```java
//********************************************************************************
//
// File:    SpaceNetwork.java
// Package: ---
// Unit:    Class SpaceNetwork
//
//********************************************************************************

import edu.rit.pj2.vbl.DoubleVbl;
import edu.rit.util.Random;


/**
 * Class models a network of space stations placed in random positions in 3D
 * space. The space stations' locations are limited to
 * 1E8 million kilometers X 1E8 million kilometers X 1E8 million kilometers.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 */
public class SpaceNetwork {

    /**
     * maximum dimension value allowed in 3D space
     */
    public static final double MAX_DIM = 1.0E8;

    /**
     * number of space stations
     */
    public final int n;

    // private data members
    private boolean connected;
    private double[][] adj;
    private double[][] shortest;
    private SpaceStation[] stations;

    /**
     * Construct a SpaceNetwork
     *
     * @param prng the pseudorandom number generator to use
     * @param n the number of space stations in this network
     */
    public SpaceNetwork(Random prng, final int n) {
        this.n = n;
        this.adj = new double[n][n];
        this.shortest = new double[n][n];
        this.stations = new SpaceStation[n];
        initStations(prng);
        initAdjacency();
        floydWarshall();
        checkConnectivity();
    }

    /**
     * initialize the coordinates of the <TT>n</TT> stations
     * @param prng the pseudorandom number generator to get random numbers from
```

```
59      */
60     private void initStations(Random prng) {
61         double x, y, z;
62         for(int i = 0; i < n; i++) {
63             x = prng.nextDouble() * MAX_DIM;
64             y = prng.nextDouble() * MAX_DIM;
65             z = prng.nextDouble() * MAX_DIM;
66             stations[i] = new SpaceStation(i, x, y, z);
67         }
68     }
69 
70     /**
71      * initialize the weights of the edges between nodes with the power needed
72      * to transmit from one station to another
73      */
74     private void initAdjacency() {
75         SpaceStation s1, s2;
76         double distance, power;
77         for(int i = 0; i < n; i++) {
78             adj[i][i] = 0; // not needed
79             s1 = get(i);
80             for(int j = i+1; j < n; j++) {
81                 s2 = get(j);
82                 distance = s1.distance(s2);
83                 if(distance > SpaceStation.MAX_DISTANCE) {
84                     power = Double.POSITIVE_INFINITY;
85                 } else {
86                     power = s1.powerNeeded(s2);
87                 }
88                 adj[i][j] = power;
89                 adj[j][i] = power;
90             }
91         }
92     }
93 
94     /**
95      * Run Floyd-Warshall on the space network to determine all-pairs shortest
96      * paths. This will tell us the least amount of power a station needs to
97      * transmit to any other station in the network by forwarding the message
98      * along the shortest path to that station.
99      */
100    private void floydWarshall() {
101        System.arraycopy(adj, 0, shortest, 0, n);
102        double s_i_j, s_i_k, s_k_j;
103        for(int k = 0; k < n; k++) {
104            for(int i = 0; i < n; i++) {
105                for(int j = 0; j < n; j++) {
106                    s_i_j = shortest[i][j];
107                    s_i_k = shortest[i][k];
108                    s_k_j = shortest[k][j];
109                    if(s_i_j > s_i_k + s_k_j) {
110                        shortest[i][j] = s_i_k + s_k_j;
111                    }
112                }
113            }
114        }
115    }
116
```

```
117     /**
118      * Check if the network is connected
119      */
120     private void checkConnectivity() {
121         boolean connected = true;
122         double temp;
123         for(int i = 0; i < n && connected; i++) {
124             for(int j = 0; j < n && connected; j++) {
125                 temp = shortest[i][j];
126                 connected = !Double.isInfinite(temp);
127             }
128         }
129         this.connected = connected;
130     }
131 
132     /**
133      * get whether the network is connected or not
134      * @return true if the network is fully-connected
135      */
136     public boolean isConnected() {
137         return connected;
138     }
139 
140     /**
141      * get a space station
142      * @param n the unique identifier of the space station
143      * @return the space station with identifier = n
144      */
145     private SpaceStation get(int n) {
146         return stations[n];
147     }
148 
149     /**
150      * Accumulate the powers needed to transmit messages into a thread-local
151      * copy of a DoubleVbl.Mean. This is what averages the powers across
152      * multiple networks
153      * @param power
154      */
155     public void accumulatePower(DoubleVbl.Mean power) {
156         double temp;
157         for(int i = 0; i < n; i++) {
158             for(int j = i + 1; j < n; j++) {
159                 temp = shortest[i][j];
160                 if(!Double.isInfinite(temp) && temp != 0)
161                     power.accumulate(temp);
162             }
163         }
164     }
165 }
166 
```