```
 1: // $Id: blockingqueue.java,v 1.3 2013-03-08 13:23:14-08 - - $
 2:
 3: //
 4: // Example of the use of a LinkedBlockingQueue.
 5: //
 6:
 7: import java.util.*;
 8: import java.util.concurrent.*;
 9: import static java.lang.System.*;
10:
11: class blockingqueue {
12:
13:    //
14:    // Class message includes the String message and a status.
15:    // NORMAL: usual case for receipt of message
16:    // TIMEOUT: polling timed out
17:    // INTERRUPT: some interrupt other than TIMEOUT
18:    // EOF: end of file
19:    //
20:    static enum status_t {NORMAL, TIMEOUT, INTERRUPT, EOF};
21:    static class message {
22:       String data;
23:       status_t status;
24:       message (String data, status_t status) {
25:          this.data = data;
26:          this.status = status;
27:       }
28:    };
29:
30:    //
31:    // Put thread itself to sleep for a while.
32:    //
33:    static void sleep (int msec) {
34:       try {
35:          Thread.sleep (msec);
36:       }catch (InterruptedException exn) {
37:       }
38:    }
39:
40:    //
41:    // Poll a queue, waiting for some number of milliseconds.
42:    // Return message or condition.  Used to avoid duplicate
43:    // code in polling different queues.
44:    //
45:    static message poll (BlockingQueue <message> source, long msec) {
46:       message msg = null;
47:       try {
48:          msg = source.poll (msec, TimeUnit.MILLISECONDS);
49:          if (msg == null) msg = new message (null, status_t.TIMEOUT);
50:       }catch (InterruptedException exn) {
51:          msg = new message (null, status_t.INTERRUPT);
52:       }
53:       return msg;
54:    }
55:
56:    //
57:    // Put a new element into a blockingqueue, waiting if necessary
58:    // if the queue is full.  Used to avoid duplicate code.
59:    //
60:    static void put (BlockingQueue <message> source, message msg) {
61:       try {
62:          source.put (msg);
63:       }catch (InterruptedException exn) {
64:          out.printf ("%s%n", exn);
```

```
65:       }
66:    }
67:
68:    //
69:    // Accepts a message from a source queue and distributes it to
70:    // each of some number of other queues.
71:    //
72:    static class distributor implements Runnable {
73:       BlockingQueue <message> source;
74:       List <BlockingQueue <message>> targets;
75:       distributor (BlockingQueue <message> source,
76:                    List <BlockingQueue <message>> targets) {
77:          this.source = source;
78:          this.targets = targets;
79:       }
80:       public void run () {
81:          for (;;) {
82:             message msg = poll (source, 5000);
83:             for (BlockingQueue <message> target: targets) {
84:                put (target, msg);
85:             }
86:          }
87:       }
88:    }
89:
90:    //
91:    // Copies its own queue elements to stdout.  In this example
92:    // there is a race condition on stdout, so we put the message
93:    // into a string and then print as a unit.  Note that the order
94:    // of echoing each message by the several threads varies.
95:    //
96:    static class copier implements Runnable {
97:       BlockingQueue <message> source;
98:       copier (BlockingQueue <message> source) {
99:          this.source = source;
100:      }
101:      public void run () {
102:         String name = Thread.currentThread ().getName ();
103:         for (;;) {
104:            message msg = poll (source, 5000);
105:            String line = name + ": " + msg.status;
106:            if (msg.data != null) line += ": " + msg.data;
107:            out.println (line);
108:            if (msg.status == status_t.EOF) return;
109:         }
110:      }
111:   }
112:
113:   //
114:   // Create a queue for the distributor, and one of each copier.
115:   // Start the copiers and distributors.  The copiers run until
116:   // EOF, but the distributor is a daemon.  Main copies stdin to
117:   // the distributor until EOF.
118:   //
119:   public static void main (String[] args) {
120:      String[] names
121:            = {"Prôtos", "Deuteros", "Tritos", "Tetratos", "Pentos"};
122:
123:      BlockingQueue <message> source
124:            = new LinkedBlockingQueue <message> (5);
125:      List <BlockingQueue <message>> targets
126:            = new ArrayList <BlockingQueue <message>> ();
127:
128:      for (int itor = 0; itor < names.length; ++itor) {
```

```
129:          BlockingQueue <message> target
130:             = new LinkedBlockingQueue <message> (5);
131:          targets.add (target);
132:          Thread copy
133:                = new Thread (new copier (target), names[itor]);
134:          copy.start ();
135:       }
136:
137:       Thread distrib
138:             = new Thread (new distributor (source, targets));
139:       distrib.setDaemon (true);
140:       distrib.start ();
141:
142:       Scanner stdin = new Scanner (in);
143:       while (stdin.hasNext ()) {
144:          put (source, new message (stdin.next (), status_t.NORMAL));
145:       }
146:       put (source, new message (null, status_t.EOF));
147:    }
148:
149: }
```