import org.junit.Test;

import java.awt.\*;

import java.util.List;

import static org.junit.Assert.\*;

/\*\*

 \* Testing strategy for the Graph class

 \*

 \* Partition the inputs as follows:

 \*

 \* For setSource(Node node) and setDestination(Node node):

 \* - Test with a node that is in the list of nodes and verify that the source/destination is set.

 \* - Test with a node that is not in the list of nodes and expect IllegalArgumentException.

 \*

 \* For getNodeCount():

 \* - Test when the graph is empty, and expect the count to be INITIAL\_NODE\_ID.

 \* - Test when the graph has one node, and expect the count to be INITIAL\_NODE\_ID + 1.

 \* - Test when the graph has multiple nodes, and expect the count to be accurate.

 \*

 \* For isFirstNode(Node node):

 \* - Test with the first node added to the graph, and expect true.

 \* - Test with a node that is not the first node, and expect false.

 \*

 \* For addNode(Point coordinates) and addNode(Node node):

 \* - Test adding a node and verify that the node is in the list of nodes.

 \* - Test adding multiple nodes and verify that all nodes are in the list of nodes.

 \* - Test adding a node and verify that the source is set if it's the first node.

 \*

 \* For addEdge(Edge newEdge):

 \* - Test adding a new edge and verify that it is in the list of edges.

 \* - Test adding an equivalent edge and verify that the list of edges remains unchanged.

 \*

 \* For deleteNode(Node node):

 \* - Test deleting a node and verify that it is no longer in the list of nodes.

 \* - Test deleting a node and verify that edges associated with it are removed from the list of edges.

 \*

 \* For clear():

 \* - Test clearing the graph and verify that nodes, edges, solved status, source, and destination are reset.

 \*

 \* For hasNodeInEdges(Node node):

 \* - Test with a node that is present in an edge and expect true.

 \* - Test with a node that is not present in any edge and expect false.

 \*

 \* Exhaustive Cartesian coverage of partitions for each method.

 \*/

public class GraphTest {

    private static final int INITIAL\_NODE\_ID = 1;

    @Test

    // covers setSource(Node node) and setDestination(Node node)

    //        with a node that is in the list of nodes

    public void testSetSourceAndDestinationWithValidNode() {

        Graph graph = new Graph();

        Node node = new Node(new Point(0, 0));

        graph.addNode(node);

        graph.setSource(node);

        assertEquals(node, graph.source);

        graph.setDestination(node);

        assertEquals(node, graph.destination);

    }

    @Test(expected = IllegalArgumentException.class)

    // covers setSource(Node node) and setDestination(Node node)

    //        with a node that is not in the list of nodes

    public void testSetSourceAndDestinationWithInvalidNode() {

        Graph graph = new Graph();

        Node validNode = new Node(new Point(0, 0));

        Node invalidNode = new Node(new Point(1, 1));

        graph.addNode(validNode);

        graph.setSource(invalidNode); // Expect IllegalArgumentException

    }

    @Test

    // covers getNodeCount() when the graph is empty

    public void testGetNodeCountEmptyGraph() {

        Graph graph = new Graph();

        assertEquals(INITIAL\_NODE\_ID, graph.getNodeCount());

    }

    @Test

    // covers getNodeCount() when the graph has one node

    public void testGetNodeCountSingleNodeGraph() {

        Graph graph = new Graph();

        graph.addNode(new Point(0, 0));

        assertEquals(INITIAL\_NODE\_ID + 1, graph.getNodeCount());

    }

    @Test

    // covers getNodeCount() when the graph has multiple nodes

    public void testGetNodeCountMultipleNodesGraph() {

        Graph graph = new Graph();

        graph.addNode(new Point(0, 0));

        graph.addNode(new Point(1, 1));

        graph.addNode(new Point(2, 2));

        assertEquals(INITIAL\_NODE\_ID + 3, graph.getNodeCount());

    }

    @Test

    // covers isFirstNode(Node node) with the first node added to the graph

    public void testIsFirstNodeWithFirstNode() {

        Graph graph = new Graph();

        Node firstNode = new Node(new Point(0, 0));

        assertTrue(graph.isFirstNode(firstNode));

    }

    @Test

    // covers isFirstNode(Node node) with a node that is not the first node

    public void testIsFirstNodeWithNonFirstNode() {

        Graph graph = new Graph();

        graph.addNode(new Point(0, 0));

        Node nonFirstNode = new Node(new Point(1, 1));

        assertFalse(graph.isFirstNode(nonFirstNode));

    }

    @Test

    // covers addNode(Point coordinates) and addNode(Node node)

    //        adding a node and verify that the node is in the list of nodes

    public void testAddNode() {

        Graph graph = new Graph();

        Point coordinates = new Point(0, 0);

        assertTrue(graph.addNode(coordinates));

        Node addedNode = graph.nodes.get(0);

        assertEquals(coordinates, addedNode.getCoordinates());

    }

    @Test

    // covers addNode(Point coordinates) and addNode(Node node)

    //        adding multiple nodes and verify that all nodes are in the list of nodes

    public void testAddMultipleNodes() {

        Graph graph = new Graph();

        Point coordinates1 = new Point(0, 0);

        Point coordinates2 = new Point(1, 1);

        assertTrue(graph.addNode(coordinates1));

        assertTrue(graph.addNode(coordinates2));

        assertEquals(2, graph.getNodeCount());

    }

    @Test

    // covers addNode(Point coordinates) and addNode(Node node)

    //        adding a node and verify that the source is set if it's the first node

    public void testAddNodeSetsSource() {

        Graph graph = new Graph();

        Point coordinates = new Point(0, 0);

        assertTrue(graph.addNode(coordinates));

        Node addedNode = graph.nodes.get(0);

        assertEquals(addedNode, graph.source);

    }

    @Test

    // covers addEdge(Edge newEdge)

    //        adding a new edge and verify that it is in the list of edges

    public void testAddEdge() {

        Graph graph = new Graph();

        Node node1 = new Node(new Point(0, 0));

        Node node2 = new Node(new Point(1, 1));

        graph.addNode(node1);

        graph.addNode(node2);

        Edge edge = new Edge(node1, node2);

        assertTrue(graph.addEdge(edge));

        assertTrue(graph.edges.contains(edge));

    }

    @Test

    // covers addEdge(Edge newEdge)

    //        adding an equivalent edge and verify that the list of edges remains unchanged

    public void testAddEquivalentEdge() {

        Graph graph = new Graph();

        Node node1 = new Node(new Point(0, 0));

        Node node2 = new Node(new Point(1, 1));

        graph.addNode(node1);

        graph.addNode(node2);

        Edge edge1 = new Edge(node1, node2);

        Edge edge2 = new Edge(node2, node1);

        assertTrue(graph.addEdge(edge1));

        assertFalse(graph.addEdge(edge2)); // Equivalent edge, expect false

        assertEquals(1, graph.edges.size());

    }

    @Test

    // covers deleteNode(Node node)

    //        deleting a node and verify that it is no longer in the list of nodes

    public void testDeleteNode() {

        Graph graph = new Graph();

        Node node1 = new Node(new Point(0, 0));

        Node node2 = new Node(new Point(1, 1));

        graph.addNode(node1);

        graph.addNode(node2);

        Edge edge = new Edge(node1, node2);

        graph.addEdge(edge);

        graph.deleteNode(node1);

        assertFalse(graph.nodes.contains(node1));

    }

    @Test

    // covers deleteNode(Node node)

    //        deleting a node and verify that edges associated with it are removed from the list of edges

    public void testDeleteNodeRemovesAssociatedEdges() {

        Graph graph = new Graph();

        Node node1 = new Node(new Point(0, 0));

        Node node2 = new Node(new Point(1, 1));

        Node node3 = new Node(new Point(2, 2));

        graph.addNode(node1);

        graph.addNode(node2);

        graph.addNode(node3);

        Edge edge1 = new Edge(node1, node2);

        Edge edge2 = new Edge(node2, node3);

        Edge edge3 = new Edge(node1, node3);

        graph.addEdge(edge1);

        graph.addEdge(edge2);

        graph.addEdge(edge3);

        graph.deleteNode(node2);

        assertFalse(graph.edges.contains(edge1));

        assertFalse(graph.edges.contains(edge2));

    }

    @Test

    // covers clear()

    //        clearing the graph and verify that nodes, edges, solved status, source, and destination are reset

    public void testClear() {

        Graph graph = new Graph();

        Node node = new Node(new Point(0, 0));

        graph.addNode(node);

        graph.addEdge(new Edge(node, new Node(new Point(1, 1))));

        graph.setSource(node);

        graph.setDestination(node);

        graph.solved = true;

        graph.clear();

        assertTrue(graph.nodes.isEmpty());

        assertTrue(graph.edges.isEmpty());

        assertNull(graph.source);

        assertNull(graph.destination);

        assertFalse(graph.solved);

        assertEquals(INITIAL\_NODE\_ID, graph.getNodeCount());

    }

    @Test

    // covers hasNodeInEdges(Node node) with a node that is present in an edge

    public void testHasNodeInEdgesNodePresent() {

        Graph graph = new Graph();

        Node node1 = new Node(new Point(0, 0));

        Node node2 = new Node(new Point(1, 1));

        graph.addNode(node1);

        graph.addNode(node2);

        Edge edge = new Edge(node1, node2);

        graph.addEdge(edge);

        assertTrue(graph.hasNodeInEdges(node1));

    }

    @Test

    // covers hasNodeInEdges(Node node) with a node that is not present in any edge

    public void testHasNodeInEdgesNodeNotPresent() {

        Graph graph = new Graph();

        Node node = new Node(new Point(0, 0));

        graph.addNode(node);

        assertFalse(graph.hasNodeInEdges(node));

    }

}