Sarah Kirby

Module 9 Assignment

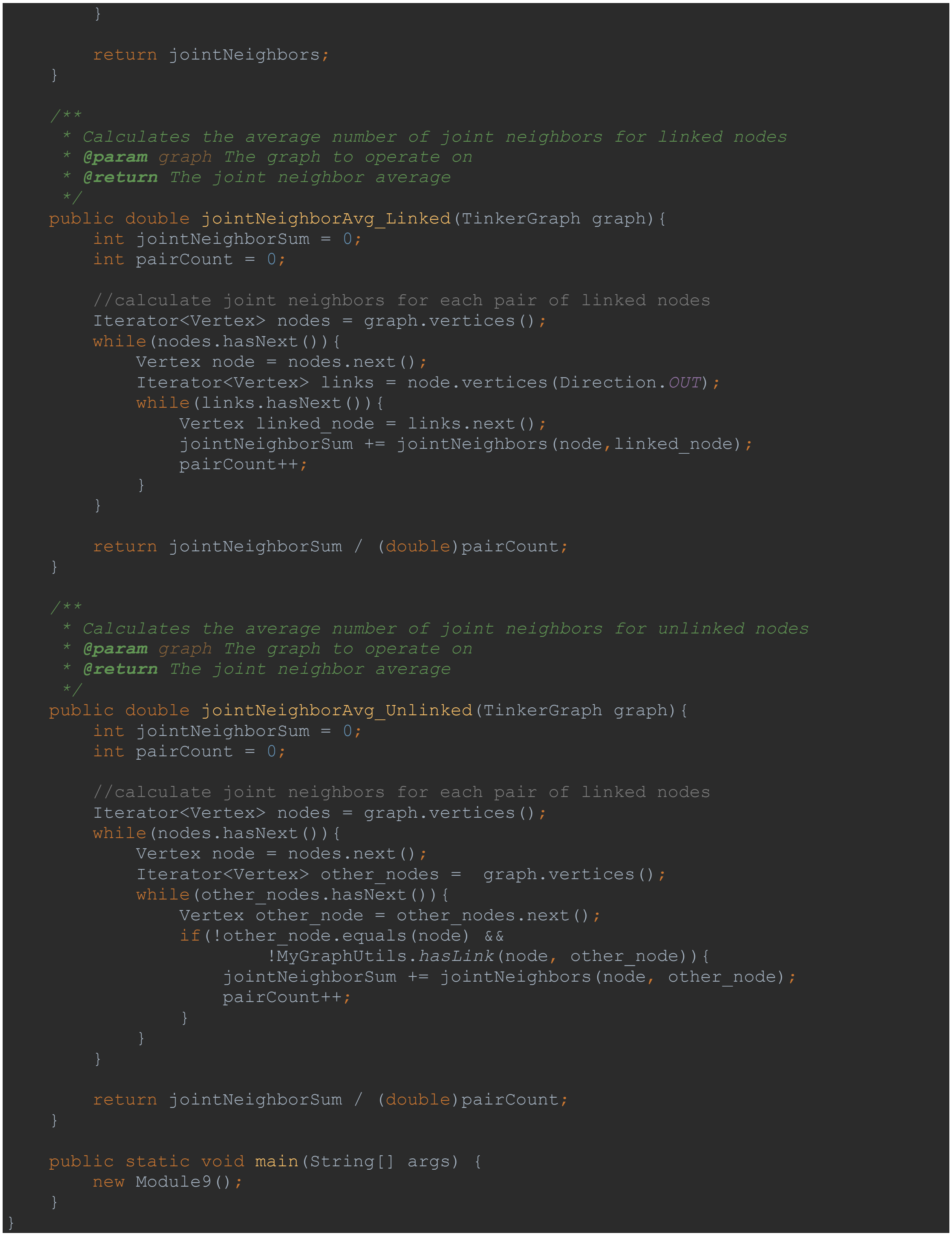
Output:

Average joint neighbors of linked nodes: 2.8861031518624642

Average joint neighbors of unlinked nodes: 0.23764582673713436

Program:

```
package assignments;

import graph.GraphUtils;
import org.apache.tinkerpop.gremlin.structure.Direction;
import org.apache.tinkerpop.gremlin.structure.Vertex;
import org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerGraph;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class Module9 {

    private String GRAPH_INPUT = "GraphDatabases\\students.graphml";

    public Module9(){
        TinkerGraph graph = GraphUtils.readGraphML(GRAPH_INPUT);

        System.out.println("Average joint neighbors of linked nodes: " +
                jointNeighborAvg_Linked(graph));

        System.out.println("Average joint neighbors of unlinked nodes: " +
                jointNeighborAvg_Unlinked(graph));
    }

    /**
     * Counts the number of joint neighbors of two nodes
     * @param v1
     * @param v2
     * @return
     */
    public int jointNeighbors(Vertex v1, Vertex v2){
        int jointNeighbors = 0;
        Map<Integer, Vertex> v1_neighbors = new HashMap<>();

        Iterator<Vertex> neighbors = v1.vertices(Direction.BOTH);
        while(neighbors.hasNext()){
            Vertex neighbor = neighbors.next();
            v1_neighbors.put(neighbor.hashCode(), neighbor);
        }

        neighbors = v2.vertices(Direction.BOTH);
        while(neighbors.hasNext()){
            Vertex neighbor = neighbors.next();
            if( !neighbor.equals(v1) && v1_neighbors.containsValue(neighbor)){
                jointNeighbors++;
            }
```



```java
        }

        return jointNeighbors;
    }

    /**
     * Calculates the average number of joint neighbors for linked nodes
     * @param graph The graph to operate on
     * @return The joint neighbor average
     */
    public double jointNeighborAvg_Linked(TinkerGraph graph){
        int jointNeighborSum = 0;
        int pairCount = 0;

        //calculate joint neighbors for each pair of linked nodes
        Iterator<Vertex> nodes = graph.vertices();
        while(nodes.hasNext()){
            Vertex node = nodes.next();
            Iterator<Vertex> links = node.vertices(Direction.OUT);
            while(links.hasNext()){
                Vertex linked_node = links.next();
                jointNeighborSum += jointNeighbors(node,linked_node);
                pairCount++;
            }
        }

        return jointNeighborSum / (double)pairCount;
    }

    /**
     * Calculates the average number of joint neighbors for unlinked nodes
     * @param graph The graph to operate on
     * @return The joint neighbor average
     */
    public double jointNeighborAvg_Unlinked(TinkerGraph graph){
        int jointNeighborSum = 0;
        int pairCount = 0;

        //calculate joint neighbors for each pair of linked nodes
        Iterator<Vertex> nodes = graph.vertices();
        while(nodes.hasNext()){
            Vertex node = nodes.next();
            Iterator<Vertex> other_nodes =  graph.vertices();
            while(other_nodes.hasNext()){
                Vertex other_node = other_nodes.next();
                if(!other_node.equals(node)  &&
                        !MyGraphUtils.hasLink(node, other_node)){
                    jointNeighborSum += jointNeighbors(node, other_node);
                    pairCount++;
                }
            }
        }

        return jointNeighborSum / (double)pairCount;
    }

    public static void main(String[] args) {
        new Module9();
    }
}
```