



- 116 Removals + 54 Additions

7 java

```

1 package server.main;
2
3 import java.util.NoSuchElementException;
4 import java.util.Queue;
5 import java.util.concurrent.ConcurrentLinkedQueue;
6
7 import server.services.protocol.InputMessageQueue;
8
9 /**
10  * A threadsafe global queue of all the messages received by
11  * all clients
12  * connected to the server. The server is constantly process
13  * ing this queue.
14  *
15  * The queue uses Round-Robin scheduling; that is, messages
16  * of higher priority
17  * are always dequeued before messages of lower priority, an
18  * d it follows a FIFO
19  * scheme for messages with the same priority.
20  *
21  * @author Adrian Petrescu
22  */
23 public class GlobalInputMessageQueue {
24
25     private static GlobalInputMessageQueue self;
26
27     /**
28      * Singleton accessor to the GlobalInputMessageQueue.
29      * If the queue has not
30      * yet been accessed, it will be created.
31      *
32      * @return A reference to the GlobalInputMessageQueue.
33      */
34     public static GlobalInputMessageQueue
35     getGlobalInputMessageQueue() {
36         if (self == null) {
37             self
38             = new GlobalInputMessageQueue();
39         }
40         return self;
41     }
42
43     protected Queue<InputMessageQueue> lowPriorityInputM
44     essageQueue;
45     protected Queue<InputMessageQueue> medPriorityInputM
46     essageQueue;
47     protected Queue<InputMessageQueue> highPriorityInput
48     MessageQueue;
49
50     /**

```

```

1 package server.main;
2
3 import java.util.EnumMap;
4 import java.util.NoSuchElementException;
5 import java.util.Queue;
6 import java.util.concurrent.ConcurrentLinkedQueue;
7
8 import server.services.protocol.InputMessageQueue;
9
10 /**
11  * A threadsafe global queue of all the messages received by
12  * all clients
13  * connected to the server. The server is constantly process
14  * ing this queue.
15  *
16  * The queue uses Round-Robin scheduling; that is, messages
17  * of higher priority
18  * are always dequeued before messages of lower priority, an
19  * d it follows a FIFO
20  * scheme for messages with the same priority.
21  *
22  * @author John Smith :)
23  */
24 public class GlobalInputMessageQueue {
25
26     private static GlobalInputMessageQueue instance;
27     public static GlobalInputMessageQueue getInstance()
28     {
29         if (instance == null) {
30             instance
31             = new GlobalInputMessageQueue();
32         }
33         return instance;
34     }
35
36     protected EnumMap<QueuePriority, ConcurrentLinkedQueue<I
37     nputMessageQueue>> queues;
38
39
40
41
42
43
44
45
46
47
48
49
50
51

```

```

40     * Creates a new instance of GlobalInputMessageQueue
e.
41     */
42     protected GlobalInputMessageQueue() {
43         lowPriorityInputMessageQueue = new ConcurrentLinkedQueue<InputMessageQueue>();
44         medPriorityInputMessageQueue = new ConcurrentLinkedQueue<InputMessageQueue>();
45         highPriorityInputMessageQueue = new ConcurrentLinkedQueue<InputMessageQueue>();
46     }
47
48     /**
49     * Add a message to the global queue.
50     *
51     * @param message The message to be queued up.
52     */
53     @SuppressWarnings("deprecation")
54     public synchronized void enqueue(InputMessageQueue message) {
55         boolean unlockListener = this.isEmpty();
56         switch (message.getPriority()) {
57             case 0:
58                 lowPriorityInputMessageQueue.add(message);
59                 break;
60             case 1:
61                 medPriorityInputMessageQueue.add(message);
62                 break;
63             case 2:
64                 medPriorityInputMessageQueue.add(message);
65                 break;
66         }
67         if (unlockListener) {
68             /* TODO: According to my experiment
69             s, main is always the first thread in
70             * the main thread group, but I'm not
71             t sure if this is always
72             * necessarily the case. Do more research
73             here.
74             *
75             * Assuming that it is saves time, but
76             ut may cause deadlock if we're
77             * wrong.
78             *
79             * If you have definite knowledge about
80             the likelihood of main not
81             * being the first thread, please file a
82             ticket!
83             */
84             Thread[] threads = new Thread[1];
85             Thread.currentThread().getThreadGroup().enumerate(threads);
86
87             if (threads[0].getName().equals("main")) {
88                 threads[0].resume();
89             }
90         }
91     }

```

```

32     protected GlobalInputMessageQueue() {
33         queues = new EnumMap<>(){{
34             put(QueuePriority.LOW, new ConcurrentLinkedQueue<InputMessageQueue>());
35             put(QueuePriority.MED, new ConcurrentLinkedQueue<InputMessageQueue>());
36             put(QueuePriority.HIGH, new ConcurrentLinkedQueue<InputMessageQueue>());
37         }};
38     }
39
40     @SuppressWarnings("deprecation")
41     public synchronized void enqueue(InputMessageQueue message) {
42         switch (message.getPriority()) {
43             case 0:
44                 addToQueue(Priority.LOW, message);
45                 break;
46             case 1:
47                 addToQueue(Priority.MED, message);
48                 break;
49             case 2:
50                 addToQueue(Priority.HIGH, message);
51                 break;
52         }
53         if (this.isEmpty()) {
54             Thread[] threads = new Thread[1];
55             Thread.currentThread().getThreadGroup().enumerate(threads);
56
57             if (threads[0].getName().equals("main")) {
58                 threads[0].resume();
59             }
60         }

```

```

82     }
83
84     /**
85      * @return The oldest message of the highest priority
86      * available, or NULL if the
87      * queue is empty.
88      */
89     public synchronized InputMessageQueue dequeue() {
90         try {
91             return
92             highPriorityInputMessageQueue.remove();
93         } catch (NoSuchElementException noHigh) {
94             try {
95                 return
96                 medPriorityInputMessageQueue.remove();
97             } catch (NoSuchElementException noMed) {
98                 try {
99                     return
100                     lowPriorityInputMessageQueue.remove();
101                 } catch (NoSuchElementException noLow) {}
102             }
103             return null;
104         }
105     }
106
107     /**
108      * Return the total number of input message queues still
109      * queued up
110      * in the global buffer, of all priorities.
111      *
112      * @return The total number of queued InputMessageStacks.
113      */
114     public int getSize() {
115         return lowPriorityInputMessageQueue.size()
116             +
117             medPriorityInputMessageQueue.size()
118             +
119             highPriorityInputMessageQueue.size();
120     }
121
122     /**
123      * Return the total number of input message queues still
124      * in the queue
125      * with a given priority.
126      *
127      */

```

```

58     }
59
60     private synchronized void addToQueue(QueuePriority priority, InputMessageQueue message){
61         getQueue(priority).add(message);
62     }
63
64     private synchronized ConcurrentLinkedQueue<InputMessageQueue> getQueue(QueuePriority priority) {
65         return queues.get(priority);
66     }
67
68
69     public synchronized InputMessageQueue dequeue() {
70
71         try {
72             return
73             getQueue(QueuePriority.HIGH).remove();
74         } catch (NoSuchElementException noHigh) {
75             try {
76                 return
77                 getQueue(QueuePriority.MED).remove();
78             } catch (NoSuchElementException noMed) {
79                 try {
80                     return
81                     getQueue(QueuePriority.LOW).remove();
82                 } catch (NoSuchElementException noLow) {}
83             }
84             return null;
85         }
86     }
87
88     public int getSize() {
89         return getQueue(QueuePriority.LOW).size()
90             +
91             getQueue(QueuePriority.MED).size()
92             +
93             getQueue(QueuePriority.HIGH).size();
94     }
95
96     public int getSize(QueuePriority priority) {
97         return getQueue(priority).size();
98     }

```

```

    * @param priority The priority level of counted mes
sages.
    * @return The total number of queued InputMessageSt
acks of the given
120    * priority.
121    */
122    public int getSize(int priority) {
123        switch (priority) {
124            case 0:
125                return lowPriorityInputMessageQueue.
126                size();
127            case 1:
128                return medPriorityInputMessageQueue.
129                size();
130            case 2:
131                return highPriorityInputMessageQueue.
132                size();
133            default:
134                return 0;
135        }
136    }
137
138    /**
139     * Returns <code>true</code> if the queue contains n
o elements of any
140     * priority.
141     * @return <code>true</code> if the queue is complet
ely empty.
142     */
143    public synchronized boolean isEmpty() {
144        return
145        lowPriorityInputMessageQueue.isEmpty()
146        &&
147        medPriorityInputMessageQueue.isEmpty()
148        &&
149        highPriorityInputMessageQueue.isEmpty();
150    }
151
152    /**
153     * Returns <code>true</code> if the queue contains n
o elements of the
154     * given priority.
155     * @param priority The priority level to check.
156     * @return <code>true</code> if the queue is empty o
f messages of the given
157     * priority.
158     */
159    public boolean isEmpty(int priority) {
160        switch (priority) {
161            case 0:
162                return lowPriorityInputMessageQueue.
163                isEmpty();
164            case 1:
165                return medPriorityInputMessageQueue.
166                isEmpty();
167            case 2:
168                return highPriorityInputMessageQueue.
169                isEmpty();
170            default:
171                return true;
172        }
173    }

```

```

93    }
94
95    public synchronized boolean isEmpty() {
96        return getQueue(QueuePriority.LOW).isEmpty()
97        &&
98        getQueue(QueuePriority.MED).isEmpty()
99        &&
100        getQueue(QueuePriority.HIGH).isEmpty();
101    }
102
103    public boolean isEmpty(QueuePriority priority) {
104        return getQueue(priority).isEmpty();
105    }

```

```
165 return true;
166 }
167 }
168
169 }
```

```
102 }
103
104 enum QueuePriority {
105     LOW,
106     MED,
107     HIGH
108 }
109 }
```