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

import java.io.IOException;

/**
 * Chirp runs a simulation of crickets chirping at night. The phenomenon we are
 * interested in studying is that some types of networks synchronize in how they
 * chirp. Based on the command line parameters, chirp tests the type of network
 * and determines what time the crickets syncrhonize.
 *
 * @author Jimi Ford (jhf3617)
 * @version 3-31-2015
 */
public class Chirp {

    private static final int GRAPH_TYPE_INDEX = 0,
                             NUM_VERTICES_INDEX = 1,
                             NUM_TICKS_INDEX = 2,
                             OUTPUT_IMAGE_INDEX = 3,
                             SEED_INDEX = 4,
                             K_INDEX = 4,
                             DE_INDEX = 4,
                             DE_SEED_INDEX = 5,
                             EDGE_PROBABILITY_INDEX = 5,
                             K_SEED_INDEX = 5,
                             REWIRE_PROBABILITY_INDEX = 6;

    /**
     * main method
     * @param args command line arguments
     */
    public static void main(String[] args) {
        if(args.length != 4 && args.length != 5 &&
                args.length != 6 && args.length != 7) usage();
        int crickets = 0, ticks = 0, k = 0, dE = 0;
        long seed = 0;
        double prob = 0;
        char mode;
        String outputImage = args[OUTPUT_IMAGE_INDEX];

        try {
            crickets = Integer.parseInt(args[NUM_VERTICES_INDEX]);
        } catch (NumberFormatException e) {
            error("<num vertices> must be a number");
        }
        try {
            ticks = Integer.parseInt(args[NUM_TICKS_INDEX]) + 1;
        } catch (NumberFormatException e) {
            error("<num ticks> must be numeric");
        }
        mode = args[GRAPH_TYPE_INDEX].toLowerCase().charAt(0);
        if(!(mode == 'c' || mode == 'r' || mode == 'k' ||
```

```
60              mode == 's' || mode == 'f')) {
61          error("<graph type> must be either 'c' for cycle, "
62                  + "'r' for random, "
63                  + "'k' for k-regular, "
64                  + "'s' for small-world, "
65                  + "'f' for scale-free");
66      }
67      UndirectedGraph g = null;
68      CricketObserver o = new CricketObserver(crickets, ticks);
69      switch(mode) {
70      case 'r': // RANDOM GRAPH
71          try {
72              seed = Long.parseLong(args[SEED_INDEX]);
73              prob = Double.parseDouble(args[EDGE_PROBABILITY_INDEX]);
74              g = UndirectedGraph.randomGraph(
75                      new Random(seed), crickets, prob, o);
76          } catch(NumberFormatException e) {
77              error("<seed> and <edge probability> must be numeric");
78          } catch(IndexOutOfBoundsException e) {
79              error("<seed> and <edge probability> must be included with "
80                      + "random graph mode");
81          }
82          break;
83      case 'c': // CYCLE GRAPH
84          g = UndirectedGraph.cycleGraph(crickets, o);
85          break;
86      case 'k': // K-REGULAR GRAPH
87          try {
88              k = Integer.parseInt(args[K_INDEX]);
89              g = UndirectedGraph.kregularGraph(crickets, k, o);
90          } catch (NumberFormatException e) {
91              error("<k> must be an integer");
92          } catch (IllegalArgumentException e) {
93              error("<k> must be < the number of crickets");
94          }
95          break;
96      case 's': // SMALL WORLD GRAPH
97          try {
98              k = Integer.parseInt(args[K_INDEX]);
99              prob = Double.parseDouble(args[REWIRE_PROBABILITY_INDEX]);
100             seed = Long.parseLong(args[K_SEED_INDEX]);
101             g = UndirectedGraph.smallWorldGraph(
102                     new Random(seed), crickets, k, prob, o);
103         } catch (NumberFormatException e) {
104             error("<k> must be an integer < V, <rewire probability> "
105                     + "must be a number "
106                     + "between 0 and 1, and <seed> must be numeric");
107         } catch (IllegalArgumentException e) {
108             error("<k> must be < the number of crickets");
109         }
110         break;
111     case 'f': // SCALE-FREE GRAPH
112         try {
113             dE = Integer.parseInt(args[DE_INDEX]);
114             seed = Long.parseLong(args[DE_SEED_INDEX]);
115             g = UndirectedGraph.scaleFreeGraph(
116                     new Random(seed), crickets, dE, o);
117         } catch (NumberFormatException e) {
```

```
118                 error("<dE> and <seed> must be numeric");
119             } catch (IndexOutOfBoundsException e) {
120                 error("<dE> and <seed> must be supplied");
121             }
122         }
123 
124         g.vertices.get(0).forceChirp();
125         Ticker.tick(g, ticks);
126 
127 
128 
129         try {
130             ImageHandler.handle(o, outputImage);
131         } catch (IOException e) {
132             error("Problem writing image");
133         }
134         int sync = o.sync();
135         String description;
136         switch(mode) {
137         case 'c': // CYCLE GRAPH
138             description = "Cycle V = " + crickets +":";
139             handleOutput(description,sync);
140             break;
141         case 'r': // RANDOM GRAPH
142             description = "Random V = " + crickets +", p = " + prob + ":";
143             handleOutput(description,sync);
144             break;
145         case 'k': // K-REGULAR GRAPH
146             description = "K-regular V = " + crickets +", k = " + k + ":";
147             handleOutput(description,sync);
148             break;
149         case 's': // SMALL-WORLD GRAPH
150             description = "Small-world V = " + crickets + ", k = " + k +
151                 ", p = " + prob + ":";
152             handleOutput(description,sync);
153             break;
154         case 'f': // SCALE-FREE GRAPH
155             description = "Scale-free V = " + crickets +", dE = " + dE + ":";
156             handleOutput(description,sync);
157             break;
158         }
159 
160     }
161 
162     /**
163      * handle printing the results of the simulation
164      * @param description the description of what kind of graph is being printed
165      * @param sync time at which the network synchronized
166      *      (-1 for not synchronized)
167      */
168     private static void handleOutput(String description, int sync) {
169         System.out.print(description);
170         if(sync >= 0) {
171             System.out.println("\t"+" synchronized at t="+sync+".");
172         } else {
173             System.out.println("\t "+(char)27+"[31m"+  "did not synchronize." +
174                     (char)27 + "[0m");
175         }
```

```
176     }
177
178     /**
179      * print an error message and call usage()
180      * @param msg
181      */
182     private static void error(String msg) {
183         System.err.println(msg);
184         usage();
185     }
186
187     /**
188      * usage message called when program improperly used
189      */
190     private static void usage() {
191         System.err.println(
192                 "usage: java Chirp <graph type> <num vertices> <num ticks> "
193                 + "<output image> {(<seed> <edge probability>), or "
194                 + "(<k>), or "
195                 + "(<k> <seed> <rewire probability>), or "
196                 + "(<dE> <seed>)}");
197         System.exit(1);
198     }
199 }
200
```