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

import java.io.File;
import java.io.IOException;
import java.text.DecimalFormat;
import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Plot;
import edu.rit.numeric.plot.Strokes;
import edu.rit.util.AList;

/**
 * Class PlotHandler is the delegate for dealing with visualizing the data
 * generated by the "number crunching" program, SimulationStation.
 * Its purpose is to be instantiated in SimulationStation with the data to plot,
 * where the write() method should then be called.
 *
 * Running this program and specifying in the command line arguments the plot
 * files previously generated will open a graphical representation of these
 * plots for each file.
 *
 * @author Jimi Ford
 * @version 4-4-2015
 *
 */
public class PlotHandler {

    // private data members
    private final String averagePowerFile;
    private final String probabilityFile;
    private final AList<SimulationResult> results;

    /**
     * Construct a new plot handler that plots average distances for a fixed
     * vertex count v, while varying the edge probability p
     *
     * @param   plotFilePrefix prefix to be used in the name of
     *          the plot file
     * @param   results collection of results of the finished set of
     *          simulations.
     */
    public PlotHandler(String plotFilePrefix,
            AList<SimulationResult> results) {
        averagePowerFile = plotFilePrefix + "-average-power.dwg";
        probabilityFile = plotFilePrefix + "-probability-connected.dwg";
        this.results = results;
    }

    /**
     * Save the plot information into a file to visualize by running
     * the main method of this class
     *
     * @throws IOException if it can't write to the file specified
     */
    public void write() throws IOException {
        ListXYSeries averagePowerSeries = new ListXYSeries();
        ListXYSeries probabilitySeries = new ListXYSeries();
        SimulationResult result = null;
        for(int i = 0; i < this.results.size(); i++) {
            result = results.get(i);
            if(!Double.isNaN(result.averagePower))
                averagePowerSeries.add(result.v, result.averagePower);
            if(!Double.isNaN(result.percentConnected))
                probabilitySeries.add(result.v, result.percentConnected);
        }

        Plot powerPlot = new Plot()
            .plotTitle ("Average Power vs. Number of Nodes")
            .xAxisTitle ("Number of Nodes <I>V</I>")
            .xAxisTickFormat(new DecimalFormat("0"))
            .yAxisTitle ("Average Power Needed")
            .yAxisTickFormat (new DecimalFormat ("0E0"))
            .seriesDots(null)
            .seriesStroke (Strokes.solid(2))
            .xySeries (averagePowerSeries);
        Plot.write(powerPlot, new File(averagePowerFile));
        Plot probabilityPlot = new Plot()
        .plotTitle ("Percent Connected vs. Number of Nodes")
        .xAxisTitle ("Number of Nodes <I>V</I>")
        .xAxisTickFormat(new DecimalFormat("0"))
        .yAxisTitle ("Percent Connected")
        .yAxisTickFormat (new DecimalFormat ("0.0"))
        .seriesDots(null)
        .seriesStroke (Strokes.solid(2))
        .xySeries (probabilitySeries);
        Plot.write(probabilityPlot, new File(probabilityFile));
    }

    /**
     * Open a GUI for each plot in order to visualize the results of a
     * previously run set of simulations.
     *
     * @param args each plot file generated that you wish to visualize
     */
    public static void main(String args[]) {
        if(args.length < 1) {
            System.err.println("Must specify at least 1 plot file.");
            usage();
        }

        for(int i = 0; i < args.length; i++) {
            try {
                Plot plot = Plot.read(args[i]);
                plot.getFrame().setVisible(true);
            } catch (ClassNotFoundException e) {
                System.err.println("Could not deserialize " + args[i]);
            } catch (IOException e) {
                System.err.println("Could not open " + args[i]);
            } catch (IllegalArgumentException e) {
                System.err.println("Error in file " + args[i]);
            }
        }
    }

    /**
     * Print the usage message for this program and gracefully exit.
     */
    private static void usage() {
        System.err.println("usage: java PlotHandler <plot-file-1> "+
                "(<plot-file-2> <plot-file-3>... etc.)");
        System.exit(1);
    }
}

```

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

/**
 * Class SimulationResult is designed to be just a data container for recording
 * the results of running <I>n</I> simulations given a number of space stations
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class SimulationResult {

    /**
     * the percentage of connected networks generated
     */
    public final double percentConnected;

    /**
     * the average power needed by the stations to transmit to any other station
     * in each network
     */
    public final double averagePower;

    /**
     * the number of vertices (i.e. nodes or space stations)
     */
    public final int v;

    /**
     * the number of trials that were run to generate this result
     */
    public final int trials;

    /**
     * Construct a SimulationResult
     * @param v number of space stations
     * @param trials number of trials that were run
     * @param connectedCount the number of connected networks produced
     * @param averagePower the average power needed by each space station to
     *      transmit messages to any other space station in the network
     */
    public SimulationResult(int v, int trials, int connectedCount,
            double averagePower) {
        this.v = v;
        this.trials = trials;
        this.percentConnected = connectedCount / (double) trials;
        this.averagePower = averagePower;
    }
}

```

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

import java.io.IOException;
import edu.rit.pj2.Task;
import edu.rit.util.AList;

/**
 * Class runs a number of trials simulating a network of space stations
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 */
public class SimulationStation extends Task {

    // java pj2 SimulationStation
    // [0] = <lower_bound_stations>
    // [1] = <upper_bound_stations>
    // [2] = <increment_stations>
    // [3] = <seed>
    // [4] = <file_prefix>

    private static final int
        LOWER_INDEX = 0,
        UPPER_INDEX = 1,
        INCREMENT_INDEX = 2,
        TRIALS_INDEX = 3,
        SEED_INDEX = 4,
        FILE_INDEX = 5;

    /**
     * main method
     * @param args command line arguments
     */
    public void main(String[] args) {
        if(args.length != 6) {
            usage();
        }
        int lowerBound = 0, upperBound = 0, increment = 0, trials = 0;
        long seed = 0;
        String filePrefix = null;
        try {
            lowerBound = Integer.parseInt(args[LOWER_INDEX]);
            upperBound = Integer.parseInt(args[UPPER_INDEX]);
            increment = Integer.parseInt(args[INCREMENT_INDEX]);
            trials = Integer.parseInt(args[TRIALS_INDEX]);
            seed = Long.parseLong(args[SEED_INDEX]);
            filePrefix = args[FILE_INDEX];
        } catch (NumberFormatException e) {
            error("Detected non-numeric input where expected numeric value");
        }
        AList<SimulationResult> results = new AList<SimulationResult>();
        for(int vertices = lowerBound; vertices <= upperBound;
                vertices += increment) {
            results.addLast(
                    new Simulator(this, vertices, trials, seed).simulate());
        }
        try {
            new PlotHandler(filePrefix, results).write();
            new TableHandler(filePrefix, results).write();
        } catch (IOException e) {
            error("Error writing results file(s) using prefix: " + filePrefix);
        }

    }

    /**
     * print usage statement and gracefully exit
     */
    private static void usage() {
        System.err.println("java pj2 SimulationStation "
                + "<lower_bound_stations> "
                + "<upper_bound_stations> "
                + "<increment_stations> "
                + "<num_trials> "
                + "<seed> "
                + "<file_prefix>");
        System.exit(1);
    }

    /**
     * print an error message and call the usage() method
     * @param msg the error message to print
     */
    private static void error(String msg) {
        System.err.println(msg);
        usage();
    }
}

```

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

import edu.rit.pj2.Loop;
import edu.rit.pj2.Task;
import edu.rit.pj2.vbl.DoubleVbl;
import edu.rit.pj2.vbl.IntVbl;
import edu.rit.util.Random;

/**
 * Class is responsible for the majority of the runtime of the program. It
 * generates the given number of networks in parallel by utilizing Prof. Alan
 * Kaminsky's PJ2 library.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class Simulator {

    private Task ref;
    private int v;
    private int trials;
    private long seed;
    private IntVbl.Sum countConnected;
    private DoubleVbl.Mean averagePower;

    /**
     * Construct a Simulator
     * @param ref the reference to the main task - necessary for utilizing the
     *      class's parallelFor method
     * @param v number of space stations (or nodes in the graph)
     * @param trials the number of random networks to generate
     * @param seed seed value for the PRNG used in instantiated classes
     */
    public Simulator(Task ref, int v, int trials, long seed) {
        this.ref = ref;
        this.v = v;
        this.trials = trials;
        this.seed = seed;
        countConnected = new IntVbl.Sum();
        averagePower = new DoubleVbl.Mean();
    }

    /**
     * Run all <TT>trials<TT>
     * @return a SimulationResult containing the findings of the given number of
     *          simulations
     */
    public SimulationResult simulate() {
        ref.parallelFor(0, trials - 1).exec(new Loop() {

            Random prng;
            DoubleVbl.Mean thrAverage;
            IntVbl.Sum thrCount;

            // (Non-javadoc)
            public void start() {
                prng = new Random(seed + rank());
                thrAverage = threadLocal(averagePower);
                thrCount = threadLocal(countConnected);
            }

            // (Non-javadoc)
            public void run(int i) throws Exception {
                SpaceNetwork sn = new SpaceNetwork(prng, v);
                if(sn.isConnected()) {
                    thrCount.item++;
                }
                sn.accumulatePower(thrAverage);
            }
        });
        return new SimulationResult(
                v,
                trials,
                countConnected.intValue(),
                averagePower.doubleValue());
    }
}
```

```
//******************************************************************************
//
// 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
```

```java
     */
    private void initStations(Random prng) {
        double x, y, z;
        for(int i = 0; i < n; i++) {
            x = prng.nextDouble() * MAX_DIM;
            y = prng.nextDouble() * MAX_DIM;
            z = prng.nextDouble() * MAX_DIM;
            stations[i] = new SpaceStation(i, x, y, z);
        }
    }

    /**
     * initialize the weights of the edges between nodes with the power needed
     * to transmit from one station to another
     */
    private void initAdjacency() {
        SpaceStation s1, s2;
        double distance, power;
        for(int i = 0; i < n; i++) {
            adj[i][i] = 0; // not needed
            s1 = get(i);
            for(int j = i+1; j < n; j++) {
                s2 = get(j);
                distance = s1.distance(s2);
                if(distance > SpaceStation.MAX_DISTANCE) {
                    power = Double.POSITIVE_INFINITY;
                } else {
                    power = s1.powerNeeded(s2);
                }
                adj[i][j] = power;
                adj[j][i] = power;
            }
        }
    }

    /**
     * Run Floyd-Warshall on the space network to determine all-pairs shortest
     * paths. This will tell us the least amount of power a station needs to
     * transmit to any other station in the network by forwarding the message
     * along the shortest path to that station.
     */
    private void floydWarshall() {
        System.arraycopy(adj, 0, shortest, 0, n);
        double s_i_j, s_i_k, s_k_j;
        for(int k = 0; k < n; k++) {
            for(int i = 0; i < n; i++) {
                for(int j = 0; j < n; j++) {
                    s_i_j = shortest[i][j];
                    s_i_k = shortest[i][k];
                    s_k_j = shortest[k][j];
                    if(s_i_j > s_i_k + s_k_j) {
                        shortest[i][j] = s_i_k + s_k_j;
                    }
                }
            }
        }
    }

    /**
     * Check if the network is connected
     */
    private void checkConnectivity() {
        boolean connected = true;
        double temp;
        for(int i = 0; i < n && connected; i++) {
            for(int j = 0; j < n && connected; j++) {
                temp = shortest[i][j];
                connected = !Double.isInfinite(temp);
            }
        }
        this.connected = connected;
    }

    /**
     * get whether the network is connected or not
     * @return true if the network is fully-connected
     */
    public boolean isConnected() {
        return connected;
    }

    /**
     * get a space station
     * @param n the unique identifier of the space station
     * @return the space station with identifier = n
     */
    private SpaceStation get(int n) {
        return stations[n];
    }

    /**
     * Accumulate the powers needed to transmit messages into a thread-local
     * copy of a DoubleVbl.Mean. This is what averages the powers across
     * multiple networks
     * @param power
     */
    public void accumulatePower(DoubleVbl.Mean power) {
        double temp;
        for(int i = 0; i < n; i++) {
            for(int j = i + 1; j < n; j++) {
                temp = shortest[i][j];
                if(!Double.isInfinite(temp) && temp != 0)
                    power.accumulate(temp);
            }
        }
    }
}
```

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

/**
 * Class models a space station floating around in 3D space. This class contains
 * the math needed to calculate distances to other stations and the power needed
 * to transmit to them.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-2-2015
 *
 */
public class SpaceStation {

    /**
     * maximum distance a space station can transmit
     */
    public static final double MAX_DISTANCE = 40.0E6;

    /**
     * the station's x-coordinate
     */
    public final double x;

    /**
     * the station's y-coordinate
     */
    public final double y;

    /**
     * the station's z-coordinate
     */
    public final double z;

    /**
     * the station's unique identifier
     */
    public final int id;

    /**
     * Construct a new space station. It is assumed that all the parameters are
     * less than or equal to MAX_DIM.
     * @param x x-coordinate in 3D space
     * @param y y-coordinate in 3D space
     * @param z z-coordinate in 3D space
     */
    public SpaceStation(int id, double x, double y, double z) {
        this.id = id;
        this.x = x;
        this.y = y;
        this.z = z;
    }

    /**
     * compute the straight line distance to another space station
     * @param other the other space station to compute the distance to
     * @return the Euclidean distance to this space station
     */
    public double distance(SpaceStation other) {
        return Math.sqrt(powerNeeded(other));
    }

    /**
     * compute the power needed to transmit to another space station
     * @param other the other space station to calculate the power needed
     * @return the power needed to transmit to the other space station
     */
    public double powerNeeded(SpaceStation other) {
        return  ((other.x - x)*(other.x - x)) +
                ((other.y - y)*(other.y - y)) +
                ((other.z - z)*(other.z - z));
    }
}

```

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

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import edu.rit.util.AList;

/**
 * Class handles writing the CSV file containing the results of the simulations.
 *
 * @author Jimi Ford (jhf3617)
 * @version 4-4-2015
 */
public class TableHandler {

    // private data members
    private final String file;
    private final AList<SimulationResult> results;

    /**
     * Construct a Tablehandler
     * @param prefix the prefix of the file name
     * @param results the collective results of the simulations
     */
    public TableHandler(String prefix, AList<SimulationResult> results) {
        this.file = prefix + "-table.csv";
        this.results = results;
    }

    /**
     * write a CSV file containing the results of the simulations
     */
    public void write() {
        SimulationResult temp;
        StringBuilder builder = new StringBuilder();
        builder.append("num_stations, average_power, percent_connected,"+'\n');
        for(int i = 0; i < results.size(); i++) {
            temp = results.get(i);
            builder.append(temp.v + ", " + temp.averagePower + ", " +
                    temp.percentConnected + ", " +'\n');
        }
        PrintWriter tableWriter = null;
        try {
            tableWriter = new PrintWriter(file);
            tableWriter.print(builder.toString());
        } catch (FileNotFoundException e) {
            System.err.println("Error writing table data to file \"" +
                    file+"\"");
        } finally {
            if(tableWriter != null) tableWriter.close();
        }
    }
}
```