

1 BitBufferSafeletExecutor

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.safetycritical.Safelet;
4 import javax.safetycritical.SafeletExecutor;
5
6 //Application entry point, runs the Safelet
7 public class BitBufferSafeletExecutor extends SafeletExecutor
8 {
9
10     public BitBufferSafeletExecutor(Safelet arg0)
11     {
12         super(arg0);
13     }
14
15     public static void main (String [] args)
16     {
17         System.out.println("BitBufferSafeletExecutor ");
18         //Run the safelet which starts the whole application
19         BitBufferSafeletExecutor.run(new BitBuffer());
20     }
21 }
```

2 BitBuffer

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PriorityParameters;
4 import javax.safetycritical.*;
5 import javax.safetycritical.annotate.Level;
6
7 public class BitBuffer implements Safelet
8 {
9
10     @Override
11     public Level getLevel()
12     {
13         return Level.LEVEL_2;
14     }
15
16     @Override
17     public MissionSequencer getSequencer()
18     {
19         System.out.println("BitBuffer");
20         //Create and return the main mission sequencer
21         return new BitBufferMissionSequencer(new PriorityParameters(5),
22             new StorageConfigurationParameters(1048576, 1048576, 1048576));
23     }
24
25     @Override
26     public void setup()
27     {
28     }
29
30     @Override
31     public void teardown()
32     {
33     }
34
35 }
36 }
```

3 BitBufferMissionSequencer

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PriorityParameters;
4 import javax.safetycritical.Mission;
5 import javax.safetycritical.MissionSequencer;
6 import javax.safetycritical.StorageConfigurationParameters;
7
8
9 public class BitBufferMissionSequencer extends MissionSequencer
10 {
11     private boolean returnedMission;
12
13     public BitBufferMissionSequencer(PriorityParameters priorityParameters,
14         StorageConfigurationParameters storageConfigurationParameters)
15     {
16         super(priorityParameters, storageConfigurationParameters);
17         returnedMission = false;
18     }
19
20     @Override
21     protected Mission getNextMission()
22     {
23         System.out.println("BitBufferMissionSequencer");
24
25         //As this sequencer only delivers one mission,
26         //if it has not been returned yet then return it,
27         //else return null which will terminate the sequencer
28
29         if(!returnedMission)
30         {
31             System.out.println("BitBufferMissionSequencer returns mission");
32             returnedMission = true;
33             return new BitBufferMission();
34         }
35         else
36         {
37             return null;
38         }
39     }
40
41
42
43 }
```

4 BitBufferMission

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PeriodicParameters;
4 import javax.realtime.PriorityParameters;
5 import javax.realtime.RelativeTime;
6 import javax.safetycritical.Mission;
7 import javax.safetycritical.PriorityScheduler;
8 import javax.safetycritical.StorageConfigurationParameters;
9
10
11 public class BitBufferMission extends Mission
12 {
13     private volatile int[] buffer;
14     private WriterMissionSequencer writerMissionSequencer;
15     private ReaderMissionSequencer readerMissionSequencer;
16     private boolean terminatingPending;
17
18     @Override
19     protected void initialize()
20     {
21         //start the two submission sequencers, note a reference to this object is
22         passed to both so that they can access the buffer
23         writerMissionSequencer = new WriterMissionSequencer (new PriorityParameters
24             (5),
25             new StorageConfigurationParameters(1048576, 1048576, 1048576),
26             this );
27         writerMissionSequencer.register();
28
29         readerMissionSequencer = new ReaderMissionSequencer (new PriorityParameters
30             (5),
31             new StorageConfigurationParameters(1048576, 1048576, 1048576),
32             this);
33         readerMissionSequencer.register();
34
35         buffer = new int [1];
36         buffer[0]= 0;
37         terminatingPending = false;
38
39         System.out.println("BitBufferMission");
40     }
41
42     public boolean bufferEmpty()
43     {
44         return buffer[0] == 0;
45     }
46
47     public void notifyReader()
48     {
49         readerMissionSequencer.notifyReader();
50     }
51
52     public void notifyWriter()
53     {
54         writerMissionSequencer.notifyWriter();
55     }
56
57     public void write(int update)
58     {
59         this.buffer[0] = update;
60     }
61
62     public int read()
63     {
64         int out = buffer[0];
65         this.buffer[0] = 0;
66         return out;
67     }
68 }
```

```
64 public boolean terminationPending()  
65 {  
66     return terminatingPending;  
67 }  
68  
69 @Override  
70 public long missionMemorySize()  
71 {  
72     return 100000;  
73 }  
74  
75 }
```

5 WriterMissionSequencer

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PriorityParameters;
4 import javax.safetycritical.Mission;
5 import javax.safetycritical.MissionSequencer;
6 import javax.safetycritical.StorageConfigurationParameters;
7
8
9 public class WriterMissionSequencer extends MissionSequencer
10 {
11     private boolean returnedMission;
12     private final BitBufferMission bbMission;
13     private WriterMission writerMission;
14
15     public WriterMissionSequencer(PriorityParameters priorityParameters,
16         StorageConfigurationParameters storageConfigurationParameters,
17         BitBufferMission bbMission)
18     {
19         super(priorityParameters, storageConfigurationParameters);
20         returnedMission = false;
21         this.bbMission = bbMission;
22
23         System.out.println("WriterMissionSequencer constructor");
24     }
25
26     public void notifyWriter()
27     {
28         writerMission.notifyWriter();
29     }
30
31     @Override
32     protected Mission getNextMission()
33     {
34         System.out.println("WriterMissionSequencer getNextMission");
35
36         //As this sequencer only delivers one mission,
37         //if it has not been returned yet then return it,
38         //else return null which will terminate the sequencer
39
40         if(!returnedMission)
41         {
42             System.out.println("WriterMissionSequencer returns mission");
43             writerMission = new WriterMission(bbMission);
44             returnedMission = true;
45             return writerMission;
46         }
47         else
48         {
49             return null;
50         }
51     }
52 }
```

6 WriterMission

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PeriodicParameters;
4 import javax.realtime.PriorityParameters;
5 import javax.realtime.RelativeTime;
6 import javax.safetycritical.Mission;
7 import javax.safetycritical.PriorityScheduler;
8 import javax.safetycritical.StorageConfigurationParameters;
9
10
11 public class WriterMission extends Mission
12 {
13     private final BitBufferMission bbMission;
14     private Writer writer;
15
16     public WriterMission(BitBufferMission bbMission)
17     {
18         super();
19         this.bbMission= bbMission;
20     }
21
22     public void notifyWriter()
23     {
24         writer.notifyWriter();
25     }
26
27     @Override
28     protected void initialize()
29     {
30         //Start this mission's handler
31         writer = new Writer(new PriorityParameters(10),
32                             new StorageConfigurationParameters(1000, 1000, 1000),
33                             bbMission);
34         writer.register();
35     }
36
37
38     @Override
39     public long missionMemorySize()
40     {
41         return 100000;
42     }
43
44 }
```

7 Writer

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PeriodicParameters;
4 import javax.realtime.PriorityParameters;
5 import javax.safetycritical.ManagedThread;
6 import javax.safetycritical.StorageConfigurationParameters;
7
8 import java.io.*;
9
10
11 public class Writer extends ManagedThread
12 {
13     private final BitBufferMission bbMission;
14     private int i;
15
16     public Writer(PriorityParameters priority, StorageConfigurationParameters storage
17         , BitBufferMission bbMission)
18     {
19         super(priority, storage);
20         this.bbMission = bbMission;
21     }
22
23     public void notifyWriter()
24     {
25         notify();
26     }
27
28     @Override
29     public void run()
30     {
31         while(! bbMission.terminationPending())
32         {
33             if(bbMission.bufferEmpty())
34             {
35                 bbMission.write(i);
36                 i++;
37                 bbMission.notifyReader();
38                 try{
39                     Thread.sleep(500);
40                 }
41                 catch(InterruptedException e) {}
42             }
43             else
44             {
45                 wait();
46             }
47         }
48     }
49 }
50
51 }
52 }
```


8 ReaderMissionSequencer

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PriorityParameters;
4 import javax.safetycritical.Mission;
5 import javax.safetycritical.MissionSequencer;
6 import javax.safetycritical.StorageConfigurationParameters;
7
8
9 public class ReaderMissionSequencer extends MissionSequencer
10 {
11     private boolean returnedMission;
12     private final BitBufferMission bbMission;
13     private ReaderMission readerMission;
14
15     public ReaderMissionSequencer(PriorityParameters priorityParameters,
16         StorageConfigurationParameters storageConfigurationParameters,
17         BitBufferMission bbMission)
18     {
19         super(priorityParameters, storageConfigurationParameters);
20         returnedMission = false;
21         this.bbMission = bbMission;
22         readerMission = new ReaderMission(bbMission);
23         System.out.println("ReaderMissionSequencer constructor");
24     }
25
26     public void notifyReader()
27     {
28         readerMission.notifyReader();
29     }
30
31     @Override
32     protected Mission getNextMission()
33     {
34         System.out.println("ReaderMissionSequencer getNextMission");
35
36         //As this sequencer only delivers one mission,
37         //if it has not been returned yet then return it,
38         //else return null which will terminate the sequencer
39         if(!returnedMission)
40         {
41             System.out.println("ReaderMissionSequencer returns mission");
42             returnedMission = true;
43             return readerMission;
44         }
45         else
46         {
47             return null;
48         }
49     }
50 }
51
52 }
```

9 ReaderMission

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PeriodicParameters;
4 import javax.realtime.PriorityParameters;
5 import javax.realtime.RelativeTime;
6 import javax.safetycritical.Mission;
7 import javax.safetycritical.PriorityScheduler;
8 import javax.safetycritical.StorageConfigurationParameters;
9
10
11 public class ReaderMission extends Mission
12 {
13     private final BitBufferMission bbMission;
14     private Reader reader;
15
16     public ReaderMission(BitBufferMission bbMission)
17     {
18         super();
19         this.bbMission= bbMission;
20     }
21
22     public void notifyReader()
23     {
24         reader.notifyReader();
25     }
26
27
28     @Override
29     protected void initialize()
30     {
31         //Start this mission's handler
32         reader = new Reader(new PriorityParameters(10),
33                             new StorageConfigurationParameters(1000, 1000, 1000),
34                             bbMission);
35         reader.register();
36     }
37
38
39     @Override
40     public long missionMemorySize()
41     {
42         return 100000;
43     }
44
45 }
```

10 Reader

```
1 package scjlevel2examples.bitbuffer;
2
3 import javax.realtime.PeriodicParameters;
4 import javax.realtime.PriorityParameters;
5 import javax.safetycritical.ManagedThread;
6 import javax.safetycritical.StorageConfigurationParameters;
7
8
9
10 public class Reader extends ManagedThread
11 {
12     private final BitBufferMission bbMission;
13
14     public Reader(PriorityParameters priority, StorageConfigurationParameters storage
15         , BitBufferMission bbMission)
16     {
17         super(priority, storage);
18         this.bbMission = bbMission;
19     }
20
21     public void notifyReader()
22     {
23         notify();
24     }
25
26     @Override
27     public void run()
28     {
29         while(! bbMission.terminationPending())
30         {
31             if (bbMission.bufferEmpty())
32             {
33                 wait();
34             }
35             else
36             {
37                 System.out.println("" + bbMission.read());
38                 bbMission.notifyWriter();
39                 try{
40                     Thread.sleep(500);
41                 }
42                 catch(InterruptedException e) {}
43             }
44         }
45     }
46 }
47
48 }
```