COMP261 – Assignment One Report

Features and Functionality:

- The program loads and parses the given data set of files into a set of data structures which is achieved through the utilization of the *BufferedReader* and *FileReader* classes. It uses appropriate Java Utility parsing methods such as Integer.parseInt(str), Double.parseDouble(str) to accurately parse the data and store them into their respective class fields.
- There are three primary classes in which the data is loaded into – Node, RoadSegments & Road. Each class has its own set of fields which are based on the fields that are given in the data set. I.e. ID, Location, Coordinates etc. In addition, there is also a Polygon class which also loads and parses the data from the Polygon data set. All data is loaded when the onLoad() method is called.
- Once all data has been loaded into their respective data structures then a new Trie data structure is initialized by populating itself with all the road names from the Road data set. It essentially becomes a mini directory for all the road names across Auckland which is queried via the onSearch() method. Fundamentally, the Trie accurately highlights all the roads that are based on the prefix of the input string.
- In addition, a QuadTree data structure is also initialized where it first establishes a BoundingBox object acting as a boundary for all quadNodes. The QuadTree is then populated by iterating through the entire set of Nodes/Intersections and extracts its location to determine the smallest quad that this Intersection belongs to. The QuadTree is utilized for extracting the Intersection that is closest to the mouse click of the user and a query function is called inside the onClick() method.
- Once all forms of data loading has been processed then the redraw() method is called where it simply iterates through all the Intersections, RoadSegments and Polygons and draws them onto the drawing pane of the GUI.
- The onMove() method responds based on the MouseEvent instance entered by the user, three primary fields are used here – offsetX, offsetY & scale. These three fields allow the redraw() method to accurately redraw and shift/adjust the graph based on the pan and zoom input commands of the user.
- There are two additional methods implemented inside the GUI class which are the onScroll() and onMouseMove() these allow the user to simply use the scroll wheel to zoom in and out of the graph and pan around the map using the mouse.

Data Structures:

- **HashMap<Integer, Obj>:** A HashMap was utilized to handle the storage of the Node and Road object where the key represents the ID and the Obj is simply a Node or a Road object. A Map is an efficient data structure for these data types as they both contain unique identifiers meaning that an object can be directly referenced using the Map.get(key) method which has a cost of O(1), it saves us time from performing a linear search across the map.
- **List<RoadSegments>:** An arrayList is used as it is a dynamic data structure that can adapt to the size of the given data set. It also contains useful utility methods such as .add() .clear() .get() and .contains() which are all critical when loading, resetting and making object

comparisons. An argument could be made that an Array could be more efficient as it does not utilise as much memory as an ArrayList however we would have to manually record the number of entries/lines in the data files and use these as the length of these arrays, these lengths would be dependent on which data set is used - large or small.

- **Trie:** A Trie data structure is utilised to create a directory of road names which can be queried by the user to find a particular road. It functions by taking each character of the prefix and treats them as TrieNodes which consists of a particular character along with a set of TrieNode children. Each character that is read from the prefix is then processed by the Trie to see if there are any valid matches between the prefix and some entry in the Trie. If a match or multiple matches occur then the Trie returns the name(s) of the road(s) and highlights them on the GUI.
- **QuadTree:** A QuadTree is utilised to quickly determine the closest possible Intersection with respect to the mouse click of the user. It functions by initially populating it's Tree with all the locations of all the Intersections on the graph and finds the smallest possible quadrant that each Intersection belongs to. It finds the smallest quadrant by continuously splitting it's Nodes into four children and as each split occurs the boundaries of each quadrant (width x height) are halved until it reaches the point where there is only a maximum of 4 data points for each quad. If a Node is determined as a "leaf" then it must contain data but not have children, if it is not a leaf then it will have children but no data. The tree is queried by starting from the root and it traverses deeper and deeper into the tree until the mouse click location is within the boundaries of the quadNode that matches the corresponding Intersection location.

Testing:

- The program was tested by using a mixture of Eclipse debugger tools and print statements.
- Both data sets were used – Large and Small, both successfully loaded.
- To check whether each data structure had the correct data loaded, I would check the size() of each structure and check to see whether it corresponds to the total number of lines in the data set. I.e. for the small data set of Nodes it contained 1080 lines hence I would check if the Map containing all the Nodes also had a size of 1080, which it did.
- Testing the creation of a new graph object involved placing breakpoints right before the creation statements and check to see whether the parameters passed onto its constructor are accurate. I.e. I would halt the program and check to see if the Node about to be created contains the correct ID, lat & lon variables before creating the actual object.
- Testing the process of drawing the graph involved checking that each pixel location was converted accurately from the physical location of the Intersections and RoadSegments. This involved checking that the Origin location is within the appropriate bounds, the scale suits the size of the data set and that the panning and zooming components respond according to the input commands of the user. I.e. when a user hits the right button then the offsetX has 10units added to it which causes a shift in the pixel location of all Node objects 10 units to the right.
- Testing the onSearch() method with the Trie data structure involved a combination of the onClick() method and the use of print statements. The onClick() would be used to determine the names of the roads at an intersection to give me an idea of a potential road that could be searched. The print statements were then used to check whether each character of the prefix had a valid entry/match in the Trie, it would return a valid entry if a match did occur and it would print the corresponding road onto the text box. In addition to printing the output it would also highlight the corresponding road on the GUI. There were instances

when it only highlighted a section of the road, this flaw was rectified by realizing the fact that in addition to a road consisting of a sequence of segments, a road can also contain multiple roads formed to create one road object.

- The mouseScrolling and mousePanning were simply tested by adapting a similar way that the onMove() method was tested where if a mouseWheel was scrolled then the GUI would simply zoom in or out of the graph and if the mouse was moved then it would simply pan around the graph.