**Documentation**

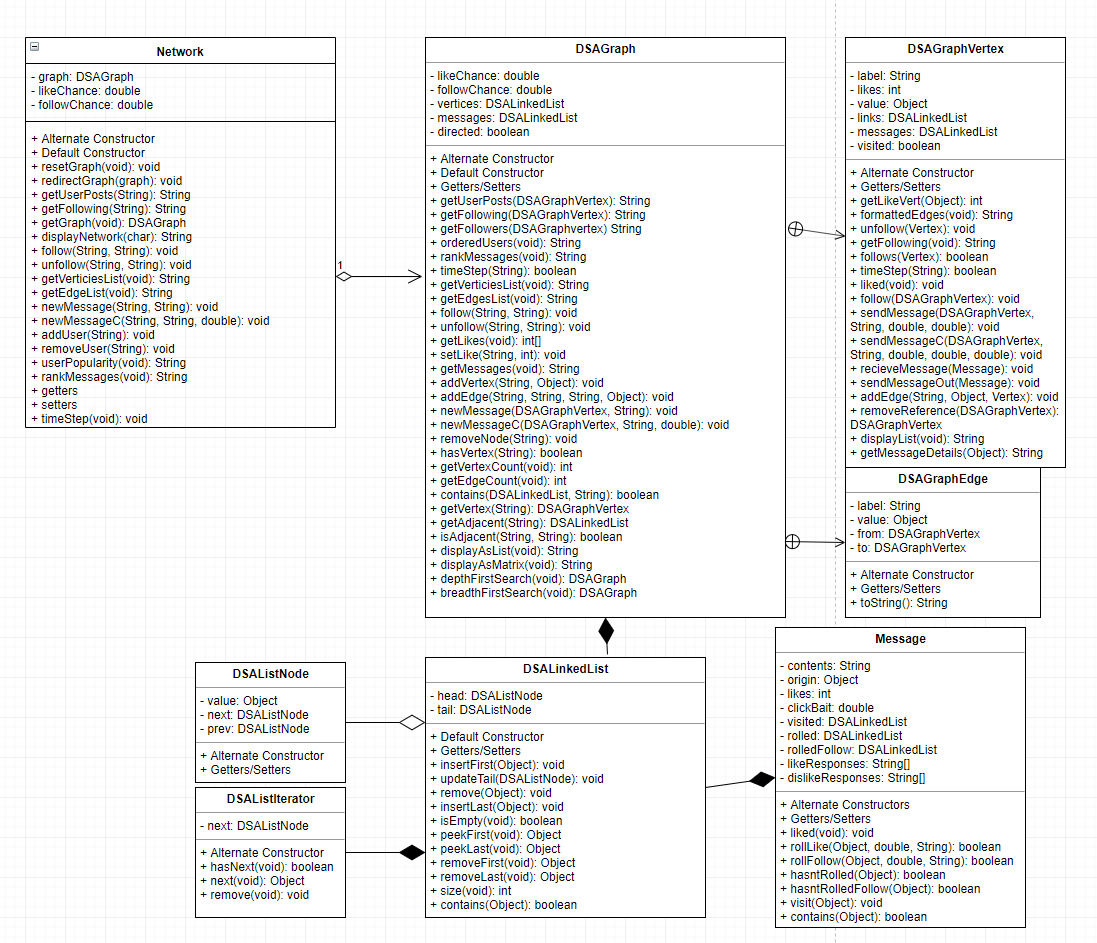
**DSA 2019 S2 Assignment – Social Network Simulation**

**Jonathan Wright (19779085)**

**Overview**

In my code I decided to make Network not inherit from graph as it didn’t seem as logical as making a Network have a graph, as a have-a relationship works with 0 – or more, where in this case a Network can indeed exist without a graph in my implementation.  
I decided to serialize as well as save/load as CSV to give the user a choice on how they would like to save their Network.  
I didn’t use many ADTs apart from LinkedLists, (Stack/Queues within methods) and graphs because the only ADT’s I thought could work better were Binary Search Trees and Hash Tables. I didn’t use BSTs as it would require a complete overhaul and rather than using a graph, I would be using a BST (Unsure if we could do this). I also did not use Hash Tables as I don’t have enough time for a refactor of my code, however, I admit that using a Hash Table instead of LinkedLists for storing messages would be much more optimized.  
I decided to rather than remaking the Network over and over in the main program I instead, ‘Redirect’ the graph of the Network to a new graph, this was to keep the probabilities correct as they are linked to the Network class rather than graph class.

**UML**



**Class Descriptions (Justifications in Purpose where relevant)**

**DSACircularQueue:**   
- Purpose: This was an Abstract Datatype implementation of a Queue, (Circular Implementation), it manipulates data in a FIFO format.  
- Why: We made this class in Practical 3 in order to explore a way of manipulating data.

**DSAGraph:**- Purpose: This is a modified Graph Abstract Datatype, it has been changed to have edges and some specific methods to make it work better with a network in mind.  
- Why: I have this Abstract Datatype in the final product and it was made in order to store a large amount of users in a logical format, showing connections and likes through edges and nodes.

**DSALinkedList:**- Purpose: This is a list abstract datatype that stores data in a Linked List implementation, it contains a implementation of a Doubly Linked Double Ended Linked List.  
- Why: I chose a Doubly Linked Double Ended Linked List as it is has better order than a singly linked list in some operations and I was not concerned with memory overhead in a Network.

**DSAQueue:**- Purpose: This is a abstract class for the abstract methods enqueue, dequeue and peek, this is implemented by another queue that contains the actual implementation for the ADT.  
- Why: I originally chose to have this as a abstract class rather than a superclass as in my mind it made sense to not have any code in it, however, I now believe it is far more logical to have to as a superclass.

**DSAStack:**  
- Purpose: This is a Stack abstract datatype, it manipulates data in a LIFO format.  
- Why: We made this class in Practical 3 in order to explore a way of manipulating data.

**FakeJUnit:**- Purpose: This is a class used very similar to Junit to make and test testHarnesses.  
- Why: I made this in Practical 5 after I was unable to figure out how to use JUnit without Dave’s zip file from ISE (I didn’t like running it with gradle it took to long on lab pc), this way I was still able to make test harnesses rather quickly with the same notation and have nice to look at pass/fail.

**FileIOGraph:**- Purpose: This will load graphs from csv files.  
- Why: I made this in Practical 5 to load graphs from files because it was required for the practical, it made sense to make this external to the graph class as it is very specific to IO.

**GraphTest:**- Purpose: Tests the Graph ADT.  
- Why: Need to test the graph ADT works as intended.

**Interactive:**  
- Purpose: Handles the Interactive Mode of Social Simulation.  
- Why: I chose to split up Interactive, Simulation and the startup into their own classes, it doesn’t really make sense from a ‘Object Oriented’ approach, however, it makes it more readable as Simulation and Interactive are very different in design.

**Message:**  
- Purpose: Stores message data from users in social network in a class.  
- Why: It makes a lot of sense from a ‘Object Oriented’ view to make all the message data into its own class that handles the storage and manipulation of this data.

**Network:**  
- Purpose: This is where the actual social network data is stored, it contains a graph ADT rather than inheriting from the graph as I preferred the ‘has-a’ approach to ‘is-a’ because a network can exist without a graph.  
- Why: Rather then using a graph for a very specific use case I can make a specific class for that specific use case to allow for easier use of a graph.

**NetworkIO:**  
- Purpose: Reads in CSV/Serialized Files to graphs and Saves to CSV/Serialized Files.  
- Why: IO is very specific and not always used by the other developers who might be using my Network class, as such I have it as its own separate class file.

**NetworkTestHarness:**- Purpose: This is a class that tests the network class.  
- Why: I need to test Network works as designed.

**SimulationMode:**  
- Purpose: Handles the Simulation Mode of Social Simulation.  
- Why: \*Refer To Interactive’s Why\*

**SocialSim:**  
- Purpose: Handles the start of the program, this is where it is all ‘linked’ into a program.  
- Why: The user needs a simple way to start the simulation.

**Sorts**  
- Purpose: Stores various sorting algorithms including a new modified Dutch Flags designed for messages.  
- Why: I use Dutch Flag sorting algorithm a lot as it is good with random data (Fastest out of all explored so far), as such I need Sorts.java

**testAppend**  
- Purpose: Tests my append method  
- Why: I needed to be sure my method was working.

**Utilities**  
- Purpose: This stores methods that various classes use.  
- Why: It avoids repeated code.