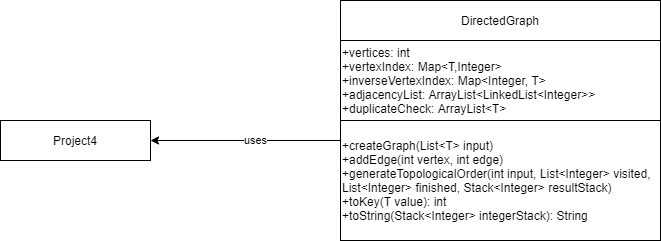
# CMSC350 Project 4 Documentation

By Brian Yu

## Goal

The goal of this project was to develop a program that simulates a Java command line compiler. The program reads in a file that lists all the class dependencies which the program uses to build a directed graph. The program then performs a depth-first traversal to collect all the class dependencies and then displays the result. The program will display an error for file issues, cycles in the directed graph, and incorrect class names.

## UML Diagram

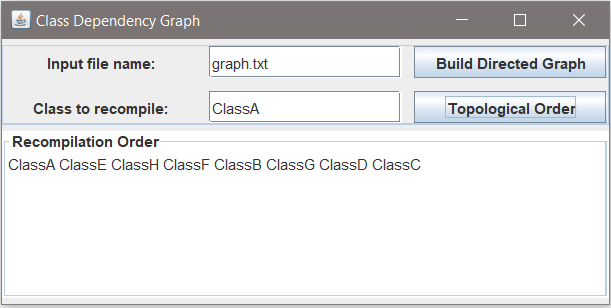
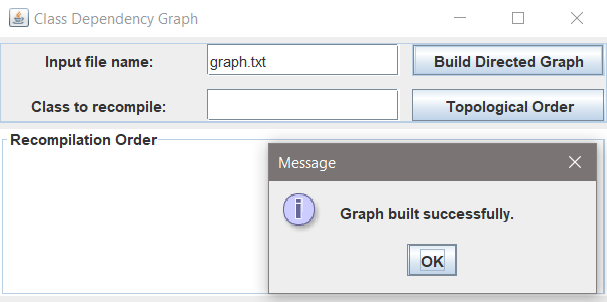


## Test Cases

I performed based functional testing to make sure my program could handle the expected uses cases of a Java command line compiler simulator. For error cases, I specifically tested to make sure the simulator threw an error for file errors, cycles in the directed graph, and invalid class names. In these test cases, when I entered in the file name, I hard coded in the path prefix “C:\temp\” into my code in order to prevent the need for typing that every time. I removed it for my final submission so other users can direct the program to the exact location of their file.

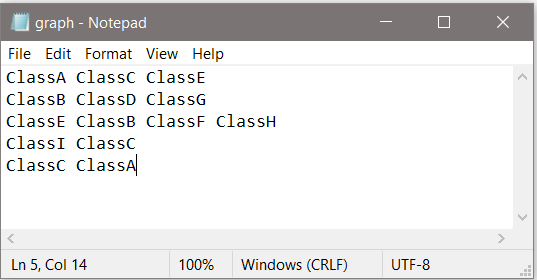
### Test Case 1

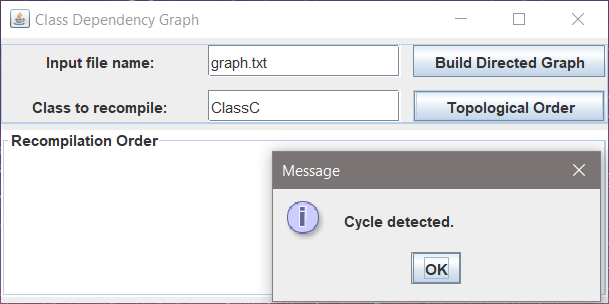
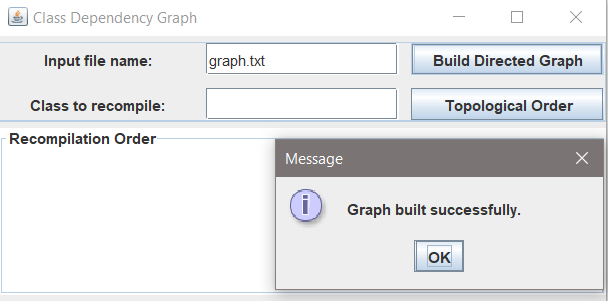
This test case inputted the example file that was listed in the Project 4 instructions. I entered in ClassA to recompile. The recompilation order returned was “ClassA ClassE ClassH ClassF ClassB ClassG ClassD ClassC” as expected. This test case passes.



### Test Case 2

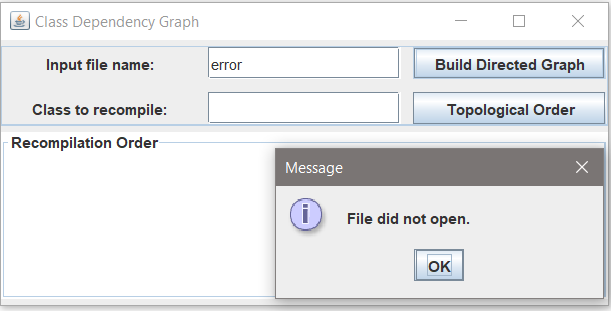
This test case inputted the example file that was listed in the Project 4 instructions with the line “ClassC ClassA” added. I entered in ClassC to recompile. The program returned a “Cycle detected” error as expected. This test case passes.





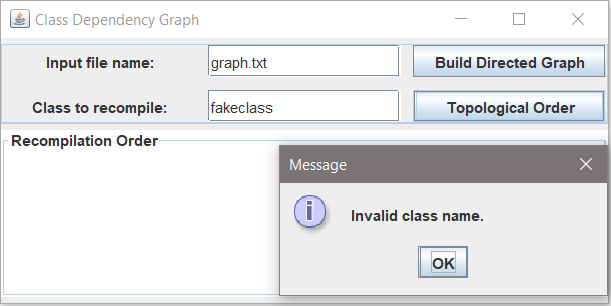
### Test Case 3

This test case inputted an invalid file name in the “Input filed name:” field. The program returned a “File did not open” as expected. This test case passes.



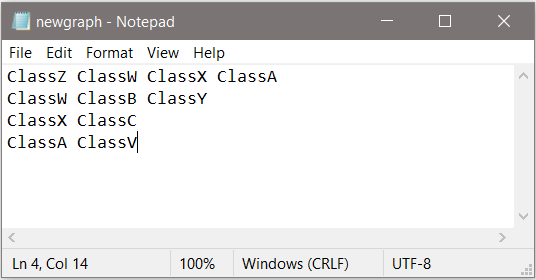
### Test Case 4

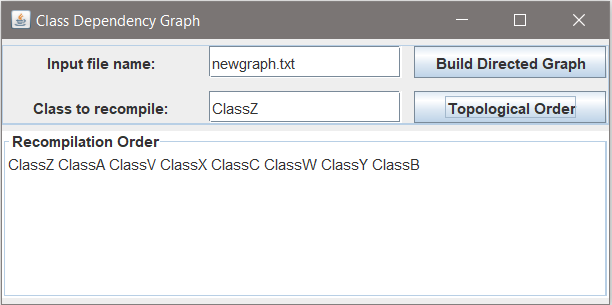
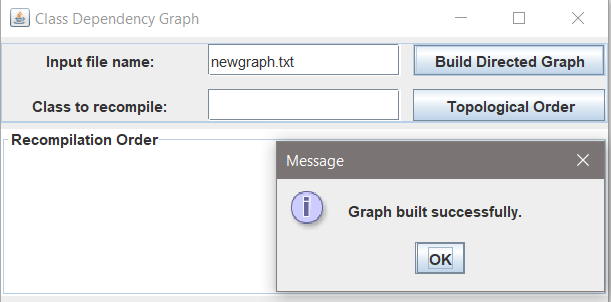
This test case inputted the example file that was listed in Project 4 instructions. I entered an invalid class name to recompile. The program returned a “Invalid class name” error as expected. This test case passes.



### Test Case 5

This test case inputted a file that I created. I entered in ClassA to recompile. The recompilation order was “ClassZ ClassA ClassV ClassX ClassC ClassW ClassY ClassB” as expected. This test case passes.





## Lessons Learned

I had the most trouble with this project out of all the projects in the entire class. Most of the issues surrounded making the DirectedGraph class generic and constructing the hashmap and directed graph from the file input.

Initially my strategy to construct the hashmap and the directed graph from the input file was to first read through the file to populate the hashmap with all the unique vertices and then read through the file again to populate the adjacency list. This created a lot of code duplication as I had to run through the file twice. However, I initially had problems trying to write both the hashmap and the adjacency list by going through the input file one line at a time because I had to insert linkedlists into the adjacency list arraylist at specific rows which is not a function supported by arraylists. Eventually I was able to overcome this by populating the arraylist with empty linkedlists while constructing the hashmap.

I also ran into a bunch of issues trying to make the DirectedGraph class generic. Part of this was due to the fact I didn’t really understand the need, Java is best designed around extracting info from input files as Strings, as we were doing for the project. Due to type compatability, I eventually had to move all the file reading lines out of the DirectedGraph class and into the Project4 class in order to maintain the ability to accept generic type parameters in my createGraph method.

Finally, I was forced to create additional methods in addition to the required 3 in order to catch the incorrect class name error and to print out the recompilation order as a String. I originally though that the hashmap would return the value tied to the key if the key was passed to the get command, however I found that it didn’t work that way and needed to create a reverse hashmap if I wanted to do that. I originally thought about calling on this during the depth first traversal method, but I realized I only really needed to do so when printing out the recompilation order result as a string.