```java
import java.io.IOException;

/**
 *
 * @author jimiford
 *
 */
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;

    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' ||
        mode == 's' || mode == 'f')) {
      error("<graph type> must be either 'c' for cycle, "
          + "'r' for random, "
          + "'k' for k-regular, "
          + "'s' for small-world, "
          + "'f' for scale-free");
    }
    UndirectedGraph g = null;
    CricketObserver o = new CricketObserver(crickets, ticks);
    switch(mode) {
    case 'r': // RANDOM GRAPH
      try {
        seed = Long.parseLong(args[SEED_INDEX]);
        prob = Double.parseDouble(args[EDGE_PROBABILITY_INDEX]);
        g = UndirectedGraph.randomGraph(new Random(seed), crickets, prob, o);
      } catch(NumberFormatException e) {
        error("<seed> and <edge probability> must be numeric");
      } catch(IndexOutOfBoundsException e) {
        error("<seed> and <edge probability> must be included with random graph mode");
      }
      break;
    case 'c': // CYCLE GRAPH
      g = UndirectedGraph.cycleGraph(crickets, o);
      break;
    case 'k': // K-REGULAR GRAPH
      try {
        k = Integer.parseInt(args[K_INDEX]);
        g = UndirectedGraph.kregularGraph(crickets, k, o);
      } catch (NumberFormatException e) {
        error("<k> must be an integer");
      } catch (IllegalArgumentException e) {
        error("<k> must be < the number of crickets");
      }
      break;
    case 's': // SMALL WORLD GRAPH
      try {
```

```java
                k = Integer.parseInt(args[K_INDEX]);
                prob = Double.parseDouble(args[REWIRE_PROBABILITY_INDEX]);
                seed = Long.parseLong(args[K_SEED_INDEX]);
                g = UndirectedGraph.smallWorldGraph(new Random(seed), crickets, k, prob, o);
            } catch (NumberFormatException e) {
                error("<k> must be an integer < V, <rewire probability> must be a number "
                        + "between 0 and 1, and <seed> must be numeric");
            } catch (IllegalArgumentException e) {
                error("<k> must be < the number of crickets");
            }
            break;
        case 'f':
            try {
                dE = Integer.parseInt(args[DE_INDEX]);
                seed = Long.parseLong(args[DE_SEED_INDEX]);
                g = UndirectedGraph.scaleFreeGraph(new Random(seed), crickets, dE, o);
            } catch (NumberFormatException e) {
                error("<dE> and <seed> must be numeric");
            } catch (IndexOutOfBoundsException e) {
                error("<dE> and <seed> must be supplied");
            }
        }

        g.vertices.get(0).forceChirp();
        Ticker.tick(g, ticks);



        try {
            ImageHandler.handle(o, outputImage);
        } catch (IOException e) {
            error("Problem writing image");
        }
        int sync = o.sync();
        String description;
        switch(mode) {
```

```java
        case 'c': // CYCLE GRAPH
            description = "Cycle V = " + crickets +":";
            handleOutput(description,sync);
            break;
        case 'r': // RANDOM GRAPH
            description = "Random V = " + crickets +", p = " + prob + ":";
            handleOutput(description,sync);
            break;
        case 'k': // K-REGULAR GRAPH
            description = "K-regular V = " + crickets +", k = " + k + ":";
            handleOutput(description,sync);
            break;
        case 's': // SMALL-WORLD GRAPH
            description = "Small-world V = " + crickets + ", k = " + k +
                ", p = " + prob + ":";
            handleOutput(description,sync);
            break;
        case 'f': // SCALE-FREE GRAPH
            description = "Scale-free V = " + crickets +", dE = " + dE + ":";
            handleOutput(description,sync);
            break;
        }

    }

    private static void handleOutput(String description, int sync) {
        System.out.print(description);
        if(sync >= 0) {
            System.out.println("\t"+" synchronized at t="+sync+".");
        } else {
            System.out.println("\t "+(char)27+"[31m"+  "did not synchronize." +
                    (char)27 + "[0m");
        }
    }

    private static void error(String msg) {
        System.err.println(msg);
        usage();
    }

    private static void usage() {
        System.err.println("usage: java Chirp <graph type> <num vertices> <num ticks> "
                + "<output image> {(<seed> <edge probability>), or "
                + "(<k>), or "
                + "(<k> <seed> <rewire probability>), or "
                + "(<dE> <seed>)}");
        System.exit(1);
    }
}

```