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

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import edu.rit.pj2.Task;

/**
 * Class MonteCarlo takes in a seed value for a random number
 * generator, the upper and lower boundaries for the number of vertices in
 * each graph as well as a number to increment by, the upper and lower
 * boundaries for the edge probability as well as a number to increment by,
 * the number of random graphs to generate for each combination of V (vertices)
 * and p (edge probability), and finally, a prefix for naming each plot
 * generated by this program. After checking for valid input, this program
 * loops through each combination of vertices and edge probabilities, running
 * the specified number of simulations on each combination. Each random graph
 * (or simulation) is generated by looking at every possible pair of vertices,
 * generating a random floating point between 0 and 1, and marking these
 * vertices with an edge connecting them if the random value is less than or
 * equal to the specified edge probability (for that unique graph). In each
 * simulation, the distance values of each graph are calculated with a breadth
 * first search from vertex A to vertex B using the depth of the search as the
 * distance from A to B.
 *
 * @author  Jimi Ford
 * @version 2-15-2015
 */
public class MonteCarlo extends Task {

    // Private constants
    private static final String[] arguments = {
        "<seed>",
        "<min_v>",
        "<max_v>",
        "<v_grain>",
        "<min_p>",
        "<max_p>",
        "<p_grain>",
        "<num_simulations>",
        "<optional plotfile prefix>"
    };

    private static final int
        SEED = 0,
        MIN_VERTICES = 1,
        MAX_VERTICES = 2,
        VERTEX_GRANULARITY = 3,
        MIN_P = 4,
        MAX_P = 5,
        P_GRANULARITY = 6,
        NUMBER_OF_SIMULATIONS = 7,
```

```
59          PLOT_FILE_PREFIX = 8;
60
61      /**
62       * MonteCarlo's main method to be invoked by Prof. Alan Kaminsky's
63       * Parallel Java 2 library.
64       *
65       * @param args command line arguments
66       *
67       * <P>
68       * usage: java pj2 MonteCarlo &lt;seed&gt; &lt;min_v&gt; &lt;max_v&gt;
69       * &lt;v_grain&gt; &lt;min_p&gt; &lt;max_p&gt; &lt;p_grain&gt;
70       * &lt;num_simulations&gt; &lt;optional plotfile prefix&gt;
71       * <P>
72       */
73      public void main(String[] args) {
74          if(args.length != 8 && args.length != 9) {
75              usage();
76          }
77
78          long seed = 0;
79          int minVertices = 0, maxVertices = 0, vertexGranularity = 0,
80                  numSimulations = 0;
81          double pGrain = 0, minP = 0, maxP = 0;
82
83          try {
84              seed = Long.parseLong(args[SEED]);
85          } catch (NumberFormatException e) {
86              displayError(
87                      String.format("Argument %1s must be numeric and between %2d "+
88                      "and %3d inclusive.\n", arguments[SEED],
89                  Long.MIN_VALUE, Long.MAX_VALUE));
90          }
91
92          try {
93              minVertices = Integer.parseInt(args[MIN_VERTICES]);
94              if(minVertices < 1) throw new NumberFormatException();
95          } catch (NumberFormatException e) {
96              displayError(
97                  String.format("Argument %1s must be numeric and between 1 "+
98                          "and %2d inclusive.\n", arguments[MIN_VERTICES],
99                          Integer.MAX_VALUE));
100         }
101
102         try {
103             maxVertices = Integer.parseInt(args[MAX_VERTICES]);
104             if(maxVertices < minVertices)
105                 displayError(String.format(
106                     "Argument %1s must be greater than or equal to %2s.\n",
107                     arguments[MAX_VERTICES], arguments[MIN_VERTICES]));
108         } catch (NumberFormatException e) {
109             displayError(String.format(
110                 "Argument %1s must be numeric and between 1 and %2d inclusive.\n",
111                     arguments[MAX_VERTICES], Integer.MAX_VALUE));
112         }
113
114         try {
115             vertexGranularity = Integer.parseInt(args[VERTEX_GRANULARITY]);
116             if(vertexGranularity < 1) throw new NumberFormatException();
```

```
117         } catch (NumberFormatException e) {
118             displayError(String.format(
119                 "Argument %1s must be numeric and between 1 and %2d inclusive.\n",
120                     arguments[VERTEX_GRANULARITY], Integer.MAX_VALUE));
121         }
122 
123         try {
124             minP = Double.parseDouble(args[MIN_P]);
125             if(minP < 0 || minP > 1) throw new NumberFormatException();
126         } catch (NumberFormatException e) {
127             displayError(String.format(
128                     "Argument %1s must be numeric and between "+
129                     "0 inclusive and 1 inclusive.\n",
130                     arguments[MIN_P]));
131         }
132 
133         try {
134             maxP = Double.parseDouble(args[MAX_P]);
135             if(maxP < minP)
136                 displayError(String.format(
137                         "Argument %1s must be greater than or equal to %2s.\n",
138                         arguments[MAX_P], arguments[MIN_P]));
139             if(maxP > 1) throw new NumberFormatException();
140         } catch (NumberFormatException e) {
141             displayError(String.format(
142                 "Argument %1s must be numeric and between "+
143                 "0 inclusive and 1 inclusive.\n",
144                     arguments[MAX_P]));
145         }
146 
147         try {
148             pGrain = Double.parseDouble(args[P_GRANULARITY]);
149             if(pGrain <= 0 || pGrain > 1)
150                 throw new NumberFormatException();
151         } catch (NumberFormatException e) {
152             displayError(String.format(
153                 "Argument %1s must be numeric and between "+
154                 "0 exclusive and 1 inclusive.\n",
155                     arguments[P_GRANULARITY]));
156         }
157 
158         try {
159             numSimulations = Integer.parseInt(args[NUMBER_OF_SIMULATIONS]);
160             if(numSimulations < 1) throw new NumberFormatException();
161         } catch (NumberFormatException e) {
162             displayError(String.format(
163                 "Argument %1s must be numeric and between 1 and %2d inclusive.\n",
164                     arguments[NUMBER_OF_SIMULATIONS], Integer.MAX_VALUE));
165         }
166 
167         // store file prefix
168         final String plotFilePrefix = args.length == 9 ?
169                 args[PLOT_FILE_PREFIX] : "plot";
170 
171         String pMinStr = Double.toString(minP);
172         String pMaxStr = Double.toString(maxP);
173         String pGrainStr = Double.toString(pGrain);
174         final int sigFig =
```

```
175                 Math.max(Math.max(
176                         pGrainStr.length() - pGrainStr.indexOf('.') - 1,
177                         pMaxStr.length() - pMaxStr.indexOf('.') - 1),
178                         pMinStr.length() - pMinStr.indexOf('.') - 1);
179         int exp = 1;
180         for(int i = 0; i < sigFig; i++) {
181             exp *= 10;
182         }
183         final int pMax = (int) (Math.round(maxP * exp));
184         final int pInc = (int) (Math.round(pGrain * exp));
185         // if 0 is the lower bound, set pMin to the next "step" of edge probability
186         // which is pInc
187         final int pMin = ((int) (Math.round(minP * exp))) == 0 ?
188                 pInc : ((int) (Math.round(minP * exp)));
189         pGrainStr = null;
190 
191 
192 
193         SimulationResultCollection results = new SimulationResultCollection(
194                 minVertices, maxVertices, vertexGranularity, pMin, pMax, pInc, exp);
195 
196         // loop through number of vertices
197         for(int vCount = minVertices; vCount <= maxVertices;
198                 vCount += vertexGranularity) {
199             // loop through edgeProbability
200             for(int p = pMin; p <= pMax; p += pInc) {
201                 double prob = p / (double) exp;
202                 // loop through each simulation
203                 results.add(new Simulation(this, seed, vCount, prob,
204                         numSimulations).simulate());
205             }
206             try {
207                 new PlotHandler(plotFilePrefix, results, vCount).write();
208             } catch (IOException e) {
209                 System.err.println("Error writing file for v="+vCount);
210             }
211         }
212 
213         StringBuilder builder = new StringBuilder();
214         for(int p = 0; p<= pMax; p+= pInc) {
215             builder.append(", " + (p / ((double) exp)));
216         }
217         builder.append('\n');
218         for(int v = minVertices; v<= maxVertices; v+= vertexGranularity) {
219             builder.append(v + ", ");
220             for(int p = pMin; p <= pMax; p+=pInc) {
221                 builder.append(results.get(v,p)+", ");
222             }
223             builder.append('\n');
224         }
225         PrintWriter tableWriter = null;
226         final String tableSuffix = "-table.csv";
227         try {
228             tableWriter = new PrintWriter(plotFilePrefix + tableSuffix);
229             tableWriter.print(builder.toString());
230         } catch (FileNotFoundException e) {
231             System.err.println("Error writing table data to file \""+
232                 plotFilePrefix + tableSuffix +"\"");
```

```
233         } finally {
234             if(tableWriter != null) tableWriter.close();
235         }
236         System.out.println("Finished simulations! run \"java PlotHandler\" "+
237         "followed by any number of .dwg files (that were previously generated) "+
238         "to visualize the results.");
239     } // main
240 
241 
242     /**
243      * Display the proper usage of this program and exit.
244      */
245     private static void usage() {
246         System.err.printf ("Usage: java pj2 MonteCarlo "+
247                 "%1s %2s %3s %4s %5s %6s %7s %8s %9s\n",
248                 arguments[SEED],
249                 arguments[MIN_VERTICES],
250                 arguments[MAX_VERTICES],
251                 arguments[VERTEX_GRANULARITY],
252                 arguments[MIN_P],
253                 arguments[MAX_P],
254                 arguments[P_GRANULARITY],
255                 arguments[NUMBER_OF_SIMULATIONS],
256                 arguments[PLOT_FILE_PREFIX]);
257         System.exit(1);
258     }
259 
260     /**
261      * Print an error message to System.err and gracefully exit
262      * @param msg the error message to display
263      */
264     private static void displayError(String msg) {
265         System.err.println(msg);
266         usage();
267     }
268 }
269 
```