Graph.java

```java
//Name:     Mark Lambert, Darius Herdes
//Date:     10/12/2024
//Purpose:  Data Structures & Algorithms Final Assessment
//Class for Graph (AdjacencyMatrix) Implementation
package Graph;

public class Graph {
    //Helper Variables
    private final int SIZE = 6;
    private static int count = 0;
    //Array of sites stored within the Graph object
    Site sites[] = new Site[SIZE];

    //2D Matrix wherein the Graph data structure is actually stored
    private double[][]adjMatrix = new double[SIZE][SIZE];


    //Default constructor initialises all weights and edges to 0
    public Graph()
    {
        initialiseGraph();
    }

    //Initialises all nodes and edges to zero
    public void initialiseGraph()
    {
        for(int row = 0; row < SIZE; row++)
        {
            for(int col = 0; col < SIZE; col++ )
            {
                adjMatrix[row][col] = 0;
            }

        }
```

```
35      }
36
37
38      //Add a site to the array with a name, x and y value
39      public void addSite(String name, double x, double y)
40      {
41          sites[count] = new Site(name, x, y);
42          count++;
43          System.out.println("Site: " + name + " has been added as a node to the graph");
44      }
45
46
47      //Method to search and output the details of a given site
48      public void search(String site)
49      {
50          for(int i = 0; i < count; i++)
51          {
52              if(sites[i].getName().equals(site))
53              {
54                  //If found, print the Site toString Ovverride method
55                  System.out.println(sites[i].toString() + "\n");
56              }
57          }
58      }
59
60      //Method to check if a site name input by user is valid
61      public int isSite(String name)
62      {
63          for(int i = 0; i < count; i++)
64          {
65              //If the name input is assigned to the name of a site in the array sites
66              if(sites[i].getName().equals(name))
67              {
68                  return sites[i].getIndex();
69              }
70          }
71          return -1;
```

```
72      }
73
74      //Method to insert a weight between two nodes
75      public void insert(String site1, String site2, double weight)
76      {
77          //Assigns the return value of isSite to two variables
78          int site1Index = isSite(site1);
79          int site2Index = isSite(site2);
80
81          //If the two sites are valid
82          if(site1Index > -1 && site2Index > -1)
83          {
84              //Sets both indeces flipped on the diagonal to the weight, as it's an undirected graph
85              adjMatrix[site1Index][site2Index] = weight;
86              adjMatrix[site2Index][site1Index] = weight;
87              System.out.println(site1 + " has been connected to: " + site2 + " with a weight of: " + weight);
88              System.out.println();
89          }
90          else
91          {
92              System.out.println("Invalid!");
93          }
94      }
95
96      //Method to print the full Adjacency Maatrix for own visual purposes
97      public void printMatrix()
98      {
99          System.out.print("A,    B,      C,      D,      E,      F\n");
100
101         for(int row = 0; row < SIZE; row++)
102         {
103
104             for(int col = 0; col < SIZE; col++)
105             {
106
107                 System.out.print(adjMatrix[row][col] + ",    ");
108             }
```

```
                System.out.println();
            }

            System.out.println();
        }

        //Method to output list of sites that are connected to a given input site
        public void allCons(String site)
        {
            //Get the index of the input site
            int index = isSite(site);
            String outputString = "";
            //If valid (if isSite() returned a number that isn't -1 it is valid)
            if(index > -1)
            {
                //Loop through jus the col of the adjMatrix (we only care about the input site)
                for(int col = 0; col < count; col++)
                {
                    //If the weight is greater than 0 there is a connection
                    if(adjMatrix[index][col] > 0)
                    {
                        outputString += sites[col].getName() + ", ";
                    }
                }
            }
            System.out.println("List of Connected Sites to: " + site + " are - " + outputString);
        }

        //Method to find the smallest weight (distance) between two sites (nodes)
        public void closest(String site)
        {
            //Smallest initially set to arbitrary large value
            double smallest = 999999;
            int smallestIndex = 0;
            int index = isSite(site);
            //If valid
            if(index > -1)
```

```
        {
                //Loop through just the col of the adjMatrix (we only care about the input site)
                for(int col = 0; col < count; col++)
                {
                    //If the weight is greater than 0 and NOT 0 there is a connection, check if it is smaller then
                    if(adjMatrix[index][col] < smallest && adjMatrix[index][col] != 0)
                    {
                        smallest = adjMatrix[index][col];
                        smallestIndex = col;
                    }
                }
                System.out.println("Closest Site to " + sites[index].getName() + " is " + sites[smallestIndex].getName() + " with a weight of " + smallest);
        }
    }
}
```

Site.java

```java
//Name:     Mark Lambert, Darius Herdes
//Date:     10/12/2024
//Purpose:  Data Structures & Algorithms Final Assessment
//Class for Site Objects, allowing us to store extra information (name, coordinates, siteIndex etc)
package Graph;

public class Site {
    private String name;
    //Length 2; 0 for x, 1 for y
    private double coords[] = new double[2];
    private int siteIndex;
    private static int count = 0;

    //Constructors
    public Site()
    {
        setName("");
        setCoOrds(0,0);
    }

    public Site(String name, double x, double y)
    {
        setName(name);
        setCoOrds(x, y);
        setIndex();
    }

    //Set name of a site
    public void setName(String name)
    {
        this.name = name;
    }

    //Returns the name of a site
```

```
35      public String getName()
36      {
37          return name;
38      }
39
40      //Methods to get coordinates
41      public double getX() {
42          return coords[0];
43      }
44
45      public double getY() {
46          return coords[1];
47      }
48
49      //Method to set coordinates for Site
50      public void setCoOrds(double x, double y)
51      {
52          //Index 0 = x coords
53          coords[0] = x;
54          //Index 1 = y coords
55          coords[1] = y;
56      }
57
58      //Method used to set the index of the site
59      public void setIndex()
60      {
61          siteIndex = count;
62          count++;
63      }
64
65      public int getIndex()
66      {
67          return siteIndex;
68      }
69
70      public String toString()
71      {
```

```
72          return "-------------Site Search Details-------------" +
73                  "\nSite Name:       " + name +
74                  "\nCo-ordinates (x,y): "+ coords[0] + ", " + coords[1];
75      }
76  }
77
```

driver.java

```java
//Name:     Mark Lambert, Darius Herdes
//Date:     10/12/2024
//Purpose:  Data Structures & Algorithms Final Assessment
//Main class
package Graph;

public class driver {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Graph g = new Graph();

        //Node A
        g.addSite("Rock of Donamase", 52.637827, -6.7937563);
        //Node B
        g.addSite("O'Moore Park", 52.662661, -6.6598326);
        //Node C
        g.addSite("Donamase Art Centre", 52.690146, -6.650620);
        //Node D
        g.addSite("The Heath Golf Club", 52.682317, -6.581056);
        //Node E
        g.addSite("Emo Court", 52.637440, -6.622554);
        //Node F
        g.addSite("Portlaoise Herritage Hotel", 52.590639, -6.499220);

        //Print new line for interface purposes
        System.out.println();


        //Sample search calls
        g.search("Rock of Donamase");
        g.search("Portlaoise Herritage Hotel");
        g.search("Emo Court");

```

```
35          //Print new line for interface purposes
36          System.out.println();
37
38
39          //Insertion of node A associated edges
40          g.insert("Rock of Donamase", "Emo Court", 7.71);
41          g.insert("Rock of Donamase", "The Heath Golf Club", 2.41);
42          g.insert("Rock of Donamase", "O'Moore Park", 3.68);
43
44          //Insertion of node B associated edges
45          g.insert("O'Moore Park", "Donamase Art Centre", 6.01);
46
47          //Insertion of node C associated edges
48          g.insert("Donamase Art Centre", "The Heath Golf Club", 5.52);
49          g.insert("Donamase Art Centre", "Portlaoise Herritage Hotel", 8.42);
50
51          //Insertion of node E associated edges
52          g.insert("Emo Court", "Portlaoise Herritage Hotel", 9.97);
53
54          //Remaining associations complete as un-dirrected graph completes the reverse association automatically
55
56          g.allCons("Rock of Donamase");
57          g.closest("Rock of Donamase");
58      }
59
60  }
61
```