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

/**
 * Class SimulationResultcollection keeps track of the average distance measured
 * for each pair of edge probability values and number of vertices. It also
 * contains the necessary computation to account for using integers as
 * probabilities, treating them as floating point values ranging from 0 to 1.
 *
 * @author Jimi Ford
 * @version 2-15-2015
 */
public class SimulationResultCollection {

    // private data members
    private double[][] averages;
    private int rows, cols;

    /**
     * The lower bound on number of vertices
     */
    public final int vMin;

    /**
     * The upper bound on number of vertices
     */
    public final int vMax;

    /**
     * The amount to increment the number of vertices by in each set of trials
     */
    public final int vInc;

    /**
     * The scaled lower bound on edge probability
     */
    public final int pMin;

    /**
     * The scaled upper bound on edge probability
     */
    public final int pMax;

    /**
     * The amount to increment the edge probability by in each set of trials
     */
    public final int pInc;

    /**
     * The number of decimal places necessary to convert the edge probability
     * into an integer. This is in order to combat floating point arithmetic.
     * One can't just loop from 0 to 1 incrementing by .1 because compounding
     * error is accumulated on each increment. Integers play nicely when
```

```
59      * incremented.
60      */
61     public final int pExp;
62
63     /**
64      * Construct a simulation result collection. The parameter values
65      * should reflect the values passed into the program through the
66      * command line arguments.
67      *
68      * @param vMin The lower bound on number of vertices
69      * @param vMax The upper bound on number of vertices
70      * @param vInc The amount to increment the number of vertices by in
71      *             each set of trials
72      * @param pMin The scaled lower bound on edge probability
73      * @param pMax The scaled upper bound on edge probability
74      * @param pInc The scaled amount to increment the edge probability by
75      *             in each set of trials
76      * @param pExp The number of decimal places used to convert the edge
77      *             probability into an integer
78      */
79     public SimulationResultCollection (int vMin, int vMax, int vInc,
80             int pMin, int pMax, int pInc, int pExp) {
81         this.vMin = vMin;
82         this.vMax = vMax;
83         this.vInc = vInc;
84         this.pMin = pMin;
85         this.pMax = pMax;
86         this.pInc = pInc;
87         this.pExp = pExp;
88         this.rows = (vMax - vMin + vInc) / vInc;
89         this.cols = (pMax - pMin + pInc) / pInc;
90         this.averages = new double[rows][cols];
91     }
92
93     /**
94      * Add a simulation result to the data matrix.
95      *
96      * @param result the simulation result to record
97      */
98     public void add(SimulationResult result) {
99         int p = p(result.p);
100        int col = col(p);
101        int row = row(result.v);
102        averages[row][col] = result.averageDistance;
103
104    }
105
106    /**
107     * Get the average distance recorded for a given vertex count
108     * and a scaled edge probability
109     *
110     * @param v the vertex count
111     * @param p the scaled edge probability
112     * @return the average distance recorded for this pair
113     */
114    public double get(int v, int p) {
115        int row = row(v);
116        int col = col(p);
```

```
117         return averages[row][col];
118     }
119 
120     /**
121      * Get an array of averages for varying edge probabilities and
122      * a given vertex count.
123      *
124      * @param v the vertex count of interest
125      * @return the array of averages for this vertex count
126      */
127     public double[] getAveragesForV(int v) {
128         return averages[row(v)].clone();
129     }
130 
131     /**
132      * Convert a vertex value into its associated row value in the
133      * data matrix.
134      *
135      * @param v the vertex count (or number of vertices) to convert
136      * @return the associated row value in the data matrix
137      */
138     private int row(int v) {
139         return (v - vMin) / vInc;
140     }
141 
142     /**
143      * Convert an edge probability into a scaled integer in order
144      * to get rid of floating point arithmetic errors.
145      *
146      * @param p the edge probability to convert
147      * @return the scaled integer ranging from pMin to pMax
148      */
149     private int p(double p) {
150         return (int) (Math.round(p * pExp));
151     }
152 
153     /**
154      * Convert a scaled edge probability into the associated
155      * column value in the data matrix.
156      *
157      * @param p the scaled edge probability to convert
158      * @return the associated column value in the data matrix
159      */
160     private int col(int p) {
161         return (p - pMin) / pInc;
162     }
163 
164 }
165 
```