

Chirp.java

```
1 import java.io.IOException;
4
5 /**
6  *
7  * @author jimiford
8  *
9  */
10 public class Chirp {
11
12     private static final int GRAPH_TYPE_INDEX = 0,
13                             NUM_VERTICES_INDEX = 1,
14                             NUM_TICKS_INDEX = 2,
15                             OUTPUT_IMAGE_INDEX = 3,
16                             SEED_INDEX = 4,
17                             K_INDEX = 4,
18                             DE_INDEX = 4,
19                             DE_SEED_INDEX = 5,
20                             EDGE_PROBABILITY_INDEX = 5,
21                             K_SEED_INDEX = 5,
22                             REWIRE_PROBABILITY_INDEX = 6;
23
24     public static void main(String[] args) {
25         if(args.length != 4 && args.length != 5 &&
26            args.length != 6 && args.length != 7) usage();
27         int crickets = 0, ticks = 0, k = 0, dE = 0;
28         long seed = 0;
29         double prob = 0;
30         char mode;
31         String outputImage = args[OUTPUT_IMAGE_INDEX];
32
33         try {
34             crickets = Integer.parseInt(args[NUM_VERTICES_INDEX]);
35         } catch (NumberFormatException e) {
36             error("<num vertices> must be a number");
37         }
38         try {
39             ticks = Integer.parseInt(args[NUM_TICKS_INDEX]) + 1;
40         } catch (NumberFormatException e) {
41             error("<num ticks> must be numeric");
42         }
43         mode = args[GRAPH_TYPE_INDEX].toLowerCase().charAt(0);
```

Chirp.java

```
44     if(!(mode == 'c' || mode == 'r' || mode == 'k' ||
45         mode == 's' || mode == 'f')) {
46         error("<graph type> must be either 'c' for cycle, "
47             + "'r' for random, "
48             + "'k' for k-regular, "
49             + "'s' for small-world, "
50             + "'f' for scale-free");
51     }
52     UndirectedGraph g = null;
53     CricketObserver o = new CricketObserver(crickets, ticks);
54     switch(mode) {
55     case 'r': // RANDOM GRAPH
56         try {
57             seed = Long.parseLong(args[SEED_INDEX]);
58             prob =
59             Double.parseDouble(args[EDGE_PROBABILITY_INDEX]);
60             g = UndirectedGraph.randomGraph(new Random(seed),
61             crickets, prob, o);
62         } catch (NumberFormatException e) {
63             error("<seed> and <edge probability> must be
64             numeric");
65         } catch (IndexOutOfBoundsException e) {
66             error("<seed> and <edge probability> must be
67             included with random graph mode");
68         }
69         break;
70     case 'c': // CYCLE GRAPH
71         g = UndirectedGraph.cycleGraph(crickets, o);
72         break;
73     case 'k': // K-REGULAR GRAPH
74         try {
75             k = Integer.parseInt(args[K_INDEX]);
76             g = UndirectedGraph.kregularGraph(crickets, k, o);
77         } catch (NumberFormatException e) {
78             error("<k> must be an integer");
79         } catch (IllegalArgumentException e) {
80             error("<k> must be < the number of crickets");
81         }
82         break;
83     case 's': // SMALL WORLD GRAPH
84         try {
```

Chirp.java

```
81         k = Integer.parseInt(args[K_INDEX]);
82         prob =
Double.parseDouble(args[REWIRE_PROBABILITY_INDEX]);
83         seed = Long.parseLong(args[K_SEED_INDEX]);
84         g = UndirectedGraph.smallWorldGraph(new
Random(seed), crickets, k, prob, o);
85     } catch (NumberFormatException e) {
86         error("<k> must be an integer < V, <rewire
probability> must be a number "
87             + "between 0 and 1, and <seed> must be
numeric");
88     } catch (IllegalArgumentException e) {
89         error("<k> must be < the number of crickets");
90     }
91     break;
92     case 'f':
93     try {
94         dE = Integer.parseInt(args[DE_INDEX]);
95         seed = Long.parseLong(args[DE_SEED_INDEX]);
96         g = UndirectedGraph.scaleFreeGraph(new Random(seed),
crickets, dE, o);
97     } catch (NumberFormatException e) {
98         error("<dE> and <seed> must be numeric");
99     } catch (IndexOutOfBoundsException e) {
100         error("<dE> and <seed> must be supplied");
101     }
102 }
103
104 g.vertices.get(0).forceChirp();
105 Ticker.tick(g, ticks);
106
107
108
109 try {
110     ImageHandler.handle(o, outputImage);
111 } catch (IOException e) {
112     error("Problem writing image");
113 }
114 int sync = o.sync();
115 String description;
116 switch(mode) {
```

Chirp.java

```

117         case 'c': // CYCLE GRAPH
118             description = "Cycle V = " + crickets + ":";
119             handleOutput(description, sync);
120             break;
121         case 'r': // RANDOM GRAPH
122             description = "Random V = " + crickets + ", p = " + prob
+ ":";
123             handleOutput(description, sync);
124             break;
125         case 'k': // K-REGULAR GRAPH
126             description = "K-regular V = " + crickets + ", k = " + k
+ ":";
127             handleOutput(description, sync);
128             break;
129         case 's': // SMALL-WORLD GRAPH
130             description = "Small-world V = " + crickets + ", k = " +
k +
131                 ", p = " + prob + ":";
132             handleOutput(description, sync);
133             break;
134         case 'f': // SCALE-FREE GRAPH
135             description = "Scale-free V = " + crickets + ", dE = " +
dE + ":";
136             handleOutput(description, sync);
137             break;
138     }
139
140 }
141
142 private static void handleOutput(String description, int sync) {
143     System.out.print(description);
144     if(sync >= 0) {
145         System.out.println("\t" + "synchronized at t=" + sync + ".");
146     } else {
147         System.out.println("\t " + (char)27 + "[31m" + "did not
synchronize." +
148             (char)27 + "[0m");
149     }
150 }
151
152 private static void error(String msg) {

```

Chirp.java

```
153         System.err.println(msg);
154         usage();
155     }
156
157     private static void usage() {
158         System.err.println("usage: java Chirp <graph type> <num
vertices> <num ticks> "
159             + "<output image> {(<seed> <edge probability>), or "
160             + "(<k>), or "
161             + "(<k> <seed> <rewire probability>), or "
162             + "(<dE> <seed>)}");
163         System.exit(1);
164     }
165 }
166
```