Exercise 2

1 Objective

Practice Java generics, exceptions, and interfaces

2 Marking

This exercise will be graded of of 36 marks, as explained in detail in the starter code.

It is worth 3% of your final grade.

The submission deadline is June 17, 2016, 11:50pm.

This is an individual exercise.

Late submission policy: No late submissions are accepted.

3 How to submit your work

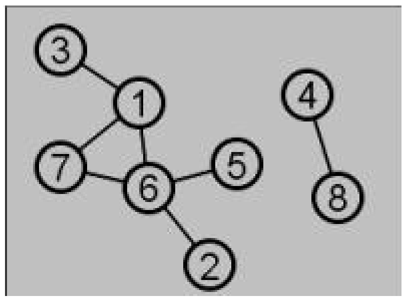
1. Your individual svn repository now contains a new directory called E2. It contains the starter code for this exercise. Checkout and study the starter code.
2. Once you are completely familiar with `GraphInterface`, complete the implemenation of the `GraphIsFullException, VertexExistsException, Graph` and`UseGraph` classes, so they obey the specifications below and the description in the starter code.
3. To submit your work, add and commit your changes to your repository.

   Do **not** commit the files and directories generated by Eclipse, such as `bin, doc, .project`, etc. Marks will be deducted if you submit these.

4 `Graph` ADT

In this exercise, you will write an implementation of the `GraphInterface` called `Graph`. A graph is an ordered pair (V, E) where V is a set of vertices, and E is the set of edges.

Each vertex will contain a unique piece of information (which can belong to any Java type, therefore our ADT is generic).



5 Specifications for `Graph.java`

- In our implementation, the set of edges is stored in an array of type T where T is a parameter.
- The set of edges, denoted E simply contains connections between two vertices.
- Our graph is undirected, so if our graph contains the edge $(1, 6)$, it also contains the edge $(6, 1)$.
- If an edge already exists between two vertices, no additional edges can be added between them.
- A vertex cannot have an edge connecting to itself.
- A connected component of a graph is a subgraph such that for any pair of vertices that belong to the set of vertices of that subgraph, there exist a path connecting these two vertices.
- A graph can contain more than one connected component.
- In our implementation, the set of edges if represented as a matrix of integers of dimension $n \times n$ (where n is the number of vertices).
- If there is an edge from vertex with index i to the vertex with index j, the element $E[i][j]$ contains a nonzero integer (we will set it to 1).
- The fact that the graph is undirected means if $E[i][j]$ is non zero, so is $E[j][i]$ (i.e the matrix E is symmetric).
- Example: the graph picured above, will be represented as follows:

$$V = [1, 2, 3, 4, 5, 6, 7, 8]$$

$$E = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- If there is no edge connectiong i and j, the respective matrix entry is 0.
- In our implementation (as indicated in the starter code as well) you must make use of our `Stack` ADT (with its linked implementation).
- Also you must use `Queue` and `Set` from Java Collections as indicated in the starter code.
- The method `DFSVisit` must use depth first search algorithm, which you can find in Wikipedia or other online resources. The code though must be entirely yours!
- In addition to the required methods, you may want to create helper methods at your will. If they are invoked from the required methods, then their functionality will be tested along with the required methods.

6 Specifications for `GraphIsFullException.java` and `VertexExistsException.java`

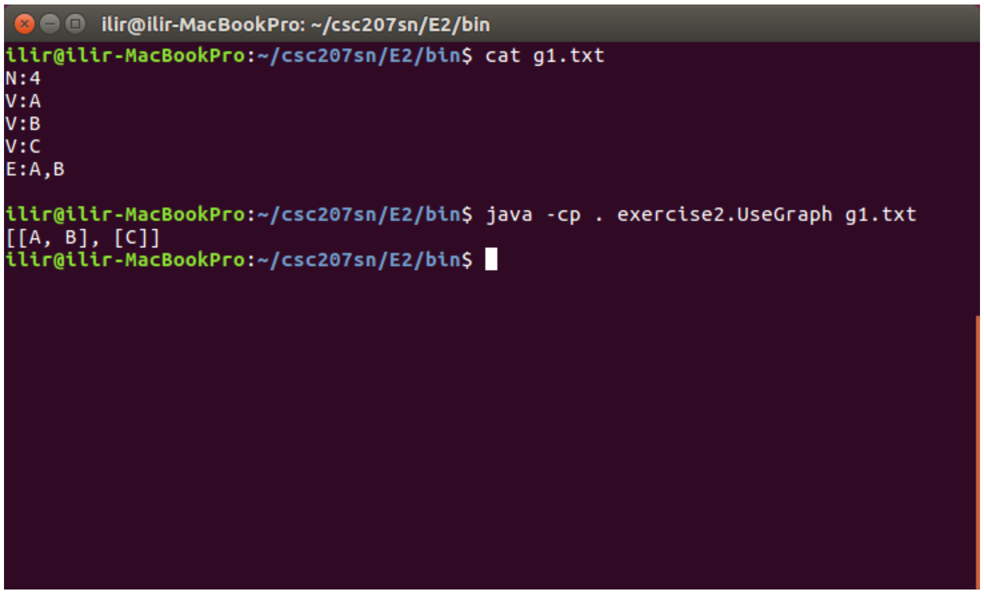
Two exception classes `GraphIsFullException` and `VertexExistsException.java` must be checked exceptions, and need only two members: a no-argument constructor, and a one-argument constructor that takes a `String` message. Each constructor should simply call the corresponding constructor of the parent class.

7 Specifications for `UseGraph`

- The `UseGraph` is contains the `main` method and it should take one command line parameter as follows:

  ```
  java -classpath .  exercise2.UseGraph file.txt
  ```

- The `file.txt` should be the name to a text file containing a graph in the format specified in the starter code (look at `loadGraph` method).
- The program should produce as output an ArrayList of connected components of the graph, represented as sets, as indicated in the screenshot below.



```
ilir@ilir-MacBookPro:~/csc207sn/E2/bin$ cat g1.txt
N:4
V:A
V:B
V:C
E:A,B

ilir@ilir-MacBookPro:~/csc207sn/E2/bin$ java -cp . exercise2.UseGraph g1.txt
[[A, B], [C]]
ilir@ilir-MacBookPro:~/csc207sn/E2/bin$
```

8 Checklist

Have you...

- tested your code on the lab computers using **Java 1.8**?
- committed the correct files in the correct directory?
- verified that your changes were committed using `svn list` and `svn status`?
- tested your solution, made any necessary changes, and re-committed if necessary?