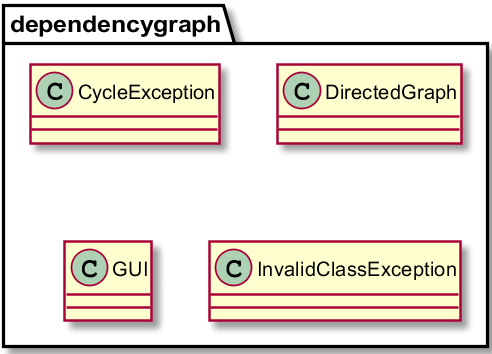
CMSC 350 – Project 4: Directed Graph

UML for written classes: 1 – 1

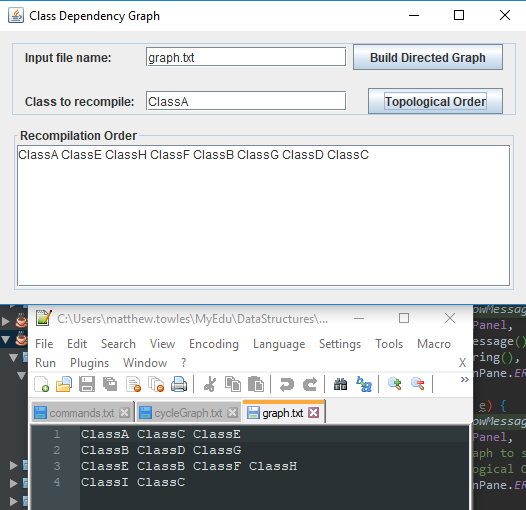
Test Plan: 2 – 5

Lessons Learned: 6 – 6

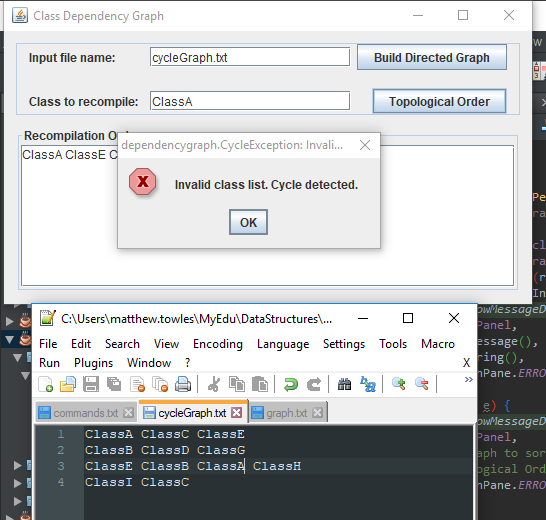
1. **UML**



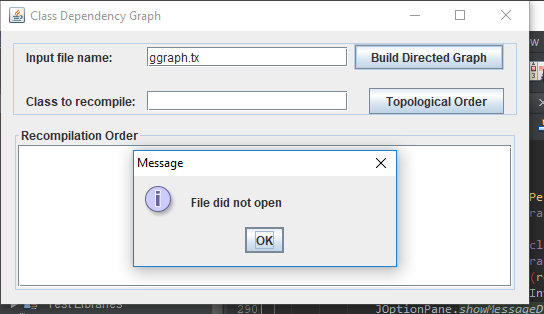
1. **Test Plan**
2. **Graph without cycles:**



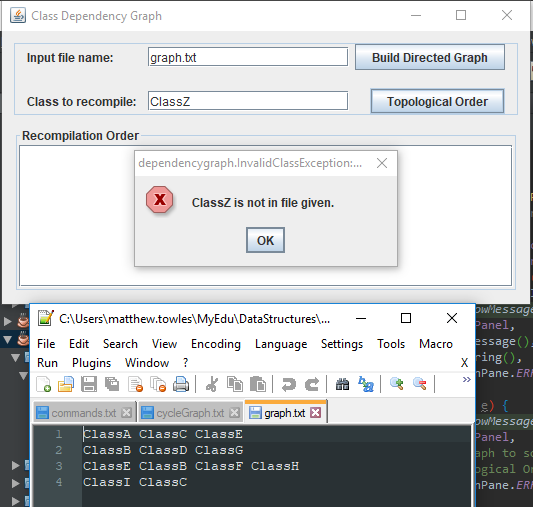
1. **Graph with cycles:**



1. **Invalid file name:**



1. **Invalid class name:**



1. **Lessons Learned**

When I began planning this out, I had defined a vertex class. However, as I moved along with the requirements given, it became apparent to me that this may not be the best way to move forward given the requirements. I came to this decision because the way we were handling the vertices via hash map, array list of linked lists and using a stack. I thought it might actually make the code unnecessarily complex without adding any real value.

There were really two main problems to solve here (one of which was greatly aided by the pseudocode given in the requirements.) First, what is a directed graph and how do we make it? I used a hash map to map the names of vertexes as they are in the file to the integer order in which they were processed. I also made a second hash map to do the reverse which was needed later. I used an array list of linked lists – the key to each linked list corresponded to the hash map value of the vertex.

The next problem was the DFS topological sort. After making sure that the vertex given is valid and was not already discovered/processed by the sort function, we check for the base case which is if the vertex stack has the vertex in it. In the event that the base case is not reached: add the vertex to the discovered vertices, get it’s linked list/adjacency list, loop through that list and call the sort function on each of those vertices. Finally, push the vertex to the stack. After all of this, we call another function to create a string from popping the stack and output to the GUI.