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;
            }

        }
    }


    //Add a site to the array with a name, x and y value
    public void addSite(String name, double x, double y)
    {
        sites[count] = new Site(name, x, y);
        count++;
        System.out.println("Site: " + name + " has been added as a node to the graph");
    }


    //Method to search and output the details of a given site
    public void search(String site)
    {
        for(int i = 0; i < count; i++)
        {
            if(sites[i].getName().equals(site))
            {
                //If found, print the Site toString Ovverride method
                System.out.println(sites[i].toString() + "\n");
            }
        }
    }

    //Method to check if a site name input by user is valid
    public int isSite(String name)
    {
        for(int i = 0; i < count; i++)
        {
            //If the name input is assigned to the name of a site in the array sites
            if(sites[i].getName().equals(name))
            {
                return sites[i].getIndex();
            }
        }
        return -1;
    }

    //Method to insert a weight between two nodes
    public void insert(String site1, String site2, double weight)
    {
        //Assigns the return value of isSite to two variables
        int site1Index = isSite(site1);
        int site2Index = isSite(site2);

        //If the two sites are valid
        if(site1Index > -1 && site2Index > -1)
        {
            //Sets both indeces flipped on the diagonal to the weight, as it's an undirected graph
            adjMatrix[site1Index][site2Index] = weight;
            adjMatrix[site2Index][site1Index] = weight;
            System.out.println(site1 + " has been connected to: " + site2 + " with a weight of: " + weight);
            System.out.println();
        }
        else
        {
            System.out.println("Invalid!");
        }
    }

    //Method to print the full Adjacency Maatrix for own visual purposes
    public void printMatrix()
    {
        System.out.print("A,    B,      C,      D,      E,      F\n");

        for(int row = 0; row < SIZE; row++)
        {

            for(int col = 0; col < SIZE; col++)
            {

                System.out.print(adjMatrix[row][col] + ",   ");
            }
```

```java
                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);
        }
    }
}


```