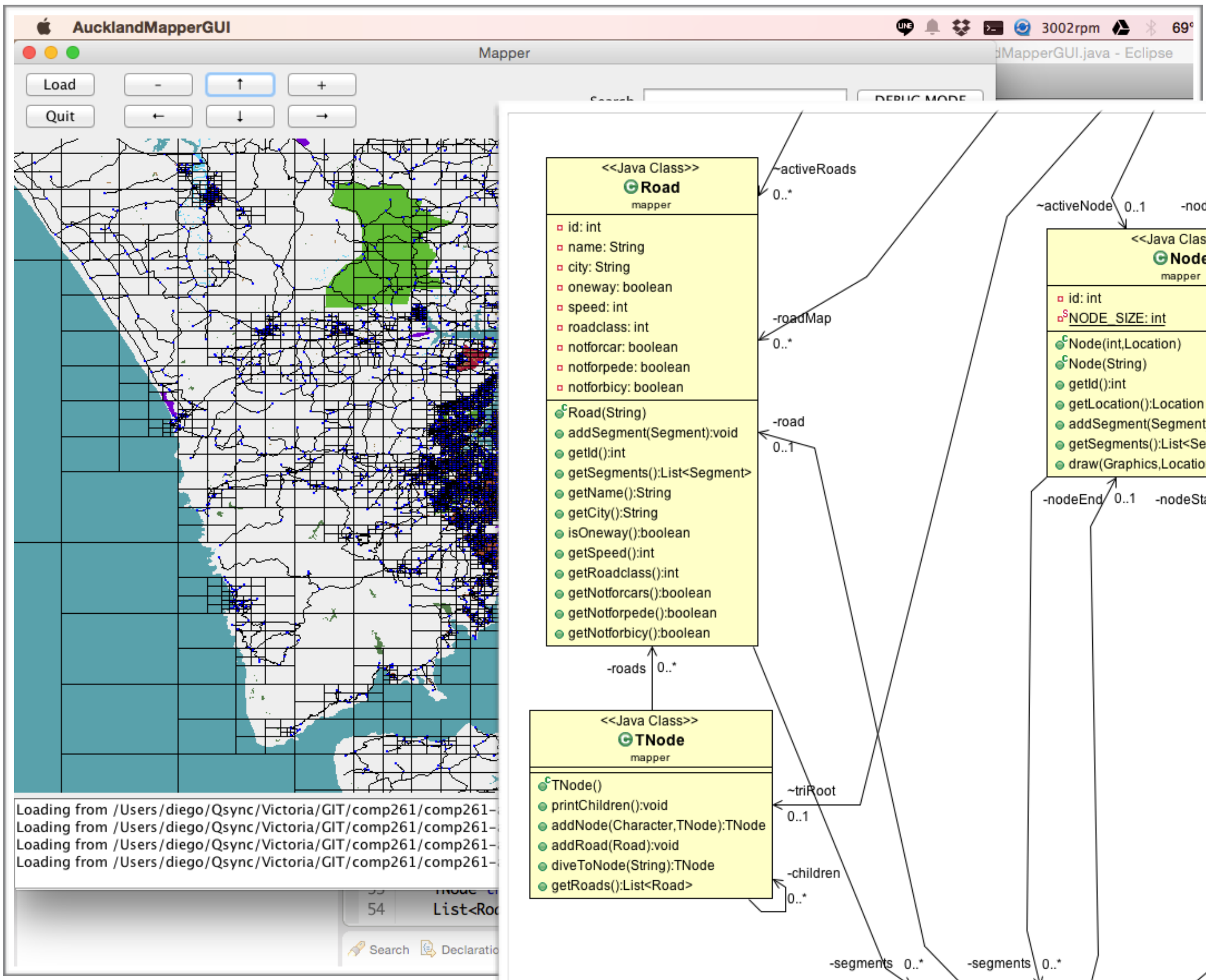
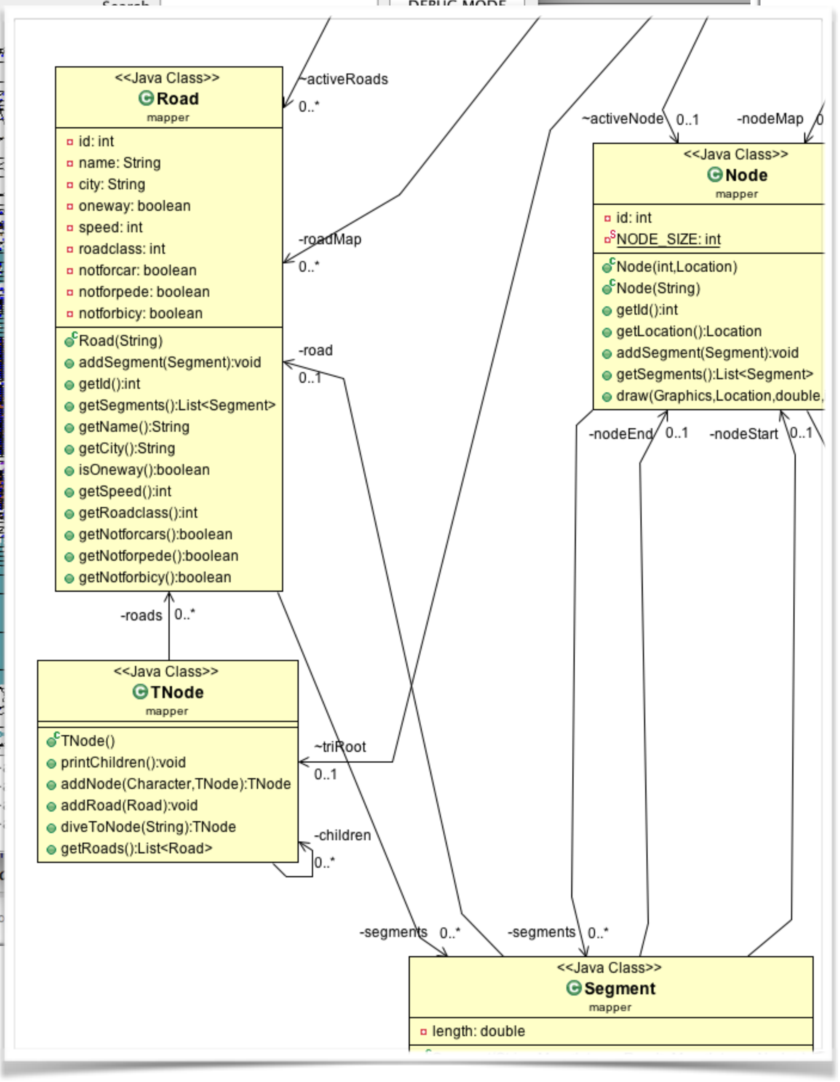


```
Loading from /Users/diego/Qsync/Victoria/GIT/comp261-
Loading from /Users/diego/Qsync/Victoria/GIT/comp261-
Loading from /Users/diego/Qsync/Victoria/GIT/comp261-
Loading from /Users/diego/Qsync/Victoria/GIT/comp261-
```



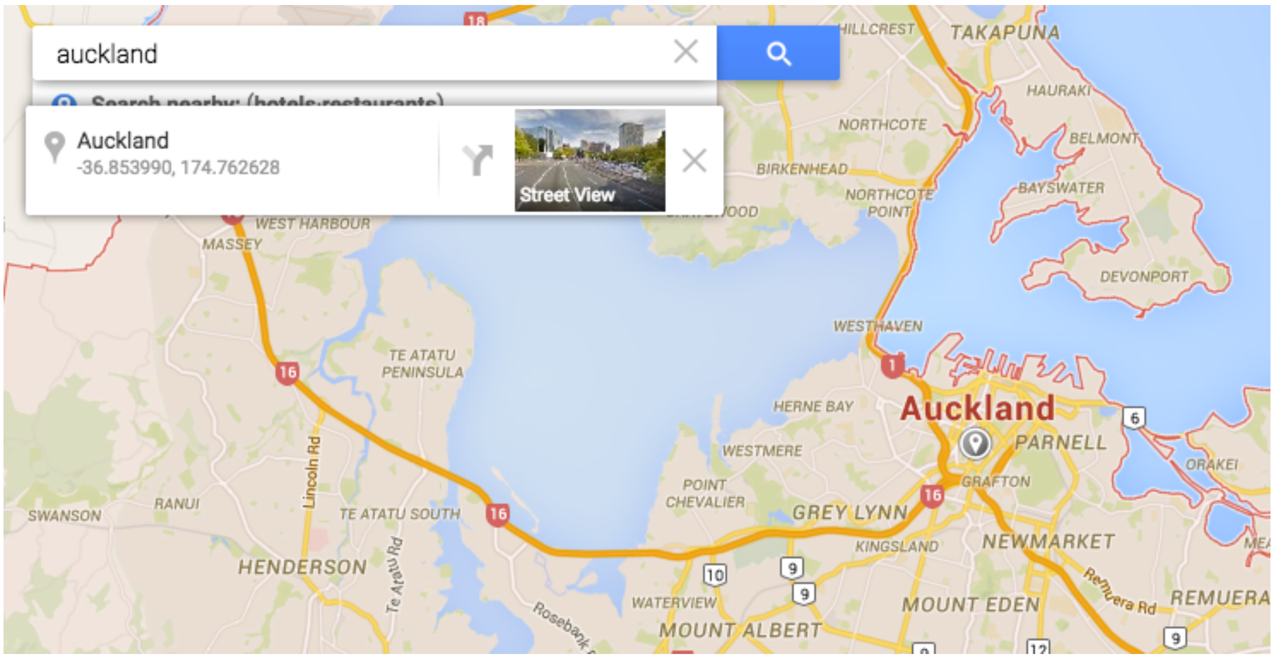
Assignment 01
Auckland Map

Prepared by: Diego Trazzi
5 March 2015
ID number: 300338937

DESIGN PROCESS

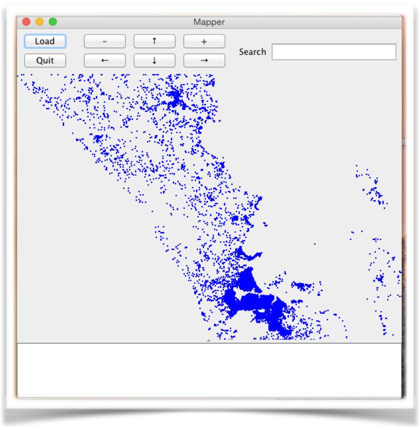
Initial prototyping - v0.1

In order to have a first prototype of the software working and displaying data, I used Google Maps to determine the GPS coordinate of Auckland so I could use them to manually set the centre of the map. The latitude and longitude of Auckland Centre is about -36.851724, 174.762628. Furthermore, latitude is the Y correspondent on the cartesian axis (north/south) whereas the longitude is the X axis.



I also changed from original code: DEFAULT_DRAWING_WIDTH from private to protected, so I could access it in the child class to determine an accurate origin:

```
protected void setOrigin() {
        double[] bounds = this.setBounds();
        origin = new Location(bounds[0], bounds[2]);
        scale = Math.min(GUI.DEFAULT_DRAWING_WIDTH / (bounds[1] - bounds[0]),
                     GUI.DEFAULT_DRAWING_HEIGHT / (bounds[2] - bounds[3]));
}
```

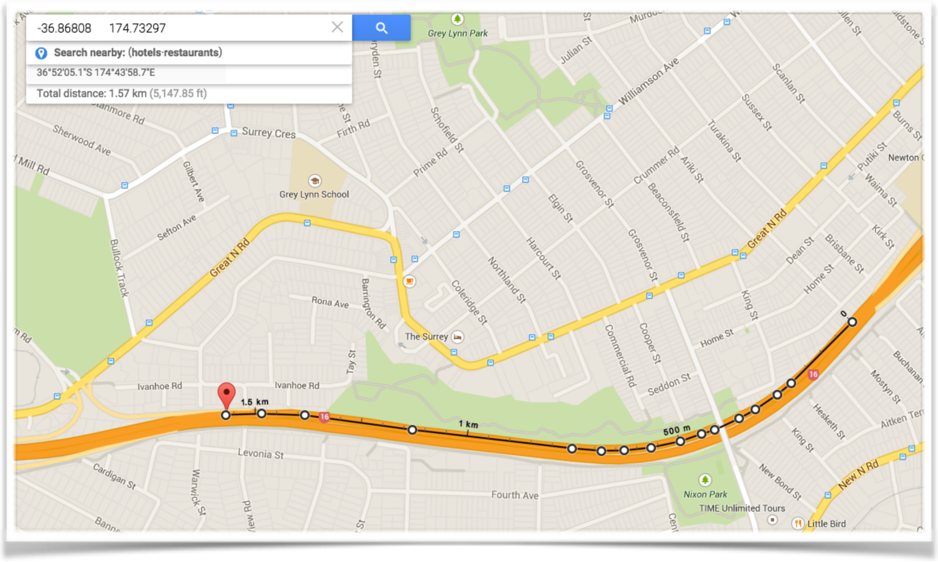


As displayed in the picture aside, this first version of the Mapper (v0.1) could only display the node locations.
During this phase I spent some time learning and understanding the way the data was formatted in the text to be able to create the appropriate classes.

ADDING SEGMENTS

To verify the position and data structure of the segment I used Google maps to manually plot some of the coordinate from the segment file. For example, entering the following data in GMaps would draw the map below :

```
13107       1.5686264261966472   15507       13539       -36.86613   174.74946   -36.8669    174.74853   -36.86741   174.74786   -36.86765
174.74748   -36.86797   174.7469    -36.86816   174.74647   -36.8684    174.74583   -36.86848   174.74548   -36.86864   174.74493   -36.86877
174.74415   -36.86883   174.74345   -36.86883   174.74284   -36.86878   174.74209   -36.8684    174.73788   -36.86812   174.73501   -36.86806
174.7339    -36.86808   174.73297
```


-36.86808 174.73297
Search nearby: (hotels·restaurants)
36°52'05.1"S 174°43'58.7"E
Total distance: 1.57 km (5,147.85 ft)

Loading and adding segments was a bit more complicated than loading roads or nodes. A segment (portion of rad), belongs both to intersections (nodes) and to streets (roads), so instead of opting for a third HashMap containing the segments, I decided it would be better to include this data in the Road Map and Node Map because, one of the requirement is to highlight the portion of segments surrounding the intersections. The drawback of this design is that the memory requirement to store the ListArray of segments will be twice as large as if it was only contained in one list.

PANNING AND ZOOMING

To improve the GUI I adjusted the zoom/shift to have an acceleration curve and to adapt to the current scale factor. In this way the user experience can be more pleasant.

For the mouse navigation I decided to utilise the mouse dragging listener combined with middle mouse button. I created new listeners in the Gui class and implemented them in the AucklandMapperGUI. One nice trick used here was that instead of converting all the locations of the nodes onto screen dimensions, I simply converted the mouse position into GPS coordinates for the comparison to the closest point. By doing so I avoided a large amount of conversions, and hopefully speeded up the interface a touch.

DRAWING POLYGONS

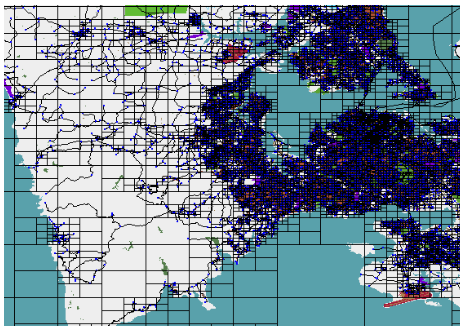
To challenge myself I decided to add polygon shapes and mouse navigation to allow a nicer user experience. The challenge of drawing polygons started with a different format for the records of the polygons. The polygons file not only contained the array with all the points for drawing the polygons, but also lot of information which such as city name, labels and type. The type was particularly important, as after some research online, I realised the numbers for the type were hex codes which represented the drawing order (http://www.cgpsmapper.com/download/custom.txt). After splitting all the text entry into appropriate fields into a new polygon class, I used a tree map with polygon type as key and polygons for values. In this way I could utilise the treeMap sorting properties to draw all the polygons in appropriate order.

OTHER GUI IMPROVEMENTS

Finally, to speed up the drawing method I redesigned the code to only draw those nodes which are within the bounds of the display area. By doing so the draw method doesn't need to process the drawing al all the nodes in the file.

QUAD TREES

As part of the challenge, although took me an entire day, I have managed to successfully implement a quad-tree structure. I was very pleased and excited about this feature, and I have added a drawing method in the UI to display the quad structure (DEBUG button). Using quads, instead of traversing the entire list, optimises the search of neighbours when the user click on a node.

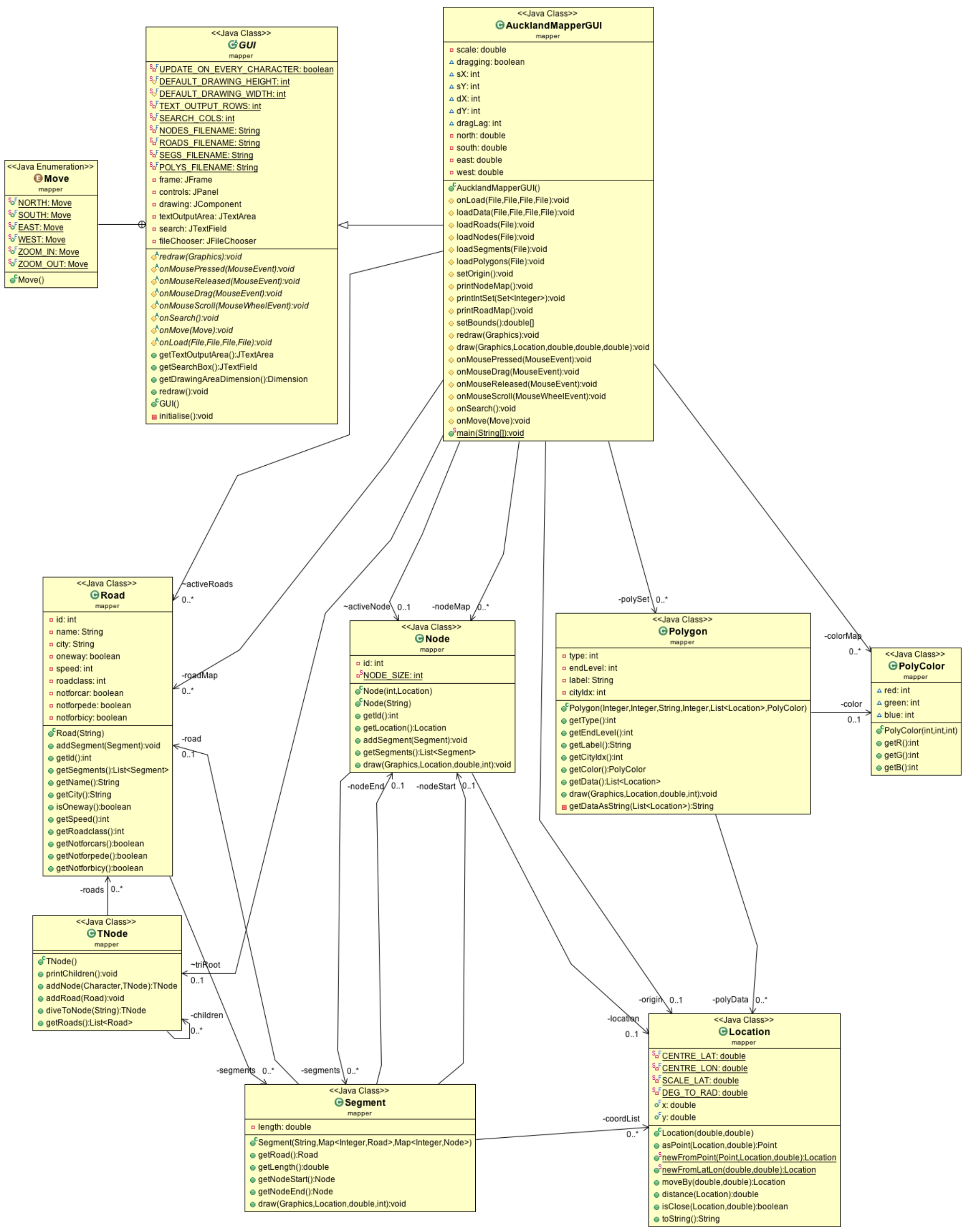


TRIE

As part of the assignment I have also implemented a TRIE structure for quick searches on the text box. The Trie will return any street name which starts with the give prefix. The TRIE structure is, again, based on a tree structure and speeds up the process of searching for a street name by organising the data in an ordered way.

UML

Finally, although, not required by the assignment handout I have prepare a UML representation of of my Assignment. I have done this though out the working stages for sanity check and for better organisation. Here is the final representation:



<<Java Enumeration>> Move mapper
NORTH: Move
SOUTH: Move
EAST: Move
WEST: Move
ZOOM_IN: Move
ZOOM_OUT: Move
Move()
<<Java Class>> GUI mapper
UPDATE_ON_EVERY_CHARACTER: boolean
DEFAULT_DRAWING_HEIGHT: int
DEFAULT_DRAWING_WIDTH: int
TEXT_OUTPUT_ROWS: int
SEARCH_COLS: int
NODES_FILENAME: String
ROADS_FILENAME: String
SEGS_FILENAME: String
POLYS_FILENAME: String
frame: JFrame
controls: JPanel
drawing: JComponent
textOutputArea: JTextArea
search: JTextField
fileChooser: JFileChooser
redraw(Graphics):void
onMousePressed(MouseEvent):void
onMouseReleased(MouseEvent):void
onMouseDrag(MouseEvent):void
onMouseScroll(MouseWheelEvent):void
onSearch():void
onMove(Move):void
onLoad(File,File,File,File):void
getTextOutputArea():JTextArea
getSearchBox():JTextField
getDrawingAreaDimension():Dimension
redraw():void
GUI()
initialise():void
<<Java Class>> AucklandMapperGUI mapper
scale: double
dragging: boolean
sX: int
sY: int
dX: int
dY: int
dragLag: int
north: double
south: double
east: double
west: double
AucklandMapperGUI()
onLoad(File,File,File,File):void
loadData(File,File,File,File):void
loadRoads(File):void
loadNodes(File):void
loadSegments(File):void
loadPolygons(File):void
setOrigin():void
printNodeMap():void
printIntSet(Set<Integer>):void
printRoadMap():void
setBounds():double[]
redraw(Graphics):void
draw(Graphics,Location,double,double,double):void
onMousePressed(MouseEvent):void
onMouseDrag(MouseEvent):void
onMouseReleased(MouseEvent):void
onMouseScroll(MouseWheelEvent):void
onSearch():void
onMove(Move):void
main(String[]):void
~activeRoads 0..*
-roadMap 0..*
-road 0..1
~activeNode 0..1
-nodeMap 0..*
-polySet 0..*
-colorMap 0..*
-color 0..1
<<Java Class>> Road mapper
id: int
name: String
city: String
oneway: boolean
speed: int
roadclass: int
notforcar: boolean
notforpede: boolean
notforbicy: boolean
Road(String)
addSegment(Segment):void
getId():int
getSegments():List<Segment>
getName():String
getCity():String
isOneway():boolean
getSpeed():int
getRoadclass():int
getNotforcars():boolean
getNotforpede():boolean
getNotforbicy():boolean
<<Java Class>> Node mapper
id: int
NODE_SIZE: int
Node(int,Location)
Node(String)
getId():int
getLocation():Location
addSegment(Segment):void
getSegments():List<Segment>
draw(Graphics,Location,double,int):void
<<Java Class>> Polygon mapper
type: int
endLevel: int
label: String
cityIdx: int
Polygon(Integer,Integer,String,Integer,List<Location>,PolyColor)
getType():int
getEndLevel():int
getLabel():String
getCityIdx():int
getColor():PolyColor
getData():List<Location>
draw(Graphics,Location,double,int):void
getDataAsString(List<Location>):String
<<Java Class>> PolyColor mapper
red: int
green: int
blue: int
PolyColor(int,int,int)
getR():int
getG():int
getB():int
-roads 0..*
-nodeEnd 0..1
-nodeStart 0..1
<<Java Class>> TNode mapper
TNode()
printChildren():void
addNode(Character,TNode):TNode
addRoad(Road):void
diveToNode(String):TNode
getRoads():List<Road>
~triRoot 0..1
-children 0..*
-segments 0..*
-segments 0..*
-origin 0..1
-polyData 0..*
-location 0..1
-coordList 0..*
<<Java Class>> Segment mapper
length: double
Segment(String,Map<Integer,Road>,Map<Integer,Node>)
getRoad():Road
getLength():double
getNodeStart():Node
getNodeEnd():Node
draw(Graphics,Location,double,int):void
<<Java Class>> Location mapper
CENTRE_LAT: double
CENTRE_LON: double
SCALE_LAT: double
DEG_TO_RAD: double
x: double
y: double
Location(double,double)
asPoint(Location,double):Point
newFromPoint(Point,Location,double):Location
newFromLatLon(double,double):Location
moveBy(double,double):Location
distance(Location):double
isClose(Location,double):boolean
toString():String