Assignment 1: Graphs

(replacement for P3a & P3b)

DUE: Apr. 5th at 11:59pm

Extra Credit Available for Early Submissions!

Setup

- Download the `assignment1.zip` and unzip it.
- This will create a folder `section-yourGMUUserName-a1`.
- Rename the folder replacing `section` with the `001`, `002`, `005`, etc. based on the lecture section you are in.
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address.
- After renaming, your folder should be named something like: `001-krusselc-a1`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`) .

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-a1.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:
      `001-krusselc-a1.zip --> 001-krusselc-a1 --> JavaFile1.java`
      `JavaFile2.java`
      `...`
- Submit to blackboard. DOWNLOAD AND VERIFY WHAT YOU HAVE UPLOADED THE RIGHT THING. ***Submitting the wrong files will result in a 0 on the assignment!***

Basic Procedures

You must:

- Have code that compiles with the command: `javac -cp .;../310libs.jar *.java` (Windows) or `javac -cp .:../310libs.jar *.java` (Linux/MacOS) in your user directory.
- Have code that runs with the command: `java -cp .;../310libs.jar SimGUI` (Windows) or `java -cp .:../310libs.jar SimGUI` (Linux/MacOS) in your user directory.
- Have a style (indentation, good variable names, etc.) -- you must pass the style checker!
- Comment your code well in JavaDoc style (no need to overdo it, just do it well) -- you must pass the comments checker!

You may:

- Add additional methods and variables to **`ThreeTenGraph`**, however these methods **must be private**.
- Add any additional classes you have written entirely yourself (such as simple data structures from previous projects). You may not add any imports not allowed on the previous projects to those files.
- Use any of the Java Collections Framework classes in `java.util` **already imported for you**. You may not add any imports that do not already exist.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables.
- Add any additional libraries/packages which require downloading from the internet.
- Use any code from the internet (including the JUNG libary) which was not provided to you in 310libs.jar.

- Import any additional libraries/packages or add any additional import statements (or use the "fully qualified name" to get around adding import statements). So you cannot use any built in Java Collections Framework classes that are not already imported for you.
- Alter any method/class signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided interfaces/classes that are complete (**`SimGUI`**, **`ThreeTenGraphComponent`**, **`ThreeTenEdge`**, **`ThreeTenNode`**).
- Add `@SuppressWarnings` to any methods unless they are private helper methods for use with a method we provided which already has an `@SuppressWarnings` on it.

Grading Rubric

No Credit

- Non submitted assignments
- Assignments late by more than 24 hours (*with or without tokens*)
- Non-compiling assignments
- Non-independent work
- Code that violates and restrictions or "you may not" mandates.
- "Hard coded" solutions
- Code that would win an obfuscated code competition with the rest of CS310 students

How will my assignment be graded?

- Automatic Testing (100%): To assess the correctness of programs.
- You CANNOT get points for code that doesn't compile or for submitting just the files given to you.
- Extra credit for early submissions:
  - 1% extra credit rewarded for every 24 hours your submission made before the due time
  - Up to 5% extra credit will be rewarded
  - Your latest submission before the due time will be used for grading and extra credit checking. You CANNOT choose which one counts.

Automated Testing Rubric

The JUnit tests used for grading will NOT be provided for you (you need to test your own programs!), but the tests will be based on what has been specified in the project description and the comments in the code templates. A breakdown of the point allocations is given below:

100 pts	ThreeTenGraph
-5pts ("off the top")	**Not following the submission format** Note: This is very, *very* important for these assignments; the graders need to return grades *very* fast. If you do not follow the submission format (the same one you've been using for P0, P1, and P2), then they will manually deduct 5pts from your score. No exceptions.
-5 pts ("off the top")	Big Graph (see below*)
-5 pts ("off the top")	Not passing code style check
-5 pts ("off the top")	Not passing JavaDoc style check

"Off the top" points (i.e. items that will lose you points rather than earn you points).

* Big Graph: We will create a 100 node graph where all nodes have a 100% chance of being connected. If your code takes more than one minute to create such a graph and display it, you will lose points. We promise to run it on a reasonably modern laptop (not a tablet, for instance).

Assignment Overview

Professional code often uses existing libraries to quickly prototype interesting programs. You are going to use 2-3 established libraries to quickly develop the internal representation of an advanced data structures (a graph) and simulate a simple interactive graph editing program.

The three libraries you are going to use are:

1. Your previous project code - You have written a number of simple data structures in this class and you are welcome to continue using these as part of this project.
2. A subset of the Java Collections Framework - This is a collection of existing simple data structures (lists, queues, etc.) which can form the basis for more advanced data structures.
3. A subset of JUNG (Java Universal Network/Graph Framework) - This library provides a lot of cool visualization tools for graphs: automatic layouts for graphs, an easy interface for creating/editing graphs, and much more.

Tasks Breakdown and Sample Schedule

There are **3** tasks in this assignment. It is suggested that you implement these tasks in the given order:

- Task 1: Examine the JCF Classes (0%)
- Task 2: Read the Provided Code Base (0%)
- Task 3: Implement a Directed Graph Class to Support the Simulator (100%)

See the Examples for Testing section of this document for what the simulator should be able to do when you are done, and then see the Task Details section for a walk-through of each specific task.
Need a schedule?

- You've got 1.5 weeks.
- The graph class has 20 methods.
- You have other classes with exams/projects.
- Assume you want to spend the last haft week getting EC or seeking additional help.
- Keeping those things in mind, fill in the following:

    - 3/23-3/26: _____ (first week period)
        - Suggestions: Task 1 and Task 2, start JavaDocs for Task 3
    - 3/27-3/29: _____ (first weekend period)
        - Suggestions: Finish JavaDocs for Task 3, Task 3
    - 3/30-4/02: _____ (second week period)
        - Suggestions: Thorough Testing and Debugging
    - 4/03-4/05: _____ (second weekend period)
        - Suggestions: Turn in early for Extra Credit

Task Details

There are **3** tasks in this assignment. It is suggested that you implement these tasks in the given order:

- Task 1: Examine the JCF Classes (0%)
- Task 2: Read the Provided Code Base (0%)
- Task 3: Implement a Directed Graph Class to Support the Simulator (100%)

Task 1: Examine the JCF Classes (0%)

Read and familiarize yourself with the following JCF classes. Some of these are allowed (or required) in certain parts of the code. Below is an overview:

1. ArrayList and LinkedList - Java's list classes supported by a dynamic array or a linked structure respectively

2. Collection - All JCF classes implement this generic interface.

Where should you start? The Java Tutorials of course! (If you didn't know, Oracle's official Java documentation includes a set of tutorials.) The Trail: Collections tutorial will provide you with more than enough information on how to use these classes.

Task 2: Read the Provided Code Base (0%)

Read and familiarize yourself with the code. This will save you a lot of time later. An overview of the provided code in is given below, but you need to read the code base yourself.

```
//Parent class of all graph components. This class is provided
//in full.
class ThreeTenGraphComponent {...}

//This class represents a directed graph formed by connecting
//nodes (of any type of ThreeTenGraphComponent) with edges (of
//and type of ThreeTenGraphComponent).
//You will write 99% of this class, but a template is provided.
class ThreeTenGraph {...}

//Represents a node in a graph used for the GUI. This class is
//provided in full. Note that this is not the only type of node
//possible for a ThreeTenGraph (it's just the one for the GUI).
class ThreeTenNode {...}

//Represents an edge in a graph used for the GUI. This class is
//provided in full. Note that this is not the only type of edge
//possible for a ThreeTenGraph (it's just the one for the GUI).
class ThreeTenEdge {...}

//This is the simulator and handles all the graphical stuff,
//it is provided in full but may be of interest to some people.
class SimGUI {...}
```

You are required to complete the JavaDocs and adhere to the style checker as you have been for all previous projects. The **`checkstyle.jar`** and associated **`.xml`** files are the same as on previous projects. You may need to correct/edit the provided style and/or JavaDocs. The JUNG library comments don't quite adhere to the style requirements for this class, so you must fix them (this is a normal process when integrating code from other libraries -- boring but necessary practice for the "real world"). It is HIGHLY RECOMMENDED that you write your JavaDocs for this project *first* and during this stage. That way you will have a full understanding of the code base as you work.

Task 3: Implement a Directed Graph Class to Support the Simulator (100%)

In order for the simulator to work, you need an internal representation of a graph. The JUNG library provides an interfaces for this: `Graph<V,E>`. You need to implement the directed graph `ThreeTenGraph<V extends ThreeTenGraphComponent, E extends ThreeTenGraphComponent>` (in `ThreeTenGraph.java`) which implements the `Graph<V,E>` and `DirectedGraph<V,E>` interfaces. The `ThreeTenGraph` class needs some internal storage and you have a lot of freedom in how you'd like to implement this within the class, so make sure to read the instructions there.

Below is a quick overview of the methods you need to support. Note that in the template, actual JavaDoc comments are provided. That said, the JavaDocs are from the `Graph<>` interface and the `HyperGraph<>` interface in JUNG. They have been copied from that library for your reference, but are otherwise unaltered. Part of this assignment is to practice reading "real" documentation and understanding what to implement based on the library's requirements.

```
//*******************************
// Graph Editing (30%)
//*******************************
boolean addEdge(E e, V v1, V v2) {...}
boolean addVertex(V vertex) {...}

boolean removeEdge(E edge) {...}
boolean removeVertex(V vertex) {...}

//*******************************
// Graph Information (60%)
//*******************************

//For a given graph...
Collection<E> getEdges() {...}
Collection<V> getVertices()  {...}

int getEdgeCount() {...}
int getVertexCount() {...}

//For a given vertex in a graph...
Collection<E> getInEdges(V vertex) {...}
Collection<E> getOutEdges(V vertex) {...}
Collection<E> getIncidentEdges(V vertex)

Collection<V> getPredecessors(V vertex) {...}
Collection<V> getSuccessors(V vertex) {...}
Collection<V> getNeighbors(V vertex)

//Given two verticies in a graph...
E findEdge(V v1, V v2) {...}

//Given an edge in a graph...
V getSource(E directed_edge) {...}
V getDest(E directed_edge) {...}
Pair<V> getEndpoints(E edge)

Collection<V> getIncidentVertices(E edge)
```

When you are done with this step, you can generate and play with some graphs in the simulator (see the Examples for Testing section).

Hints and Notes

- Read ALL the methods before you decide how to implement any methods, you may need track a lot more things than the simple graphs we covered in class.
- Note that we cannot test editing a graph or getting information about a graph independently of each other. So you cannot get points for completing only the graph editing or only the graph information parts of this interface, you need everything...

Examples for Testing

ThreeTenGraph has a main method for testing with a couple of sample "yays". You should thoroughly test this class before trying to run the simulator. If you see errors/weird things in the simulator it means something is wrong in YOUR code. All the simulator does is call your methods! For example:

Issue:

You delete a node, but there are still edges appearing "in the air" on the simulator which should have been removed.

Possible Causes:

Your removeVertex() method might not be working properly.
Your getEdges() or getEdgeCount() methods might not be working properly.

Definitely NOT the Cause:

The simulator is broken.

How do you diagnose the problems?

Debugger, breakpoints, print statements...

Displaying a Graph

Once you have the Graph class working, you can run the main program to test and debug. The main program is `SimGUI`, which can be compiled and run with the following commands if you are on Windows:

```
javac -cp .;310libs.jar *.java
java -cp .;310libs.jar SimGUI
```

or the following commands if you are on Linux/MacOS:

```
javac -cp .:310libs.jar *.java
java -cp .:310libs.jar SimGUI
```

Why is there extra stuff? The `-cp` is short for `-classpath` (meaning "where the class files can be found). The `.;310libs.jar` or `.:310libs.jar` has the following components: `.` the current directory, `;` or `:` the separator for Windows or Linux/MacOS respectively, `310libs.jar` the provided jar file which contains the library code for JUNG.

If you run the simulator with the above command, you will get a six node graph with some random edges. Each time you hit "reset" you get another graph, but the same sequence of graphs is always generated (for your testing). However, the simulator can also be run with some additional optional parameters to get some more interesting results: The number of nodes, the likelihood that two nodes have an edge between them, the random seed for the graph generator. The next few tables give examples of what you can do.

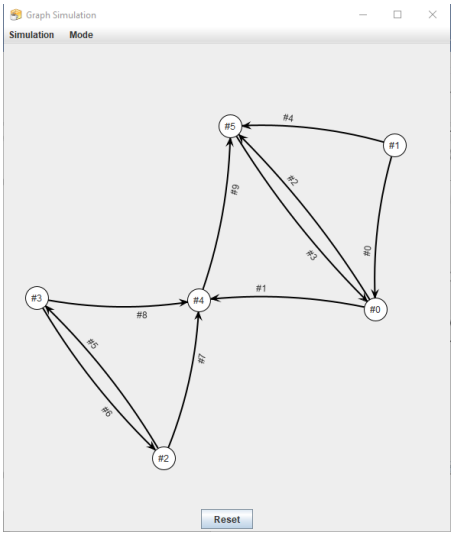
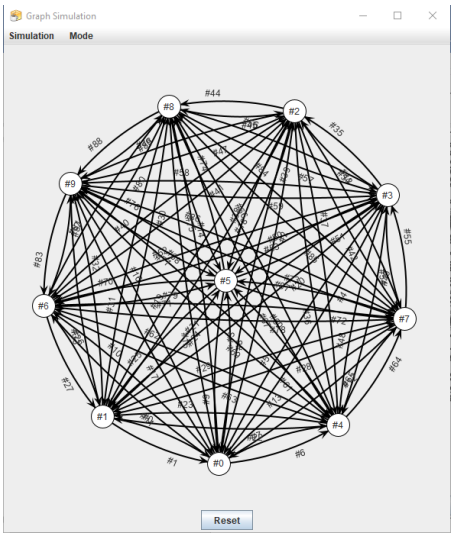
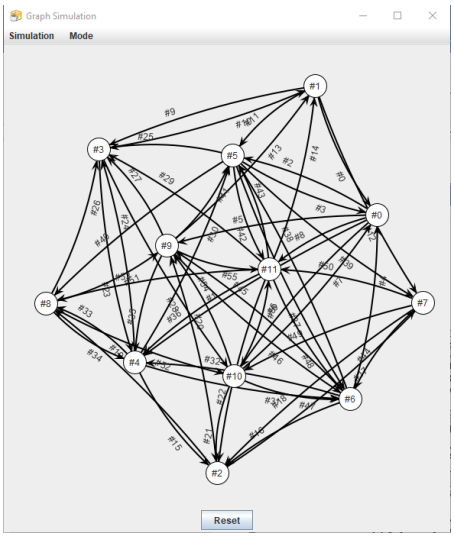
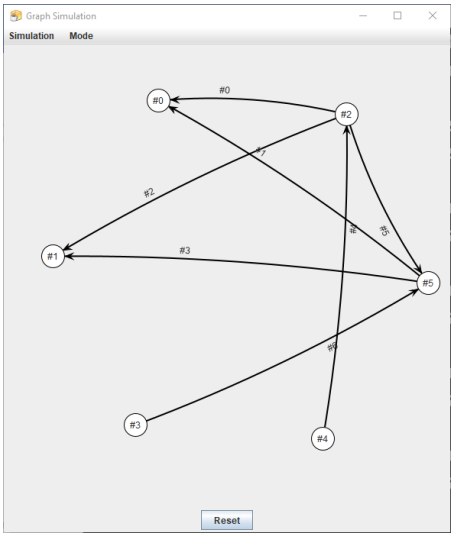
Image	Command + Explanation
	`java -cp .;../310libs.jar SimGUI` Generate a six node graph, with connection probability of 0.4, and seed 0.
	`java -cp .;../310libs.jar SimGUI 10 1` Generate a ten node graph where all nodes are connected.
	`java -cp .;../310libs.jar SimGUI 12 0.5` Generate a twelve node graph where nodes have a 50% chance of being connected.

Image	Command + Explanation
	`java -cp .;../310libs.jar SimGUI 6 0.4 1123` Generate a different sequence of graphs using seed 1123.

Adding nodes and edges to a graph

You'll want to test out adding multiple nodes and edges from your graphs to make sure you've gotten out all the bugs.

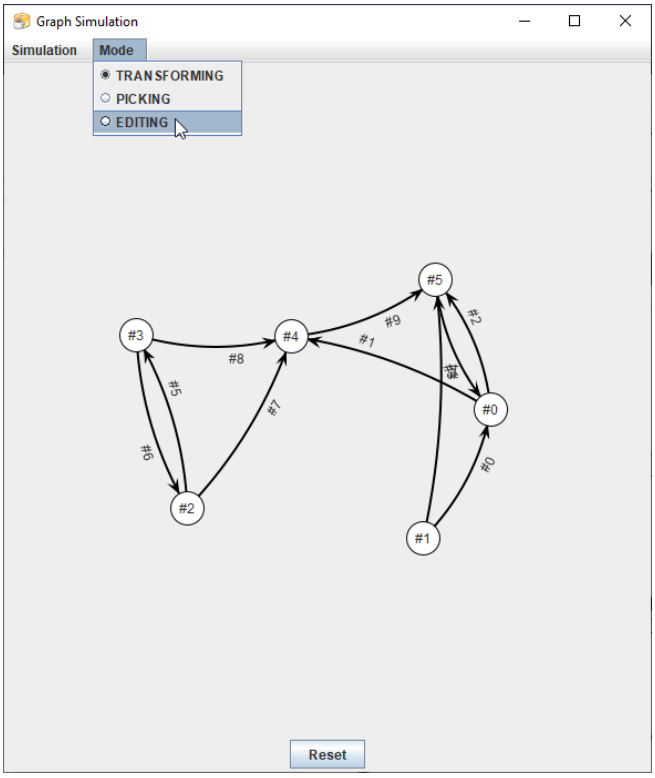
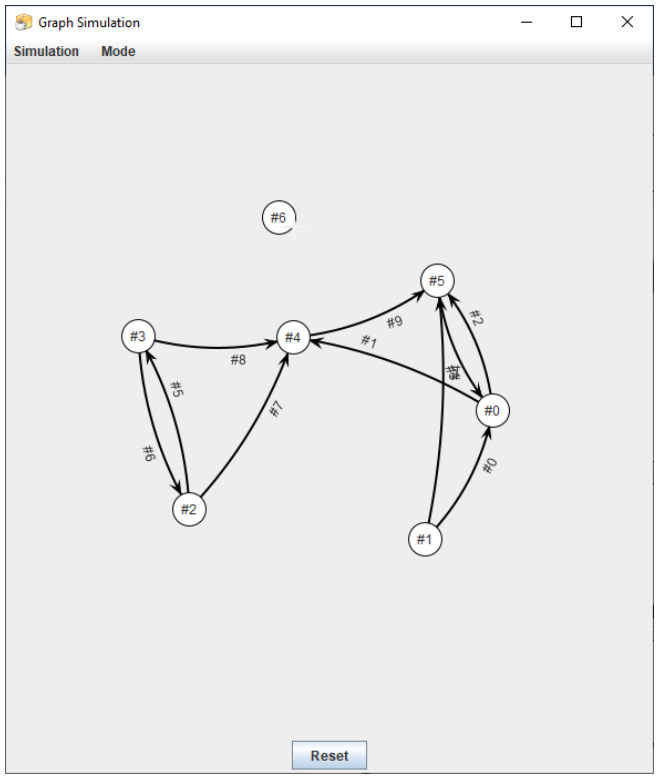
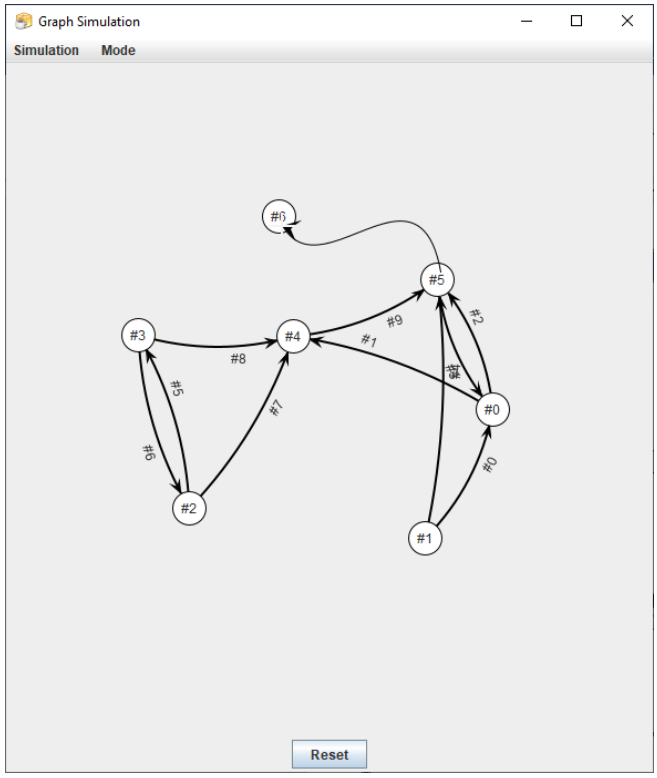
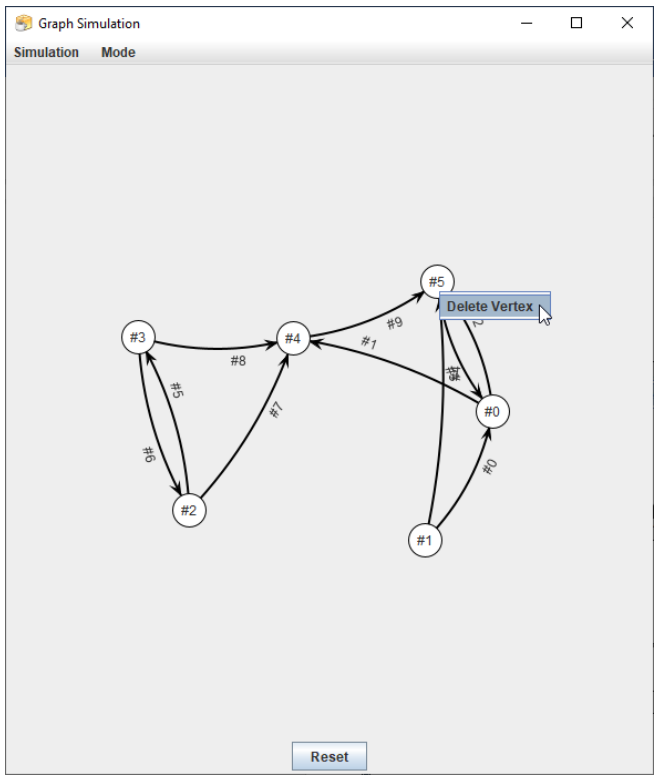
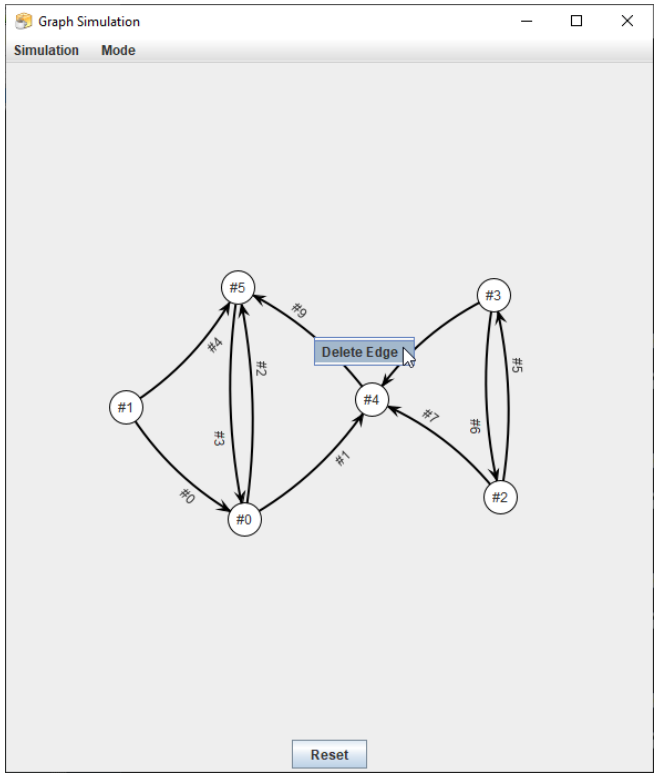
Image	Explanation
	Select "Mode", then "Editing".

Image	Explanation
	Click Anywhere on the graph surface to add a node.
	Drag to another node to add an edge with a random weight.

Removing a nodes and edges from a graph

You'll want to test out removing multiple nodes and edges from your graphs to make sure you've gotten out all the bugs.

Image	Explanation
	In any mode, right click a node and select "delete vertex".
	In any mode, right click an edge and select "delete edge".