

Assignment 4 Solution - ONLY CODE

SFWR ENG 2AA4

April 2, 2009

The purpose of this software design exercise was to create, use and test a Java program that stores a map of a hallway network, together with the location of eggs, blockages and openings.

A PointT.java

```
/**
 * Author: S. Smith
 * Revised: March 2, 2009
 *
 * Description: Point ADT class
 */
import static java.lang.Math.*;

public class PointT
{
    public static final double TOLERANCE = 1e-4;

    protected double xc;
    protected double yc;

    public PointT(double x, double y)
    {
        xc = x;
        yc = y;
    }

    public double xcoord()
    {
        return xc;
    }

    public double ycoord()
    {
        return yc;
    }

    public double dist(PointT p)
    {
        double dx;
        double dy;

        dx = this.xc - p.xc;
        dy = this.yc - p.yc;

        return sqrt(pow(dx,2.0) + pow(dy,2.0));
    }
}
```

```
public boolean equal(PointT p)
{
    return (this.dist(p) <= TOLERANCE);
}
}
```

B TestPointT.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2009
 *
 * Description: Testing PointT Class
 */

import org.junit.*;
import static org.junit.Assert.*;

public class TestPointT
{
    private static double ADMISS_ERR_CONSTRUCTOR = 0;
    private static double ADMISS_ERR_DIST = 1e-20;

    @Test
    public void testConstructorForx()
    {
        assertEquals(23, new PointT(23, 38).xcoord(), ADMISS_ERR_CONSTRUCTOR);
    }

    @Test
    public void testConstructorFory()
    {
        assertEquals(38, new PointT(23, 38).ycoord(), ADMISS_ERR_CONSTRUCTOR);
    }

    @Test
    public void testDist1()
    {
        double x = 2.0;
        double y = 2.0;
        PointT p = new PointT(x, y);
        assertEquals(Math.sqrt(x*x + y*y), p.dist(new PointT(0, 0)), ADMISS_ERR_DIST);
    }

    @Test
    public void testDist2()
    {
        double x = 10.0;
        double y = -567.9;
        PointT p = new PointT(x, y);
        assertEquals(Math.sqrt(x*x + y*y), p.dist(new PointT(0, 0)), ADMISS_ERR_DIST);
    }

    @Test
    public void testDistZero()
    {
        double x = 2.0;
        double y = 2.0;
        PointT p = new PointT(x, y);
        assertEquals(0, p.dist(p), ADMISS_ERR_DIST);
    }

    @Test
    public void testEqual()
    {
        double x = 0.0;
        double y = 0.0;
        PointT p1 = new PointT(x, y);
        PointT p2 = new PointT(x + PointT.TOLERANCE, y + PointT.TOLERANCE);
        assertEquals(true, p1.equal(p1));
    }
}
```

C NodeT.java

```
/**
 * Author: S. Smith
 * Revised: March 2, 2009
 *
 * Description: Node ADT class
 */
public class NodeT extends PointT
{
    public enum nodeTypeT {JUNCTION, EGG, BLOCKAGE, OPENING}

    private nodeTypeT nt;

    public NodeT(double x, double y, nodeTypeT n)
    {
        super(x, y);
        nt = n;
    }

    public nodeTypeT ntype()
    {
        return nt;
    }
}
```

D TestNodeT.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2009
 *
 * Description: Testing NodeT Class
 */

import org.junit.*;
import static org.junit.Assert.*;

public class TestNodeT
{
    @Test
    public void testConstructorForJunction()
    {
        NodeT n = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        assertEquals(NodeT.nodeTypeT.JUNCTION, n.ntype());
    }

    @Test
    public void testConstructorForEgg()
    {
        NodeT n = new NodeT(0.0, 0.0, NodeT.nodeTypeT.EGG);
        assertEquals(NodeT.nodeTypeT.EGG, n.ntype());
    }

    @Test
    public void testConstructorForBlockage()
    {
        NodeT n = new NodeT(0.0, 0.0, NodeT.nodeTypeT.BLOCKAGE);
        assertEquals(NodeT.nodeTypeT.BLOCKAGE, n.ntype());
    }

    @Test
    public void testConstructorForOpening()
    {
        NodeT n = new NodeT(0.0, 0.0, NodeT.nodeTypeT.OPENING);
        assertEquals(NodeT.nodeTypeT.OPENING, n.ntype());
    }
}
```

E EdgeT.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2009
 *
 * Description: Edge ADT class
 */

import static java.lang.Math.*;

public class EdgeT
{
    public static final double TOLERANCE = 1e-5;

    private NodeT n1;
    private NodeT n2;

    public EdgeT(NodeT nod1, NodeT nod2)
    {
        n1 = nod1;
        n2 = nod2;
    }

    public NodeT node1()
    {
        return n1;
    }

    public NodeT node2()
    {
        return n2;
    }

    public double length()
    {
        return n1.dist(n2);
    }

    public boolean is_horizontal()
    {
        return abs(n1.ycoord() - n2.ycoord()) <= TOLERANCE;
    }

    public boolean equal(EdgeT e)
    {
        return (n1.equal(e.node1()) && n2.equal(e.node2())) || (n1.equal(e.node2()) &&
            n2.equal(e.node1())) ;
    }
}
```

F TestEdgeT.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2009
 *
 * Description: Testing EdgeT Class
 */

import org.junit.*;
import static org.junit.Assert.*;
import static java.lang.Math.*;

public class TestEdgeT
{
    private static double ADMISS_ERR_DIST = 1e-20;

    @Test
    public void testConstructorForNode1()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(true, n1.equal(e.node1()));
    }

    @Test
    public void testConstructorForNode2()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(true, n2.equal(e.node2()));
    }

    @Test
    public void testLength()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(sqrt(2), e.length(), ADMISS_ERR_DIST);
    }

    @Test
    public void testHorizontalYes()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(true, e.is_horizontal());
    }

    @Test
    public void testHorizontalNo()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(false, e.is_horizontal());
    }

    @Test
    public void testEqualYes1()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
        EdgeT e2 = new EdgeT(n1, n2);
        assertEquals(true, e1.equal(e2));
    }

    @Test
    public void testEqualYes2()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
```

```

        EdgeT e2 = new EdgeT(n2, n1);
        assertEquals(true, e1.equal(e2));
    }

    @Test
    public void testEqualNo()
    {
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
        EdgeT e2 = new EdgeT(n1, n1);
        assertEquals(false, e1.equal(e2));
    }
}

```


G Map.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2008
 *
 * Description: Map Abstract object
 */

import java.util.ArrayList;

public class Map
{
    private static ArrayList<EdgeT> s;

    public static void init()
    {
        s = new ArrayList<EdgeT>();
    }

    public static void add(EdgeT e)
    {
        if (contains(e))
        {
            throw new ALREADY_IN_MAP("The element being added is already in the map");
        }

        s.add(e);
    }

    public static void del(EdgeT e)
    {
        int i;

        if (!contains(e))
        {
            throw new NOT_IN_MAP("The element being deleted is not already in the map");
        }

        i = 0;
        while (i < s.size())
        {
            if (e.equal(s.get(i)))
            {
                s.remove(i);
            }
            i++;
        }
    }

    public static boolean contains(EdgeT e)
    {
        int i;
        boolean containsElm = false;

        for (i = 0; i < s.size(); i++)
        {
            containsElm = containsElm || e.equal(s.get(i));
        }

        return containsElm;
    }
}
```

H TestMap.java

```
/**
 * Author: S. Smith
 * Revised: March 14, 2008
 *
 * Description: Testing Map Class
 */

import org.junit.*;
import static org.junit.Assert.*;

public class TestMap
{
    @Test
    public void testInit() //checking for correct syntax, absence of exceptions
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        assertEquals(false, Map.contains(e));
    }

    @Test
    public void testAdd1()
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        Map.add(e);
        assertEquals(true, Map.contains(e));
    }

    @Test
    public void testAdd2()
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
        NodeT n3 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n4 = new NodeT(1.0, 15.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e2 = new EdgeT(n3, n4);
        Map.add(e1);
        Map.add(e2);
        assertEquals(true, Map.contains(e1) && Map.contains(e2));
    }

    @Test
    public void testAdd3()
    {
        int MAX = 10;
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);

        Map.init();
        for (int i = 0; i < MAX; i++)
        {
            n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
            n2 = new NodeT(i, i, NodeT.nodeTypeT.JUNCTION);
            e = new EdgeT(n1, n2);
            Map.add(e);
        }
        assertEquals(true, Map.contains(e));
    }

    @Test (expected=ALREADY_IN_MAP.class)
    public void testForExceptionALREADY1()
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e = new EdgeT(n1, n2);
        Map.add(e);
    }
}
```

```

    Map.add(e);
}

@Test (expected=ALREADY_IN_MAP.class)
public void testForExceptionALREADY2()
{
    Map.init();
    NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    EdgeT e1 = new EdgeT(n1, n2);
    Map.add(e1);
    EdgeT e2 = new EdgeT(n2, n1);
    Map.add(e2);
}

@Test
public void testDel1()
{
    Map.init();
    NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    NodeT n2 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
    EdgeT e1 = new EdgeT(n1, n2);
    NodeT n3 = new NodeT(1.0, 1.0, NodeT.nodeTypeT.JUNCTION);
    NodeT n4 = new NodeT(1.0, 15.0, NodeT.nodeTypeT.JUNCTION);
    EdgeT e2 = new EdgeT(n3, n4);
    Map.add(e1);
    Map.add(e2);
    Map.del(e1);
    assertEquals(true, Map.contains(e2) && !Map.contains(e1));
}

@Test
public void testDel2()
{
    int MAX = 10;
    NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    EdgeT e = new EdgeT(n1, n2);

    Map.init();
    for (int i = 0; i < MAX; i++)
    {
        n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        n2 = new NodeT(i, i, NodeT.nodeTypeT.JUNCTION);
        e = new EdgeT(n1, n2);
        Map.add(e);
    }

    n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    n2 = new NodeT(2, 2, NodeT.nodeTypeT.JUNCTION);
    e = new EdgeT(n1, n2);

    Map.del(e);

    assertEquals(true, !Map.contains(e));
}

@Test (expected=NOT_IN_MAP.class)
public void testForExceptionNOT()
{
    int MAX = 10;
    NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    EdgeT e = new EdgeT(n1, n2);

    Map.init();
    for (int i = 0; i < MAX; i++)
    {
        n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        n2 = new NodeT(i, i, NodeT.nodeTypeT.JUNCTION);
        e = new EdgeT(n1, n2);
        Map.add(e);
    }

    n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
    n2 = new NodeT(2, 2, NodeT.nodeTypeT.JUNCTION);
    e = new EdgeT(n1, n2);

    Map.del(e);
    Map.del(e);
}

```

```

    }

    @Test
    public void testContainsYes()
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
        Map.add(e1);
        n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        n2 = new NodeT(23.0, 23.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e2 = new EdgeT(n1, n2);
        Map.add(e2);
        assertEquals(true, Map.contains(e1));
    }

    @Test
    public void testContainsNo()
    {
        Map.init();
        NodeT n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        NodeT n2 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e1 = new EdgeT(n1, n2);
        Map.add(e1);
        n1 = new NodeT(0.0, 0.0, NodeT.nodeTypeT.JUNCTION);
        n2 = new NodeT(23.0, 23.0, NodeT.nodeTypeT.JUNCTION);
        EdgeT e2 = new EdgeT(n1, n2);
        assertEquals(true, !Map.contains(e2));
    }
}

```

I AllTests.java

```
/**
 * Author: S. Smith
 * Revised: March 17, 2009
 *
 * Description: Testing all of the map and related modules
 */

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestPointT.class,
    TestNodeT.class,
    TestEdgeT.class,
    TestMap.class
})

public class AllTests
{
}
```