screenshots of program execution;
(5) Lessons learned from the project;

Grading Rubric:

Criteria	Meets	Does Not Meet
Design	**5 points**	**0 points**
	GUI is hand coded and matches required design	GUI is generated by a GUI generator or does not match required design
	Includes generic class for a directed graph	Does not include a generic class a directed graph
	Graph is represented as an array list of vertices that contain a linked list of their associated adjacency lists	Graph is not represented as an array list of vertices that contain a linked list of their associated adjacency lists
	Includes checked exception classes for cycles and invalid class names	Does not Include checked exception classes for cycles and invalid class names
	Uses good object-oriented design practice regarding code efficiency, encapsulation and information hiding, class and code reuse, high cohesion of classes, avoiding code duplication.	Does not use good object-oriented design practice regarding code efficiency, encapsulation and information hiding, class and code reuse, high cohesion of classes, avoiding code duplication.
Functionality	**10 points**	**0 points**
	Produces correct topological order for all cases without cycles	Does not produce correct topological order for all cases without cycles
	Produces error message for all cases with cycles	Does not produce error message for all cases with cycles
	Reports error message when file does not open	Does not report error message when file does not open
	Reports error message when invalid class name is entered	Does not report error message when invalid class name is entered
	Generates message confirming graph has been built	Does not generate message confirming graph has been built
	5 points	**0 points**
	Test cases table is defined and included in the P4SolutionDescription document	Test cases table is not defined and included in the P4SolutionDescription document
	Test cases include a graph without cycles	Test cases do not include a graph without

Test Cases		cycles
	Test cases include a graph with cycles	Test cases does not include a graph with cycles
	Test cases include an invalid file name	Test cases do not include an invalid file name
	Test cases include an invalid class name	Test cases do not include an invalid class name
Documentation	**5 points**	**0 points**
	Solution description document P4SolutionDescription includes all the required sections (appropriate titled).	No solution description document is included
	Source code follows Google recommendation Java style	Source code does not follow Google recommendation Java style
	Comment blocks with class description included with each class	Comment blocks with class description not included with each class
	Source code is commented and indented	Source code is not commented and indented
Overall Score	**Meets**	**Does not meet**
	16 or more	15 or less