

## Lecture 21

The main topic of today's lecture will be the textbooks's "Circular Buffer". The lecture will be on the short side because time has to be set aside for the instructor evaluations.

The circular buffer is a synchronized program that is built on four classes. The first provides `HoldIntegerSynchronized` an array of five integers `sharedInt[]`. It also provides synchronized methods and a display buffer

```
public synchronized void setSharedInt(int val)
public synchronized int getSharedInt()
public void displayBuffer(JTextArea out, int buf[])
```

The details are:

```

import javax.swing.*;
import java.text.*;

public class HoldIntegerSynchronized
{
    private int sharedInt[] = {-1,-1,-1,-1,-1};
    private boolean writeable = true;
    private boolean readable = false;

    private int readLoc = 0, writeLoc = 0;
    private JTextArea output;

    public HoldIntegerSynchronized( JTextArea o)
    {
        output = o;
    }

    public synchronized void setSharedInt(int val)
    {
        while (!writeable)
        {
            try
            {
                output.append("WAITING TO PRODUCE "+ val);
                wait();
            }
            catch( InterruptedException e)
            {
                System.err.println(e.toString());
            }
        } //end while()

        sharedInt[writeLoc ] = val;
        readable = true;

        output.append("\nProduced " + val +
                    " into cell " + writeLoc);

        writeLoc = (writeLoc + 1)%5;

        output.append("\twrite " + writeLoc +
                    "\tread " + readLoc);

        displayBuffer(output, sharedInt);

        if (writeLoc == readLoc)
        {
            writeable = false;
            output.append( "\nBUFFER FULL");
        }

        notify();
    } // end setSharedInt(int val)
}

```

```

public synchronized int getSharedInt()
{
    int val;

    while(!readable)
    {
        try
        {
            output.append(" WAITING TO CONSUME");
            wait();
        }
        catch(InterruptedException e)
        {
            System.err.println(e.toString());
        }
    }
} // end while

writeable = true;
val = sharedInt[readLoc];

output.append("\nConsumed " + val +
              " from cell " + readLoc);

readLoc = (readLoc+1)%5;

output.append( "\twrite " + writeLoc +
              "\tread " + readLoc);

displayBuffer(output, sharedInt);

if (readLoc == writeLoc)
{
    readable = false;
    output.append( "\nBUFFER EMPTY" );
}

notify();
return val;

} // end    getSharedInt()

public void displayBuffer(JTextArea out, int buf[])
{
    DecimalFormat formatNumber =
    new DecimalFormat(" #;-#");
    output.append("\tbuffer: "    );

    for (int i = 0; i < buf.length; i++)
        out.append( " " + formatNumber.format(buf[i]));
} // end displayBuffer()

} // end class HoldIntegerSynchronized

```

Next, we have classes for producing and consuming integers:

```
import javax.swing.*;

public class ProduceInteger extends Thread
{
    private HoldIntegerSynchronized pHold;
    private JTextArea output;

    public ProduceInteger( HoldIntegerSynchronized h,
                          JTextArea o)
    {
        super("ProduceInteger");
        pHold = h;
        output = o;
    } // end ProduceInteger()

    public void run()
    {
        for ( int count = 1; count <= 10; count++)
        {
            try
            {
                Thread.sleep( (int)( 500*Math.random()));
            }
            catch(InterruptedException e)
            {
                System.err.println(e.toString());
            }

            pHold.setSharedInt(count);
        } // end for

        output.append("\n" + getName() +
                    " finished producing values" +
                    "\nTerminating " + getName() + "\n");
    } // run()
} // end class ProduceInteger
```

```

import javax.swing.*;

public class ConsumeInteger extends Thread
{
    private HoldIntegerSynchronized cHold;
    private JTextArea output;

    public ConsumeInteger(HoldIntegerSynchronized h,
                          JTextArea o)
    {
        super("ConsumeInteger");
        cHold = h;
        output = o;
    } // end ConsumeInteger()

    public void run()
    {
        int val, sum = 0;

        do
        {
            try
            {
                Thread.sleep( (int)(3000*Math.random()) );
            }
            catch(InterruptedException e)
            {
                System.err.println(e.toString());
            }

            val = cHold.getSharedInt();
            sum += val;
        } while (val != 10); //end do

        output.append("\n" + getName() +
                     " retrieved values totalling " + sum +
                     "\nTerminating " + getName() + "\n");
    } // end run()
} // end class ConsumeInteger

```

And, finally, the driver

```
import java.text.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SharedCell extends JFrame
{
    public SharedCell()
    {
        super("Demonstrating Thread Synchronization");
        JTextArea output = new JTextArea(20,30);

        getContentPane().add( new JScrollPane(output));
        setSize(500,500);
        show();

        HoldIntegerSynchronized h =
        new HoldIntegerSynchronized( output );

        ProduceInteger p = new ProduceInteger(h,output);
        ConsumeInteger c = new ConsumeInteger(h, output);

        p.start();
        c.start();
    } // end SharedCell()

    public static void main(String args[])
    {
        SharedCell app = new SharedCell();

        app.addWindowListener
        (
            new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
                }
            }
        );
    } // end main()
} // end class SharedCell
```

The output is:

```
Demonstrating Thread Synchronization
Produced 1 into cell 0      write 1      read 0      buffer: 1 -1 -1 -1 -1
Produced 2 into cell 1      write 2      read 0      buffer: 1 2 -1 -1 -1
Produced 3 into cell 2      write 3      read 0      buffer: 1 2 3 -1 -1
Produced 4 into cell 3      write 4      read 0      buffer: 1 2 3 4 -1
Produced 5 into cell 4      write 0      read 0      buffer: 1 2 3 4 5
BUFFER FULLWAITING TO PRODUCE 6
Consumed 1 from cell 0      write 0      read 1      buffer: 1 2 3 4 5
Produced 6 into cell 0      write 1      read 1      buffer: 6 2 3 4 5
BUFFER FULLWAITING TO PRODUCE 7
Consumed 2 from cell 1      write 1      read 2      buffer: 6 2 3 4 5
Produced 7 into cell 1      write 2      read 2      buffer: 6 7 3 4 5
BUFFER FULLWAITING TO PRODUCE 8
Consumed 3 from cell 2      write 2      read 3      buffer: 6 7 3 4 5
Produced 8 into cell 2      write 3      read 3      buffer: 6 7 8 4 5
BUFFER FULLWAITING TO PRODUCE 9
Consumed 4 from cell 3      write 3      read 4      buffer: 6 7 8 4 5
Produced 9 into cell 3      write 4      read 4      buffer: 6 7 8 9 5
BUFFER FULLWAITING TO PRODUCE 10
Consumed 5 from cell 4      write 4      read 0      buffer: 6 7 8 9 5
Produced 10 into cell 4     write 0      read 0      buffer: 6 7 8 9 10
BUFFER FULL
ProduceInteger finished producing values
Terminating ProduceInteger

Consumed 6 from cell 0      write 0      read 1      buffer: 6 7 8 9 10
Consumed 7 from cell 1      write 0      read 2      buffer: 6 7 8 9 10
Consumed 8 from cell 2      write 0      read 3      buffer: 6 7 8 9 10
Consumed 9 from cell 3      write 0      read 4      buffer: 6 7 8 9 10
Consumed 10 from cell 4     write 0      read 0      buffer: 6 7 8 9 10
BUFFER EMPTY
ConsumerInteger retrieved values totalling 55
Terminating ConsumerInteger
```