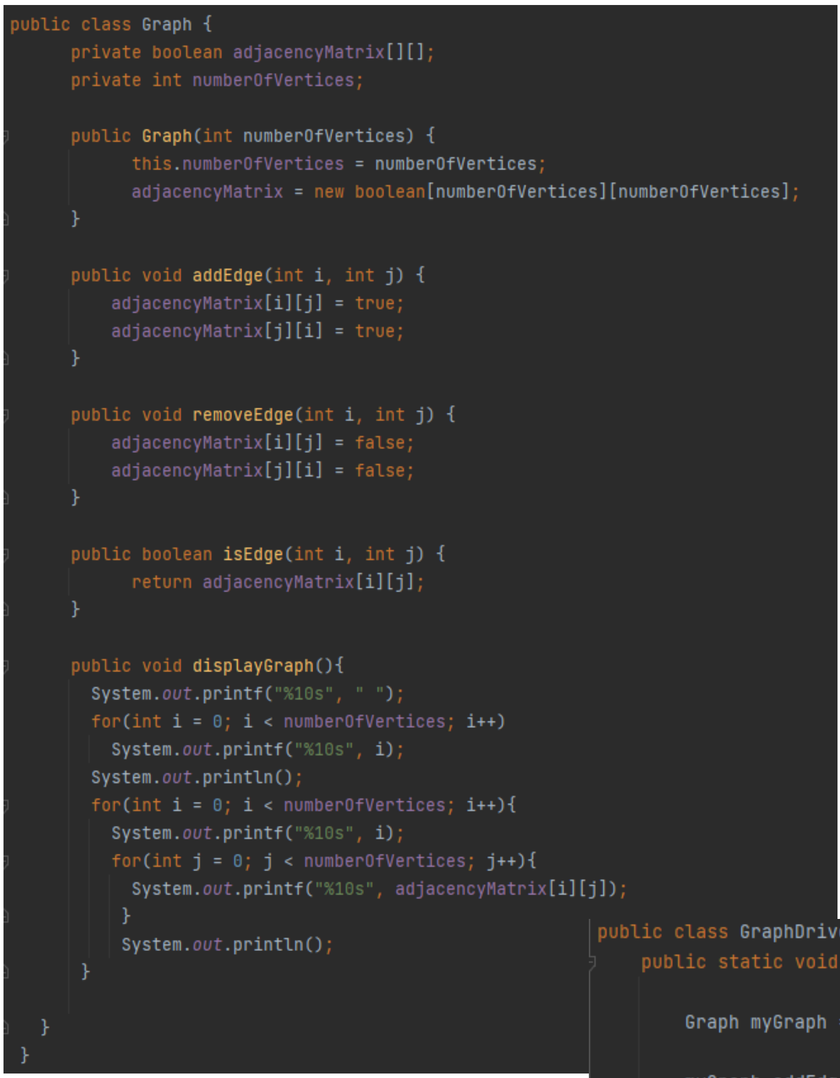
ICS-202 Lab-09 Report

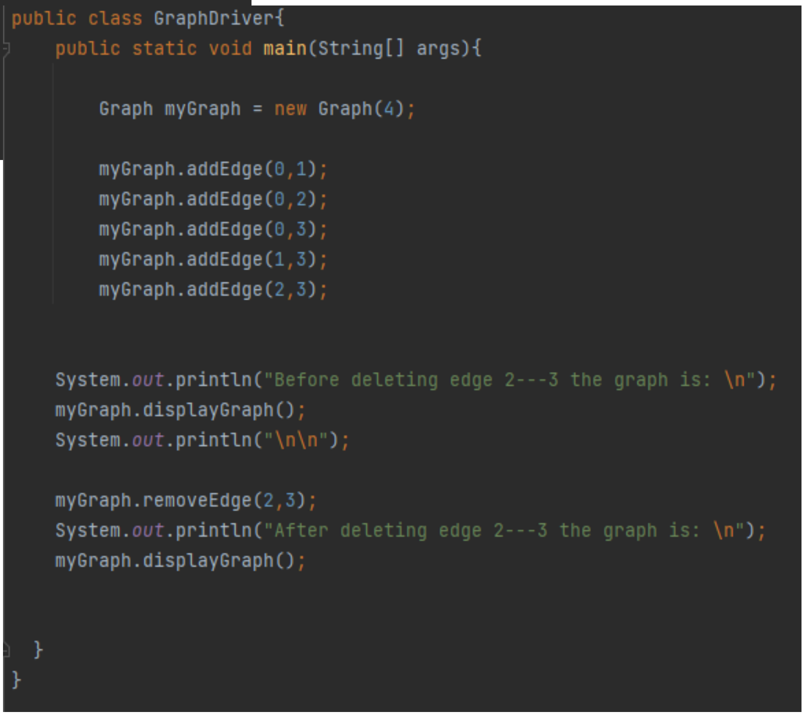
Mohammed Busaleh - 202158210

Task 1:

Code:

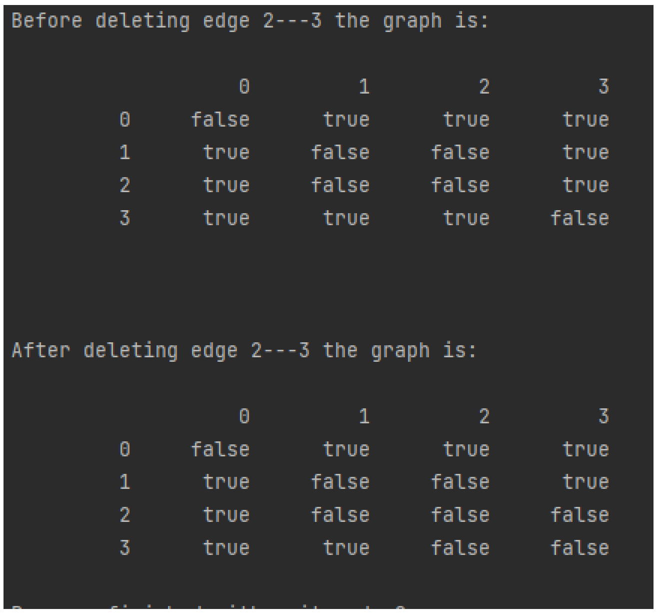


```java
public class Graph {
    private boolean adjacencyMatrix[][];
    private int numberOfVertices;

    public Graph(int numberOfVertices) {
        this.numberOfVertices = numberOfVertices;
        adjacencyMatrix = new boolean[numberOfVertices][numberOfVertices];
    }

    public void addEdge(int i, int j) {
        adjacencyMatrix[i][j] = true;
        adjacencyMatrix[j][i] = true;
    }

    public void removeEdge(int i, int j) {
        adjacencyMatrix[i][j] = false;
        adjacencyMatrix[j][i] = false;
    }

    public boolean isEdge(int i, int j) {
        return adjacencyMatrix[i][j];
    }

    public void displayGraph(){
      System.out.printf("%10s", " ");
      for(int i = 0; i < numberOfVertices; i++)
        System.out.printf("%10s", i);
      System.out.println();
      for(int i = 0; i < numberOfVertices; i++){
        System.out.printf("%10s", i);
        for(int j = 0; j < numberOfVertices; j++){
          System.out.printf("%10s", adjacencyMatrix[i][j]);
         }
         System.out.println();
     }

  }
}
```



```java
public class GraphDriver{
    public static void main(String[] args){

        Graph myGraph = new Graph(4);

        myGraph.addEdge(0,1);
        myGraph.addEdge(0,2);
        myGraph.addEdge(0,3);
        myGraph.addEdge(1,3);
        myGraph.addEdge(2,3);


    System.out.println("Before deleting edge 2---3 the graph is: \n");
    myGraph.displayGraph();
    System.out.println("\n\n");

    myGraph.removeEdge(2,3);
    System.out.println("After deleting edge 2---3 the graph is: \n");
    myGraph.displayGraph();


  }
}
```

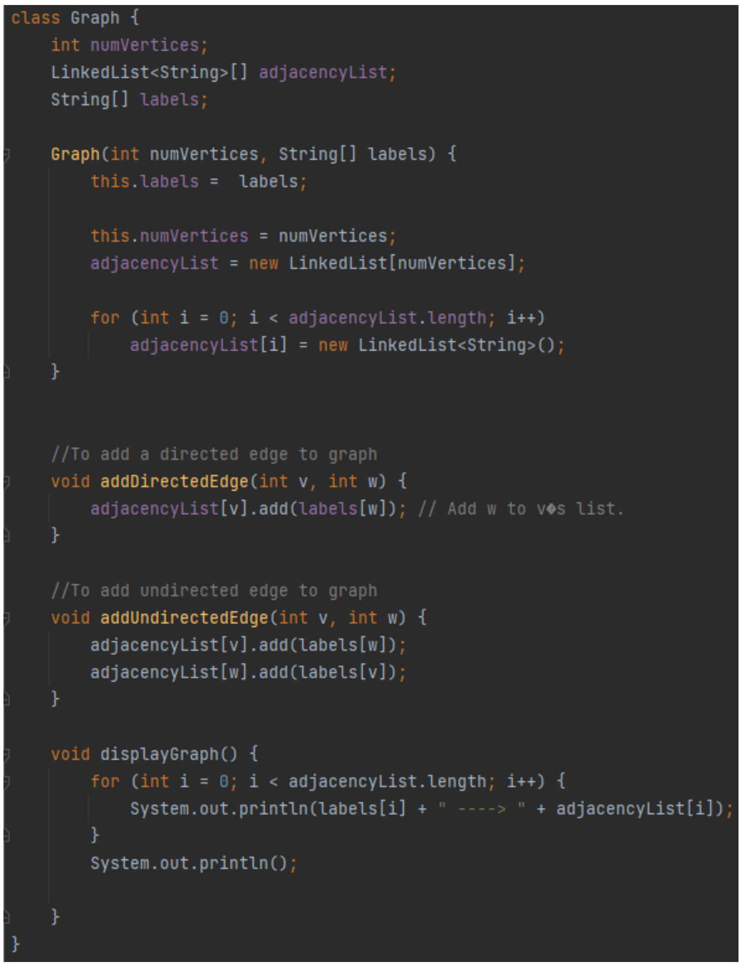
Output:



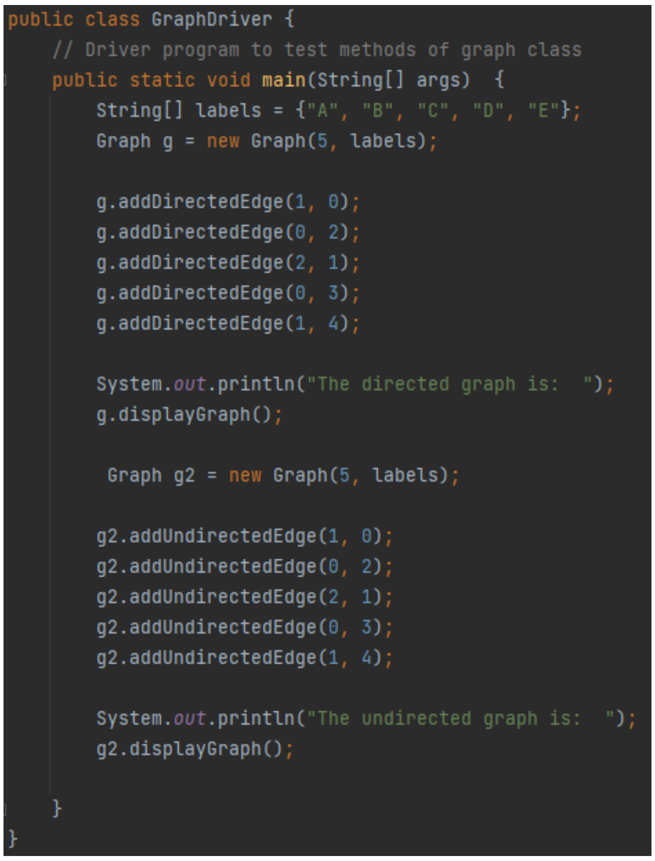
```
Before deleting edge 2---3 the graph is:

                0         1         2         3
          0   false     true      true      true
          1    true    false     false      true
          2    true    false     false      true
          3    true     true      true     false



After deleting edge 2---3 the graph is:

                0         1         2         3
          0   false     true      true      true
          1    true    false     false      true
          2    true    false     false     false
          3    true     true     false     false
```

Task 2:

Code:

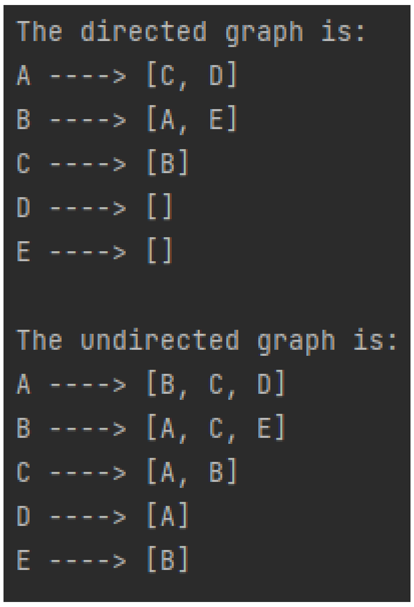


```java
class Graph {
    int numVertices;
    LinkedList<String>[] adjacencyList;
    String[] labels;

    Graph(int numVertices, String[] labels) {
        this.labels =  labels;

        this.numVertices = numVertices;
        adjacencyList = new LinkedList[numVertices];

        for (int i = 0; i < adjacencyList.length; i++)
            adjacencyList[i] = new LinkedList<String>();
    }


    //To add a directed edge to graph
    void addDirectedEdge(int v, int w) {
        adjacencyList[v].add(labels[w]); // Add w to v�s list.
    }

    //To add undirected edge to graph
    void addUndirectedEdge(int v, int w) {
        adjacencyList[v].add(labels[w]);
        adjacencyList[w].add(labels[v]);
    }

    void displayGraph() {
        for (int i = 0; i < adjacencyList.length; i++) {
            System.out.println(labels[i] + " ----> " + adjacencyList[i]);
        }
        System.out.println();

    }
}
```



```java
public class GraphDriver {
    // Driver program to test methods of graph class
    public static void main(String[] args)  {
        String[] labels = {"A", "B", "C", "D", "E"};
        Graph g = new Graph(5, labels);

        g.addDirectedEdge(1, 0);
        g.addDirectedEdge(0, 2);
        g.addDirectedEdge(2, 1);
        g.addDirectedEdge(0, 3);
        g.addDirectedEdge(1, 4);

        System.out.println("The directed graph is:  ");
        g.displayGraph();

         Graph g2 = new Graph(5, labels);

        g2.addUndirectedEdge(1, 0);
        g2.addUndirectedEdge(0, 2);
        g2.addUndirectedEdge(2, 1);
        g2.addUndirectedEdge(0, 3);
        g2.addUndirectedEdge(1, 4);

        System.out.println("The undirected graph is:  ");
        g2.displayGraph();

    }
}
```

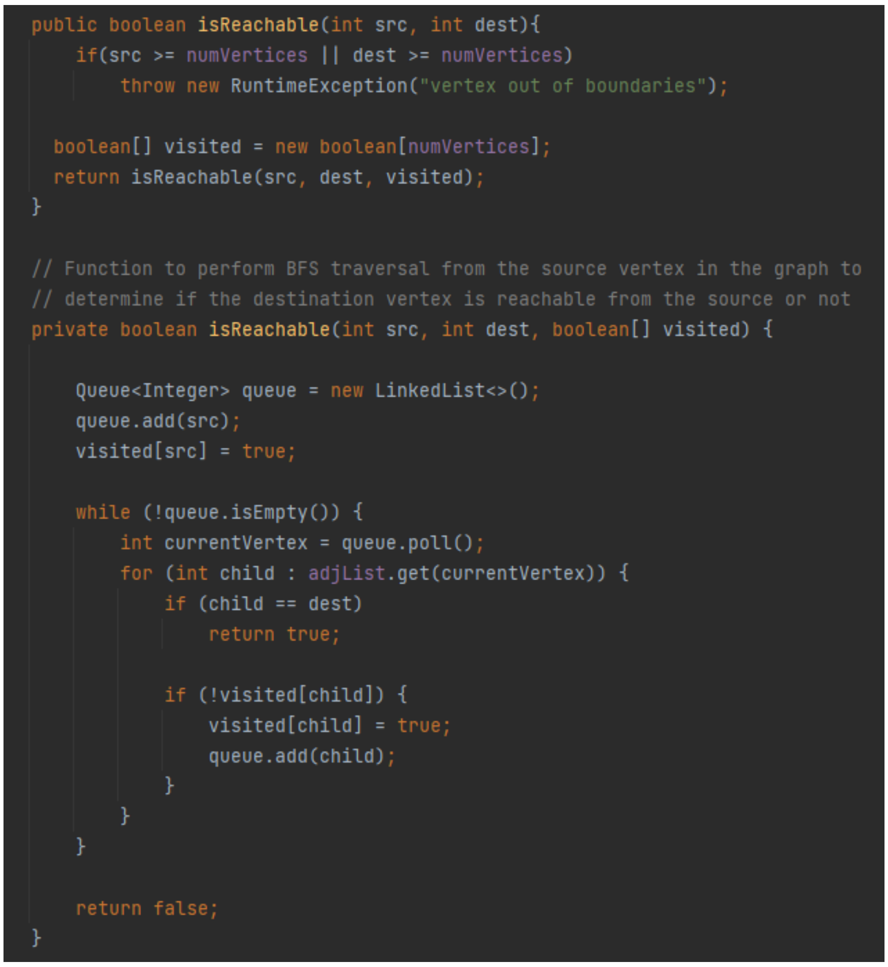
Output:



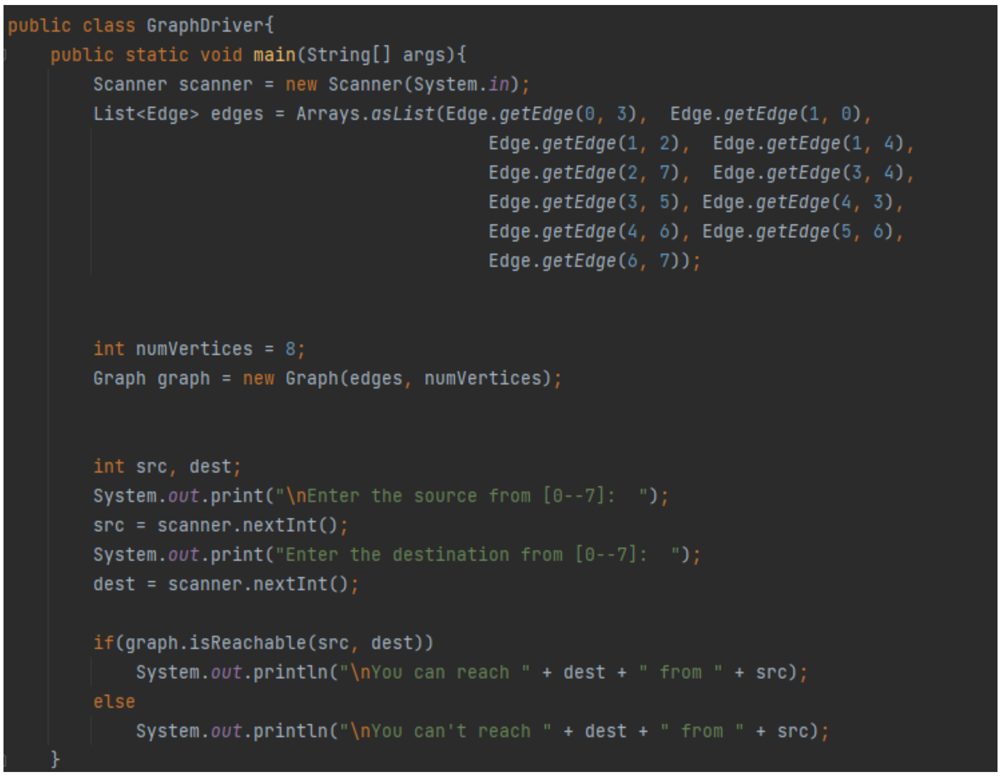
```
The directed graph is:
A ----> [C, D]
B ----> [A, E]
C ----> [B]
D ----> []
E ----> []

The undirected graph is:
A ----> [B, C, D]
B ----> [A, C, E]
C ----> [A, B]
D ----> [A]
E ----> [B]
```

Task 3:

Code:

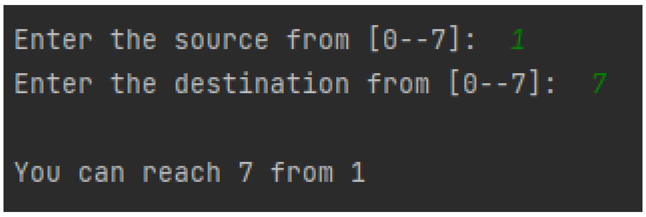


```java
public boolean isReachable(int src, int dest){
    if(src >= numVertices || dest >= numVertices)
        throw new RuntimeException("vertex out of boundaries");

  boolean[] visited = new boolean[numVertices];
  return isReachable(src, dest, visited);
}

// Function to perform BFS traversal from the source vertex in the graph to
// determine if the destination vertex is reachable from the source or not
private boolean isReachable(int src, int dest, boolean[] visited) {

    Queue<Integer> queue = new LinkedList<>();
    queue.add(src);
    visited[src] = true;

    while (!queue.isEmpty()) {
        int currentVertex = queue.poll();
        for (int child : adjList.get(currentVertex)) {
            if (child == dest)
                return true;

            if (!visited[child]) {
                visited[child] = true;
                queue.add(child);
            }
        }
    }

    return false;
}
```

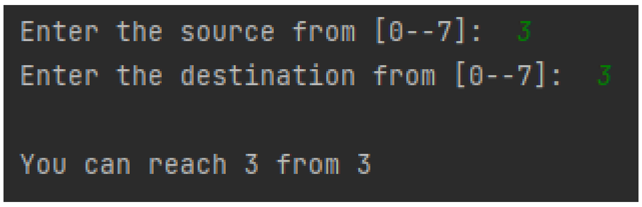


```java
public class GraphDriver{
    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        List<Edge> edges = Arrays.asList(Edge.getEdge(0, 3),  Edge.getEdge(1, 0),
                                         Edge.getEdge(1, 2),  Edge.getEdge(1, 4),
                                         Edge.getEdge(2, 7),  Edge.getEdge(3, 4),
                                         Edge.getEdge(3, 5), Edge.getEdge(4, 3),
                                         Edge.getEdge(4, 6), Edge.getEdge(5, 6),
                                         Edge.getEdge(6, 7));


        int numVertices = 8;
        Graph graph = new Graph(edges, numVertices);


        int src, dest;
        System.out.print("\nEnter the source from [0--7]:  ");
        src = scanner.nextInt();
        System.out.print("Enter the destination from [0--7]:  ");
        dest = scanner.nextInt();

        if(graph.isReachable(src, dest))
            System.out.println("\nYou can reach " + dest + " from " + src);
        else
            System.out.println("\nYou can't reach " + dest + " from " + src);
    }
```

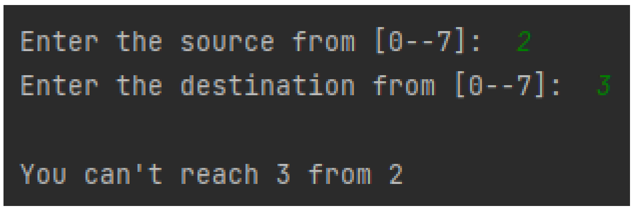
Output:



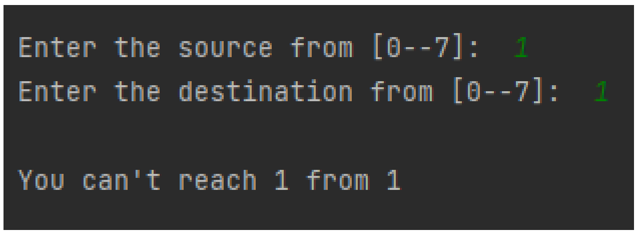
```
Enter the source from [0--7]:  1
Enter the destination from [0--7]:  7

You can reach 7 from 1
```



```
Enter the source from [0--7]:  3
Enter the destination from [0--7]:  3

You can reach 3 from 3
```



```
Enter the source from [0--7]:  2
Enter the destination from [0--7]:  3

You can't reach 3 from 2
```



```
Enter the source from [0--7]:  1
Enter the destination from [0--7]:  1

You can't reach 1 from 1
```