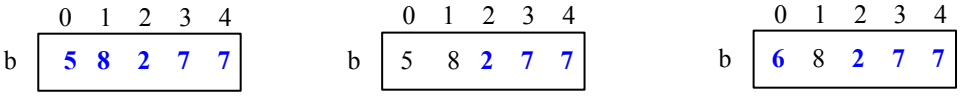
A bounded buffer

We implement a "bounded buffer" —telling you what it is— on the next page. On this page, we show how to implement a queue in an array.

Suppose we want to maintain a queue in an array b of size 5. Suppose we have added 5 values to it so that the queue contains (5, 8, 2, 7, 7) as shown to the left below. Now suppose the first two values of the queue are removed, so that the queue contains (2, 7, 7). The array is as shown in the middle diagram below. We don't change array elements when removing values from the queue. Instead, we just keep track of where the queue values start and end. In this case, the queue values are now in b[2..4], and they are shown bolded in blue.

Now consider adding the value 6 to the queue. It is supposed to follow element b[4]. Where do we put it? Instead of moving array elements around, we use wrap-around: place the 6 in b[0], so the array now looks as shown in the right diagram. The queue values (2, 7, 7, 6) are in b[2], b[3], b[4], and b[0], and b[1] is empty.



Below, we present class ArrayQueue<E>, which implements a queue in an array. Given the idea suggested above, it should be easy to understand the class invariant and to check that each method is correct. All methods take constant time!

```
/** An instance implements a queue of bounded size in an array */
public class ArrayQueue<E> {
   private E[] b;    // The n elements of the queue are in
   private int n;    // b[h], b[(h+1) % n], ... b[(h+n-1) % n]
   private int h;    // 0 <= h < b.length

   /** Constructor: a empty queue of maximum size s. */
   public ArrayQueue(int s) {
      b= (E[])new Object[s];
   }

   /** Return the size of the queue. */
   public int size() { return n; }

   /** = "queue is empty" */
   public boolean isEmpty() { return n == 0; }

   /** = "queue is full */
   public boolean isFull() { return n == b.length; }

   /** Throw RuntimeException if queue is full. Otherwise, add e to the queue. */
   public void put(E e) {
      if (n == b.length) throw new RuntimeException("queue full");
      b[(h+n) % b.length]= e;   n= n+1;
   }

   /** Throw a RuntimeException if queue empty. Otherwise, return first element */
   public E peek() {
      if (n == 0) throw new RuntimeException("queue empty");
      return b[h];
   }

   /** Throw a RuntimeException if the queue is empty.
    *  Otherwise, take head of queue off queue and return it. */
   public E take() {
      if (n == 0) throw new RuntimeException("queue empty");
      E e= b[h];   h= (h+1) % b.length;   n= n-1;    return e;
   }
}
```

A bounded buffer

Consider a bakery, with a shelf where bakers can put loaves of baked bread and customers can take them off. That shelf is an example of a *bounded buffer*. A buffer is a place to store items that may be put in and taken out at different rates. It's *bounded* because the shelf can hold only so many loaves of bread.

In computer science, a bounded buffer is usually a queue —items are placed at the end and taken out from the beginning. On the previous page, we gave a class that implemented a bounded queue in an array. We now use an object of that class to implement a bounded buffer in which many threads can act like producers (putting items into the buffer) or consumers (taking items out of the buffer).

Actually, class BoundedBuffer, given below, is *very* easy to write and to understand because of all the previous work we did on defining synchronization with methods wait() and notifyAll() and writing class ArrayQueue.

```
/** An instance maintains a bounded buffer of limited size */
public class BoundedBuffer<E> {
   ArrayQueue<E> aq;  // bounded buffer is implemented in aq

   /** Constructor: An empty buffer of max size n */
   public BoundedBuffer(int n) {
      aq= new ArrayQueue<E>(n);
   }

   /** Put v into the bounded buffer */
   public synchronized void produce(E v) {
      while (aq.isFull())
         try { wait(); }
         catch (InterruptedException e) {}
      aq.put(v);
      notifyAll();
   }

   /** Remove first element from bounded buffer and return it. */
   public synchronized E consume() {
      while (aq.isEmpty())
         try { wait(); }
         catch (InterruptedException e) {}
      E item= aq.take();
      notifyAll();
      return item;
   }
}
```