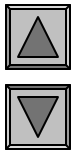
16.6 Producer/Consumer Relationship without Synchronization

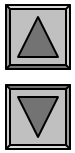
- Buffer
  - Shared memory region
- Producer thread
  - Generates data to add to buffer
  - Calls `wait` if consumer has not read previous message in buffer
  - Writes to empty buffer and calls `notify` for consumer
- Consumer thread
  - Reads data from buffer
  - Calls `wait` if buffer empty
- Synchronize threads to avoid corrupted data



Outline

Buffer.java

```
1  // Fig. 16.4: Buffer.java
2  // Buffer interface specifies methods called by Producer and Consumer.
3
4  public interface Buffer {
5     public void set( int value );  // place value into Buffer
6     public int get();              // return value from Buffer
7  }
```



Outline

```
1  // Fig. 16.5: Producer.java
2  // Producer's run method controls a thread that
3  // stores values from 1 to 4 in sharedLocation.
4
5  public class Producer extends Thread {
6     private Buffer sharedLocation; // reference to shared object
7
8     // constructor
9     public Producer( Buffer shared )
10    {
11        super( "Producer" );
12        sharedLocation = shared;
13    }
14
15    // store values from 1 to 4 in sharedLocation
16    public void run()
17    {
18       for ( int count = 1; count <= 4; count++ ) {
19
20          // sleep 0 to 3 seconds, then place value in Buffer
21          try {
22             Thread.sleep( ( int ) ( Math.random() * 3001 ) );
23             sharedLocation.set( count );
24          }
25
```

`Producer` extends `Thread`

Line 5

This is a shared object

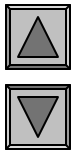
Line 6

Line 16

Lines 22-23

Method `run` is overridden

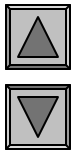
The thread goes to sleep, then the buffer is set



Outline

Producer.java

```
26          // if sleeping thread interrupted, print stack trace
27          catch ( InterruptedException exception ) {
28             exception.printStackTrace();
29          }
30
31       } // end for
32
33       System.err.println( getName() + " done producing." +
34          "\nTerminating " + getName() + ".");
35
36    } // end method run
37
38 } // end class Producer
```



Outline

```
1  // Fig. 16.6: Consumer.java
2  // Consumer's run method controls a thread that loops four
3  // times and reads a value from sharedLocation each time.
4
5  public class Consumer extends Thread {
6     private Buffer sharedLocation; // reference to shared object
7
8     // constructor
9     public Consumer( Buffer shared )
10    {
11       super( "Consumer" );
12       sharedLocation = shared;
13    }
14
15    // read sharedLocation's value four times and sum the values
16    public void run()
17    {
18       int sum = 0;
19
20       for ( int count = 1; count <= 4; count++ ) {
21
22          // sleep 0 to 3 seconds, read value from Buffer and add to sum
23          try {
24             Thread.sleep( ( int ) ( Math.random() * 3001 ) );
25             sum += sharedLocation.get();
26          }
27
```

`Consumer` extends `Thread`

Line 5

This is a shared object

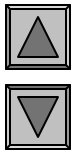
Line 6

Line 16

Lines 24-25

Method `run` is overridden

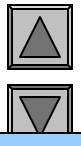
The thread goes to sleep, then the buffer is read



Outline

Consumer.java

```
28          // if sleeping thread interrupted, print stack trace
29          catch ( InterruptedException exception ) {
30             exception.printStackTrace();
31          }
32       }
33
34       System.err.println( getName() + " read values totaling: " + sum +
35          ".\nTerminating " + getName() + ".");
36
37    } // end method run
38
39 } // end class Consumer
```



Outline

```
1  // Fig. 16.7: UnsynchronizedBuffer.java
2  // UnsynchronizedBuffer represents a single shared integer.
3
4  public class UnsynchronizedBuffer implements Buffer {
5     private int buffer = -1; // shared by producer and consumer thr
6
7     // place value into buffer
8     public void set( int value )
9     {
10       System.err.println( Thread.currentThread().getName() +
11          " writes " + value );
12
13       buffer = value;
14    }
15
16    // return value from buffer
17    public int get()
18    {
19       System.err.println( Thread.currentThread().getName() +
20          " reads " + buffer );
21
22       return buffer;
23    }
24
25 } // end class UnsynchronizedBuffer
```

This class implements the `Buffer` interface

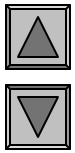
The data is a single integer

This method sets the value in the buffer

Lines 8 and 13

Lines 17 and 22

This method reads the value in the buffer



Outline

SharedBufferTest.java

Line 9

Create a `Buffer` object

Lines 12-13

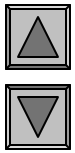
Create a `Producer` and a `Consumer`

Start the `Producer` and `Consumer` threads

```
1  // Fig. 16.8: SharedBufferTest.java
2  // SharedBufferTest creates producer and consumer threads.
3
4  public class SharedBufferTest {
5
6      public static void main( String [] args )
7      {
8          // create shared object used by threads
9          Buffer sharedLocation = new UnsynchronizedBuffer();
10
11         // create producer and consumer objects
12         Producer producer = new Producer( sharedLocation );
13         Consumer consumer = new Consumer( sharedLocation );
14
15         producer.start();  // start producer thread
16         consumer.start();  // start consumer thread
17
18     } // end main
19
20 } // end class SharedCell
```

```
Consumer reads -1
Producer writes 1
Consumer reads 1
Consumer reads 1
Consumer reads 1
Consumer read values totaling: 2.
Terminating Consumer.
Producer writes 2
Producer writes 3
Producer writes 4
Producer done producing.
Terminating Producer.
```

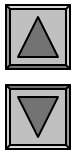
```
Producer writes 1
Producer writes 2
Consumer reads 2
Producer writes 3
Consumer reads 3
Producer writes 4
Producer done producing.
Terminating Producer.
Consumer reads 4
Consumer reads 4
Consumer read values totaling: 13.
Terminating Consumer.
```



Outline

SharedBufferTest.java

```
Producer writes 1
Consumer reads 1
Producer writes 2
Consumer reads 2
Producer writes 3
Consumer reads 3
Producer writes 4
Producer done producing.
Terminating Producer.
Consumer reads 4
Consumer read values totaling: 10.
Terminating Consumer.
```



Outline

SharedBufferTest.java

16.7 Producer/Consumer Relationship with Synchronization

- Synchronize threads to ensure correct data

```
1  // Fig. 16.9: SynchronizedBuffer.java
2  // SynchronizedBuffer synchronizes access to a single shared int
3
4  public class SynchronizedBuffer implements Buffer {
5     private int buffer = -1; // shared by producer and consumer th
6     private int occupiedBufferCount = 0; // count of occupied buf
7
8     // place value into buffer
9     public synchronized void set( int value )
10    {
11       // for output purposes, get name of thread that called this method
12       String name = Thread.currentThread().getName();
13
14       // while there are no empty locations, place thread in waiting state
15       while ( occupiedBufferCount == 1 ) {
16
17          // output thread information and buffer information, th
18          try {
19             System.err.println( name + " tries to write." );
20             displayState( "Buffer full. " + name + " waits." );
21             wait();
22          }
23
24          // if waiting thread interrupted, print stack trace
25          catch ( InterruptedException exception ) {
26             exception.printStackTrace();
27          }
```

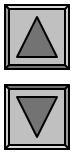
This class implements the `Buffer` interface

Remember the number of filled spaces

Method `set` is declared `synchronized`

Get the name of the thread

Wait while the buffer is filled



Outline

```
28
29       } // end while
30
31       buffer = value; // set new buffer value
32
33       // indicate producer cannot store another value
34       // until consumer retrieves current buffer value
35       ++occupiedBufferCount;
36
37       displayState( name + " writes " + buffer );
38
39       notify(); // tell waiting thread to enter ready state
40
41    } // end method set; releases lock on SynchronizedBuffer
42
43    // return value from buffer
44    public synchronized int get()
45    {
46       // for output purposes, get name of thread that called this method
47       String name = Thread.currentThread().getName();
48
```

Write to the buffer

hronizedBuf
fer.java

Increment the buffer count

Line 35

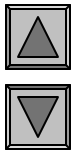
Alert a waiting thread

Line 39

Line 44

Method `get` is declared `synchronized`

Get the name of the thread



Outline

```
49       // while no data to read, place thread in waiting state
50       while ( occupiedBufferCount == 0 ) {
51
52          // output thread information and buffer information, then wait
53          try {
54             System.err.println( name + " tries to read." );
55             displayState( "Buffer empty. " + name + " waits." );
56             wait();
57          }
58
59          // if waiting thread interrupted, print stack trace
60          catch ( InterruptedException exception ) {
61             exception.printStackTrace();
62          }
63
64       } // end while
65
66       // indicate that producer can store another value
67       // because consumer just retrieved buffer value
68       --occupiedBufferCount;
69
70       displayState( name + " reads " + buffer );
71
72       notify(); // tell waiting thread to become ready to execute
73
74       return buffer;
```

Wait while the buffer is empty

Lines 50 and 56

Line 68

Line 72

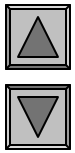
Line 74

Decrement the buffer count

Alert a waiting thread

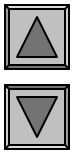
Return the buffer

```
75
76     } // end method get; releases lock on SynchronizedBuffer
77
78     // display current operation and buffer state
79     public void displayState( String operation )
80     {
81        StringBuffer outputLine = new StringBuffer( operation );
82        outputLine.setLength( 40 );
83        outputLine.append( buffer + "\t\t" + occupiedBufferCount );
84        System.err.println( outputLine );
85        System.err.println();
86     }
87
88  } // end class SynchronizedBuffer
```



Outline

SynchronizedBuffer.java



Outline

SharedBufferTest2.java

Create a `Buffer` object
Line 11

Line 19

Lines 22-23

```
1  // Fig. 16.10: SharedBufferTest2.java
2  // SharedBufferTest2creates producer and consumer threads.
3
4  public class SharedBufferTest2 {
5
6     public static void main( String [] args )
7     {
8        // create shared object used by threads; we use a SynchronizedBuffer
9        // reference rather than a Buffer reference so we can invoke
10       // SynchronizedBuffer method displayState from main
11       SynchronizedBuffer sharedLocation = new SynchronizedBuffer();
12
13       // Display column heads for output
14       StringBuffer columnHeads = new StringBuffer( "Operation" );
15       columnHeads.setLength( 40 );
16       columnHeads.append( "Buffer\t\tOccupied Count" );
17       System.err.println( columnHeads );
18       System.err.println();
19       sharedLocation.displayState( "Initial State" );
20
21       // create producer and consumer objects
22       Producer producer = new Producer( sharedLocation );
23       Consumer consumer = new Consumer( sharedLocation );
24
```

Output initial state

Create a `Producer` and a `Consumer`

```
25         producer.start();  // start producer thread
26         consumer.start();  // start consumer thread
27
28      } // end main
29
30   } // end class SharedBufferTest2
```

```
Operation                          Buffer          Occupied Count

Initial State                      -1              0

Consumer tries to read.
Buffer empty. Consumer waits.      -1              0

Producer writes 1                  1               1

Consumer reads 1                   1               0

Consumer tries to read.
Buffer empty. Consumer waits.      1               0

Producer writes 2                  2               1

Consumer reads 2                   2               0

Producer writes 3                  3               1
```

Outline

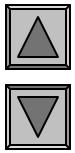
Start the `Producer` and `Consumer` threads

`SharedBufferTest2.java`

Lines 25-26

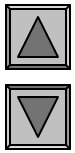
```
Consumer reads 3                        3               0

Consumer tries to read.
Buffer empty. Consumer waits.           3               0

Producer writes 4                       4               1

Consumer reads 4                        4               0
Producer done producing.
Terminating Producer.

Consumer read values totaling: 10.
Terminating Consumer.
```

```
Operation                               Buffer          Occupied Count

Initial State                           -1              0

Consumer tries to read.
Buffer empty. Consumer waits.           -1              0

Producer writes 1                       1               1

Consumer reads 1                        1               0

Producer writes 2                       2               1
```



Outline

SharedBufferTest2.java



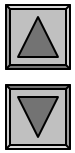
Outline

SharedBufferTest2.java

```
Producer tries to write.
Buffer full. Producer waits.            2               1

Consumer reads 2                        2               0

Producer writes 3                       3               1

Consumer reads 3                        3               0

Producer writes 4                       4               1

Producer done producing.
Terminating Producer.
Consumer reads 4                        4               0

Consumer read values totaling: 10.
Terminating Consumer.
```

```
Operation                               Buffer          Occupied Count

Initial State                           -1              0

Producer writes 1                       1               1

Consumer reads 1                        1               0

Producer writes 2                       2               1
```

```
Consumer reads 2                    2               0

Producer writes 3                   3               1

Consumer reads 3                    3               0

Producer writes 4                   4               1

Producer done producing.
Terminating Producer.
Consumer reads 4                    4               0

Consumer read values totaling: 10.
Terminating Consumer.
```



Outline

SharedBufferTest2.java