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

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