

```
public class ProducerConsumer
{
    public static void main(String[] args)
    {
        Thread producer, consumer;
        BoundedBuffer buffer;

        buffer = new BoundedBuffer();

        // Create the producer and consumer threads,
        // passing each thread a reference to the
        // shared BoundedBuffer object.
        producer = new Thread(new
                                Producer(buffer), "Producer");
        consumer = new Thread(new
                                Consumer(buffer), "Consumer");

        producer.start();
        consumer.start();
    }
}
```

```
class Producer implements Runnable {  
    BoundedBuffer buffer;  
  
    public Producer(BoundedBuffer buf){  
        buffer = buf;  
    }  
  
    public void run()  
    {  
        for(int i = 0; i < 10; i++) {  
            Integer item = new Integer(i);  
            System.out.println(  
                Thread.currentThread().getName() +  
                " produced " + item);  
        }  
    }  
}
```

```

synchronized(buffer) {
    while(!buffer.isWriteable()) {
        try {
            buffer.wait();
        } catch (InterruptedException e) {
            return;
        }
    }
    buffer.addLast(item);
    if(buffer.getCount()==1) // now readable
        buffer.notifyAll();
} // end synchronized block
try {
    Thread.sleep(500);
} catch (InterruptedException e) {}
}
}
}

```

```
class Consumer implements Runnable
{
    BoundedBuffer buffer;

    public Consumer(BoundedBuffer buf)
    {
        buffer = buf;
    }

    public void run()
    {
        for(int i = 0; i < 10; i++) {
            Object item;
```

```

synchronized(buffer) {
    while (!buffer.isReadable()) {
        try {
            wait();
        } catch (InterruptedException e) {
            return;
        }
    }
    item = buffer.removeFirst();
    if (buffer.getCount()==BoundedBuffer.SIZE-1)
        buffer.notifyAll(); // now writeable
}
System.out.println(
    Thread.currentThread().getName()
    + " consumed " + item);
try {
    Thread.sleep(1000);
} catch (InterruptedException e) {}
}
}
}

```

```
public class BoundedBuffer
{
    // a ring buffer is used to hold the data

    // buffer capacity
    public static final int SIZE = 5;
    private Object[] buffer = new Object[SIZE];
    private int inIndex = 0, outIndex = 0;
    private int count = 0;

    // If true, there is room for at least one
    // object in the buffer.
    private boolean writeable = true;

    // If true, there is at least one object
    // stored in the buffer.
    private boolean readable = false;
```

```
public boolean addLast(Object item)
{
    if(!writeable) return false;
    buffer[inIndex] = item;
    inIndex = (inIndex + 1) % SIZE;
    count++;
    if (count == SIZE)
        writeable = false; // now full
    else if (count == 1)
        // readable was false
        readable = true;
    return true;
}
```

```
public boolean isWriteable() {
    return writeable;
}
```

```
public Object removeFirst()
{
    Object item;
    if(!readable) return null;
    item = buffer[outIndex];
    outIndex = (outIndex + 1) % SIZE;
    count--;
    if (count == 0)
        readable = false; // now empty
    else if (count == SIZE-1)
        // writeable was false
        writeable = true;
    return item;
}
```

```
public boolean isReadable() {
    return readable;
}
```

```
public int getCount() {
    return count;
}
```

```
}
```