1 Problem 2: Write Tests for Graph

1.1 Write-up Paragraph:

For my testing strategy, I started with testing the cases where it should be invalid, using the strategy equivalence partitionings. Test cases should cover both valid and invalid inputs, so I accounted for it in my tests. I then created scenarios where the method should work correctly and created tests for that. For example, if you call addNode with "A", I used my contains method to check if A is in the graph. I also used the random library for labels, to test cases where there can be multiple of the same edges with different labels.

1.2 All my Test Cases:

```
class GraphTest {

private Graph graph;

@BeforeEach
void setUp() {
    // Initialize the graph before each test
    graph = new Graph();
}

@Test
void testAddNode() {
    graph.addNode("A");
    assertTrue(graph.containsNode("A"));

    assertThrows(IllegalArgumentException.class, () -> graph.addNode(""));

    assertThrows(IllegalArgumentException.class, () -> graph.addNode("A"));
}

@Test
void testAddEdge() {
 assertThrows(IllegalArgumentException.class, () -> graph.addEdge("", "B", "label"));
    assertThrows(IllegalArgumentException.class, () -> graph.addEdge("A", "", "label"));
    assertThrows(IllegalArgumentException.class, () -> graph.addEdge("A", "B", ""));

    graph.addNode("A");
    graph.addNode("B");
```

```
        // Generate a random label for the edge
        String edgeLabel = String.valueOf(new Random().nextInt(1000));
        graph.addEdge("A", "B", edgeLabel);
        assertEquals(edgeLabel, graph.getEdgeLabel("A", "B"));
    }

    @Test
    void testRemoveNode() {
     assertThrows(IllegalArgumentException.class, () -> graph.removeNode(""));
        assertThrows(IllegalArgumentException.class, () -> graph.removeNode("A"));

        graph.addNode("A");
        graph.removeNode("A");
        assertFalse(graph.containsNode("A"));
    }

    @Test
    void testRemoveEdge() {
     assertThrows(IllegalArgumentException.class, () -> graph.removeEdge("", "B", "label"));
        assertThrows(IllegalArgumentException.class, () -> graph.removeEdge("A", "", "label"))
        assertThrows(IllegalArgumentException.class, () -> graph.removeEdge("A", "B", ""));
        assertThrows(IllegalArgumentException.class, () -> graph.removeEdge("A", "B", "label"))

        graph.addNode("A");
        graph.addNode("B");
        // Generate a random label for the edge
        String edgeLabel = String.valueOf(new Random().nextInt(1000));
        graph.addEdge("A", "B", edgeLabel);
        graph.removeEdge("A", "B", edgeLabel);
        assertNull(graph.getEdgeLabel("A", "B"));
    }

    @Test
    void testListNodes() {
        graph.addNode("A");
        graph.addNode("B");
        Iterator<String> nodes = graph.listNodes();
        assertTrue(nodes.hasNext());
        assertEquals("A", nodes.next());
        assertEquals("B", nodes.next());
        assertFalse(nodes.hasNext());
    }

    @Test
    void testListChildren() {
        // Test listing children when the graph is empty
        assertThrows(IllegalArgumentException.class, () -> graph.listChildren("A"));

        // Test listing children for a node with no children
        graph.addNode("A");
        Iterator<String> childrenIterator = graph.listChildren("A");
        assertFalse(childrenIterator.hasNext());

        // Test listing children for a node with children
        graph.addNode("B");
        graph.addNode("C");
        graph.addEdge("A", "A", "label1");
        graph.addEdge("A", "B", "label1");
        graph.addEdge("A", "B", "label2");
        graph.addEdge("A", "B", "label3");
        graph.addEdge("A", "C", "label1");


        childrenIterator = graph.listChildren("A");

        List<String> expectedChildren = Arrays.asList("A(label1)", "B(label1)",
                                        "B(label2)", "B(label3)", "C(label1)");
        List<String> actualChildren = new ArrayList<>();


        while(childrenIterator.hasNext()) {
            actualChildren.add(childrenIterator.next());
        }


        assertTrue(actualChildren.containsAll(expectedChildren)
                    && expectedChildren.containsAll(actualChildren));
    }
    @Test
    void testContainsNode() {
     assertThrows(IllegalArgumentException.class, () -> graph.containsNode(""));
        graph.addNode("A");
        assertTrue(graph.containsNode("A"));
        assertFalse(graph.containsNode("B"));
    }

    @Test
    void testGetEdgeLabel() {
     assertThrows(IllegalArgumentException.class, () -> graph.getEdgeLabel("", "B"));
        assertThrows(IllegalArgumentException.class, () -> graph.getEdgeLabel("A", ""));
        assertThrows(IllegalArgumentException.class, () -> graph.getEdgeLabel("A", "B"));

        graph.addNode("A");
        graph.addNode("B");
        // Generate a random label for the edge
        String edgeLabel = String.valueOf(new Random().nextInt(1000));
        graph.addEdge("A", "B", edgeLabel);
        assertEquals(edgeLabel, graph.getEdgeLabel("A", "B"));
        assertNull(graph.getEdgeLabel("B", "A"));
    }
}
```

1.3 Additional Tests

I already added the new tests to the original tests while I was coding, so I feel like my test cases are now sufficient. A few new things I added was fixing the assertions where the code throws an error for an invalid input. I also redid the listChildren so it accounted for more edges going out of the same parent to the same child.