Assignment overview:

In this assignment we'll be making some modifications to our Graph class to support Topological Sorting on directed acyclic graphs (DAGs).

THIS LAB BUILDS ON LAB 7.

If your Lab 7 was successful and you're happy with your code, you may use it as the starting point for this lab.

Alternatively, you may use *my code*, available on the Learning Hub. See `Main.java` and `Graph.java` there.

- **Note 1**: The `main()` function in that code sets up and runs Lab7 stuff on a bunch of graphs that *are not* the same graphs as the ones in this assignment.
- **Note 2**: The BFS code is not needed for this lab, but I left mine in `Graph.java` just in case you want to see it. I don't care if you keep it in yours or not.

How, what, and when to submit:

Please submit the following to the dropbox on Learning Hub:

- Java file containing your Graph class, meeting the requirements given below
- Java file containing your Main class, meeting the requirements given below

Please do not zip or compress your files (tempting though it may be).

This lab is worth 20 points.

It is due at midnight next Wednesday, Nov 10.

Summary of requirements:

1. Modify the Graph class so that it contains labels for the vertices. I suggest keeping this simple: an array or ArrayList of Strings as a private member in your class.

2. If it was not already this way, modify your DFS so that it runs "silently". For silent operation, do not display any output while the DFS operation is being performed. Instead, store the sequence of visited vertices in some kind of data structure (another private member) within the Graph class. (ArrayList works nicely here.)

3. Provide public method(s) to access and display this DFS-ordered sequence of vertices from your main program. This output should be labeled as DFS results. When you output the vertices, you should display the *labels* of the vertices, not the indices.

4. Modify your Graph class so that it computes a topological sort ordering for the graph. The result of the topological sort is another sequence of vertices, stored within the class as you did with DFS. Use the "TopoSort Algorithm 1" from the lecture notes – a modification of DFS. Ideally, you should modify the DFS that already exists, so that it will find the DFS results and TopoSort results at the same time. Don't forget that

TopoSort is the *reverse* of the "dead-end order" that results from doing DFS.

5. Provide another public method(s) to access/display the TopoSort results. This output should be labeled as the topological sort results.

6. Add any other public or private helper methods that you find you need to support all of the above requirements.

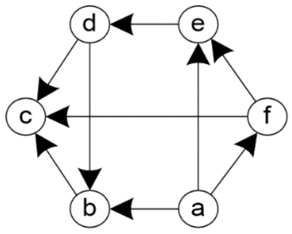
7. In your main() driver program, you should do ALL of the following for ALL of the example graphs mentioned in this handout:
    a. Construct the graph including vertex labels
    b. Run DFS on the graph
    c. Print the DFS results
    d. Run TopoSort on the graph (if needed separately from DFS)
    e. Print the TopoSort results

   Any time you display a vertex in output for *any* reason, it should be the vertex *label* (string) that you display, not the vertex *index* (0 to n-1).

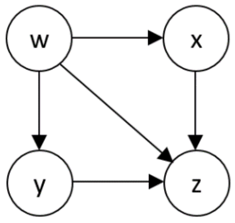
Example graphs

There are 5 graphs here.

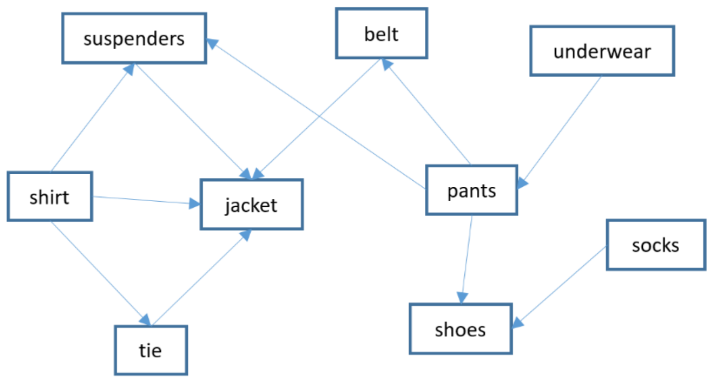
Graph 1: The 6-vertex DAG example on the definitions page in the Topo Sort section of the lecture notes. Vertices are {a,b,c,d,e,f}. This also appears as Example 1 later in the section.



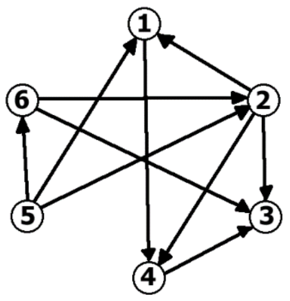
Graph 2: The 4-vertex DAG example on the definitions page, vertices are {w,x,y,z}.



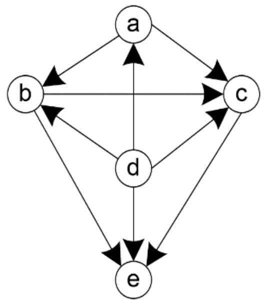
Graph 3: The “how to get dressed” graph.



Graph 4: Example 2 of the lecture notes, with vertices {1,2,3,4,5,6} ("strings").



Graph 5: Example 3 of the lecture notes, with vertices {a,b,c,d,e}.



Bonus entertainment:

This is not a required part of the assignment.

If you are looking for a little extra coding just for fun or practice, how about adding a constructor to your Graph class that takes a *filename* (String) as an argument, reads the data file which contains a description of a graph, and constructs the corresponding graph?

If you do this, be sure to submit your data files to Learning Hub along with your code.

A suggested format of a graph data file is:

- First line: an integer—the number of vertices N

- Next N lines: the labels of the vertices (these are strings, even if they happen to look like numbers). These should be stored *in the order that they are listed,* because the edge data will refer to these vertices by their index numbers.
- Next line: another integer—the number of edges K
- Next K lines: each contains two *integers* which are the indices of the "from" and "to" vertices for one edge

Here is a sample data file you can use for the "clothing graph" (Graph 3) in this handout:

```
9
suspenders
belt
underwear
shirt
jacket
pants
socks
tie
shoes
11
6 8
2 5
3 0
3 4
3 7
5 8
5 1
5 0
7 4
0 4
1 4
```

Notice that this data file format does not specify whether the graph is directed or undirected. The same format can work for both types, but since we are doing Topological Sorting, your program can simply assume that the data file is to be treated as a directed graph.