1 BitBufferSafeletExecuter

```java
package scjlevel2examples.bitbuffer;

import javax.safetycritical.Safelet;
import javax.safetycritical.SafeletExecuter;

//Application entry point, runs the Safelet
public class BitBufferSafeletExecuter extends SafeletExecuter
{

  public BitBufferSafeletExecuter(Safelet arg0)
  {
    super(arg0);
  }

  public static void main (String [] args)
  {
    System.out.println("BitBufferSafeletExecuter ");
    //Run the safelet which starts the whole application
    BitBufferSafeletExecuter.run(new BitBuffer());
  }
}
```

2 BitBuffer

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PriorityParameters;
import javax.safetycritical.*;
import javax.safetycritical.annotate.Level;

public class BitBuffer implements Safelet
{

  @Override
  public Level getLevel()
  {
    return Level.LEVEL_2;
  }

  @Override
  public MissionSequencer getSequencer()
  {
    System.out.println("BitBuffer");
    //Create and return the main mission sequencer
    return new BitBufferMissionSequencer(new PriorityParameters(5),
        new StorageConfigurationParameters(1048576, 1048576, 1048576));
  }

  @Override
  public void setup()
  {
  }

  @Override
  public void teardown()
  {
  }


}
```

3 BitBufferMissionSequencer

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PriorityParameters;
import javax.safetycritical.Mission;
import javax.safetycritical.MissionSequencer;
import javax.safetycritical.StorageConfigurationParameters;


public class BitBufferMissionSequencer extends MissionSequencer
{
  private boolean returnedMission;

  public BitBufferMissionSequencer(PriorityParameters priorityParameters,
      StorageConfigurationParameters storageConfigurationParameters)
  {
    super(priorityParameters, storageConfigurationParameters);
    returnedMission =false;
  }

  @Override
  protected Mission getNextMission()
  {
    System.out.println("BitBufferMissionSequencer");

    //As this sequencer only delivers one mission,
    //if it has not been returned yet then return it,
    //else return null which will terminate the sequencer

    if(!returnedMission)
    {
      System.out.println("BitBufferMissionSequencer returns mission");
      returnedMission = true;
      return new BitBufferMission();
    }
    else
    {
      return null;
    }
  }



}
```

4 BitBufferMission

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.realtime.RelativeTime;
import javax.safetycritical.Mission;
import javax.safetycritical.PriorityScheduler;
import javax.safetycritical.StorageConfigurationParameters;


public class BitBufferMission extends Mission
{
  private volatile int[] buffer;
  private WriterMissionSequencer writerMissionSequencer;
  private ReaderMissionSequencer readerMissionSequencer;
  private boolean terminatingPending;

  @Override
  protected void initialize()
  {
    //start the two submission sequencers, note a reference to this object is
        passed to both so that they can access the buffer
      writerMissionSequencer = new WriterMissionSequencer (new PriorityParameters
          (5),
                  new StorageConfigurationParameters(1048576, 1048576, 1048576),
                      this );
      writerMissionSequencer.register();

      readerMissionSequencer = new ReaderMissionSequencer (new PriorityParameters
          (5),
                  new StorageConfigurationParameters(1048576, 1048576, 1048576),
                      this);
      readerMissionSequencer.register();

      buffer = new int[1];
      buffer[0]= 0;
      terminatingPending = false;

      System.out.println("BitBufferMission");
  }

  public boolean bufferEmpty()
  {
    return buffer[0] == 0;
  }

  public void notifyReader()
  {
    readerMissionSequencer.notifyReader();
  }

  public void notifyWriter()
  {
    writerMissionSequencer.notifyWriter();
  }

  public void write(int update)
  {
    this.buffer[0] = update;
  }

  public int read()
  {
    int out = buffer[0];
    this.buffer[0] = 0;
    return out;
  }

  public boolean terminationPending()
  {
    return terminatingPending;
  }

  @Override
  public long missionMemorySize()
  {
    return 100000;
  }

}
```

5 WriterMissionSequencer

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PriorityParameters;
import javax.safetycritical.Mission;
import javax.safetycritical.MissionSequencer;
import javax.safetycritical.StorageConfigurationParameters;


public class WriterMissionSequencer extends MissionSequencer
{
  private boolean returnedMission;
  private final BitBufferMission bbMission;
  private WriterMission writerMission;

  public WriterMissionSequencer(PriorityParameters priorityParameters,
      StorageConfigurationParameters storageConfigurationParameters,
          BitBufferMission bbMission)
  {
    super(priorityParameters, storageConfigurationParameters);
    returnedMission = false;
    this.bbMission = bbMission;

    System.out.println("WriterMissionSequencer constructor");
  }

  public void notifyWriter()
  {
    writerMission.notifyWriter();
  }

  @Override
  protected Mission getNextMission()
  {
    System.out.println("WriterMissionSequencer getNextMission");

    //As this sequencer only delivers one mission,
    //if it has not been returned yet then return it,
    //else return null which will terminate the sequencer

    if(!returnedMission)
    {
      System.out.println("WriterMissionSequencer returns mission");
      writerMission = new WriterMission(bbMission);
      returnedMission = true;
      return writerMission;
    }
    else
    {
      return null;
    }
  }
}
```

6 WriterMission

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.realtime.RelativeTime;
import javax.safetycritical.Mission;
import javax.safetycritical.PriorityScheduler;
import javax.safetycritical.StorageConfigurationParameters;


public class WriterMission extends Mission
{
  private final BitBufferMission bbMission;
  private Writer writer;

  public WriterMission(BitBufferMission bbMission)
  {
    super();
    this.bbMission= bbMission;
  }

  public void notifyWriter()
  {
    writer.notifyWriter();
  }

  @Override
  protected void initialize()
  {
    //Start this mission's handler
    writer = new Writer(new PriorityParameters(10),
        new StorageConfigurationParameters(1000, 1000, 1000),
        bbMission);
    writer.register();

  }

  @Override
  public long missionMemorySize()
  {
    return 100000;
  }

}
```

7 Writer

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.safetycritical.ManagedThread;
import javax.safetycritical.StorageConfigurationParameters;

import java.io.*;


public class Writer extends ManagedThread
{
  private final BitBufferMission bbMission;
  private int i;

  public Writer(PriorityParameters priority, StorageConfigurationParameters storage
      , BitBufferMission bbMission)
  {
    super(priority,storage);

    this.bbMission = bbMission;

  }

  public void notifyWriter()
  {
    notify();
  }

  @Override
  public void run()
  {
    while(! bbMission.terminationPending())
    {
      if(bbMission.bufferEmpty())
      {
        bbMission.write(i);
        i ++;
        bbMission.notifyReader();
        try{
          Thread.sleep(500);
        }
        catch(InterruptedException e) {}

      }
      else
      {
        wait();
      }
    }

  }
}
```

8 ReaderMissionSequencer

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PriorityParameters;
import javax.safetycritical.Mission;
import javax.safetycritical.MissionSequencer;
import javax.safetycritical.StorageConfigurationParameters;


public class ReaderMissionSequencer extends MissionSequencer
{
  private boolean returnedMission;
  private final BitBufferMission bbMission;
  private ReaderMission readerMission;

  public ReaderMissionSequencer(PriorityParameters priorityParameters,
      StorageConfigurationParameters storageConfigurationParameters,
          BitBufferMission bbMission)
  {
    super(priorityParameters, storageConfigurationParameters);
    returnedMission = false;
    this.bbMission = bbMission;
    readerMission = new ReaderMission(bbMission);
    System.out.println("ReaderMissionSequencer constructor");
  }


  public void notifyReader()
  {
    readerMission.notifyReader();
  }


  @Override
  protected Mission getNextMission()
  {
    System.out.println(" ReaderMissionSequencer getNextMission");

    //As this sequencer only delivers one mission,
    //if it has not been returned yet then return it,
    //else return null which will terminate the sequencer
    if(!returnedMission)
    {
      System.out.println("ReaderMissionSequencer returns mission");
      returnedMission = true;
      return readerMission;
    }
    else
    {
      return null;
    }
  }

}
```

9 ReaderMission

```java
package scjlevel2examples.bitbuffer;

import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.realtime.RelativeTime;
import javax.safetycritical.Mission;
import javax.safetycritical.PriorityScheduler;
import javax.safetycritical.StorageConfigurationParameters;


public class ReaderMission extends Mission
{
  private final BitBufferMission bbMission;
  private Reader reader;

  public ReaderMission(BitBufferMission bbMission)
  {
    super();
    this.bbMission= bbMission;
  }

  public void notifyReader()
  {
    reader.notifyReader();
  }


  @Override
  protected void initialize()
  {
    //Start this mission's handler
    reader = new Reader(new PriorityParameters(10),
        new StorageConfigurationParameters(1000, 1000, 1000),
        bbMission);
    reader.register();

  }

  @Override
  public long missionMemorySize()
  {
    return 100000;
  }

}
```

10 Reader

```
package scjlevel2examples.bitbuffer;

import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.safetycritical.ManagedThread;
import javax.safetycritical.StorageConfigurationParameters;



public class Reader extends ManagedThread
{
  private final BitBufferMission bbMission;

  public Reader(PriorityParameters priority, StorageConfigurationParameters storage
      , BitBufferMission bbMission)
  {
    super(priority,  storage);

    this.bbMission = bbMission;
  }

  public void notifyReader()
  {
    notify();
  }

  @Override
  public void run()
  {
    while(! bbMission.terminationPending())
    {
      if (bbMission.bufferEmpty())
      {
        wait();
      }
      else
      {
        System.out.println("" + bbMission.read());
        bbMission.notifyWriter();
        try{
          Thread.sleep(500);
        }
        catch(InterruptedException e) {}
      }
    }

  }

}
```