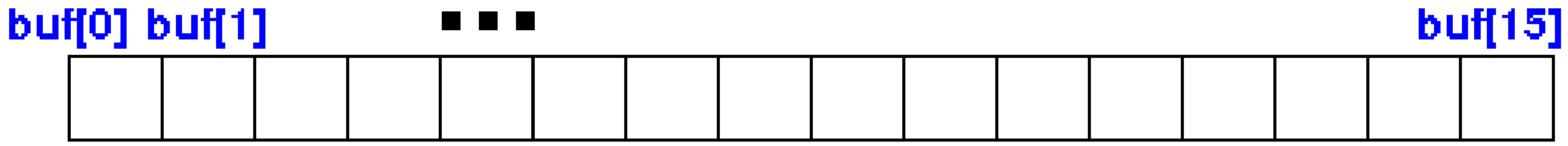
Array-Based Queue Implementation

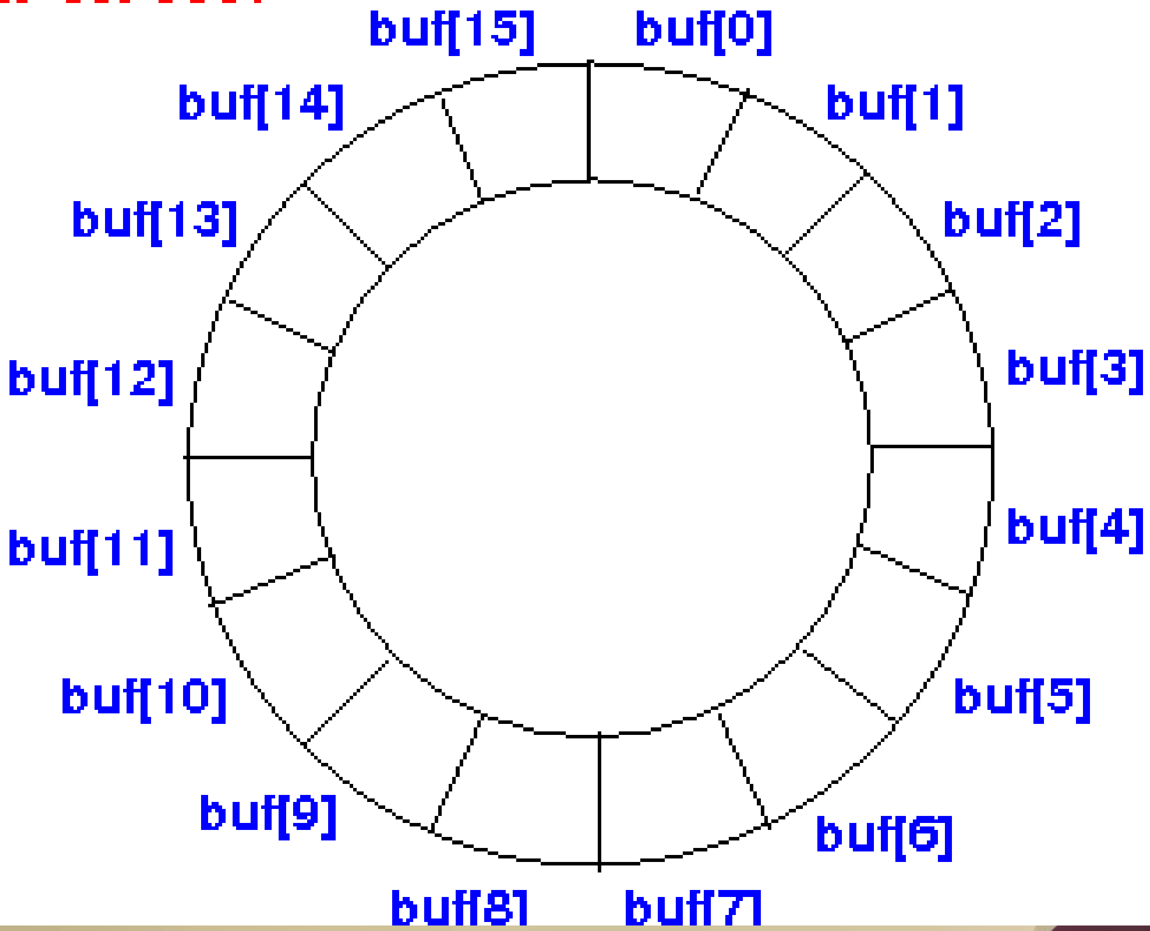
Circular Array

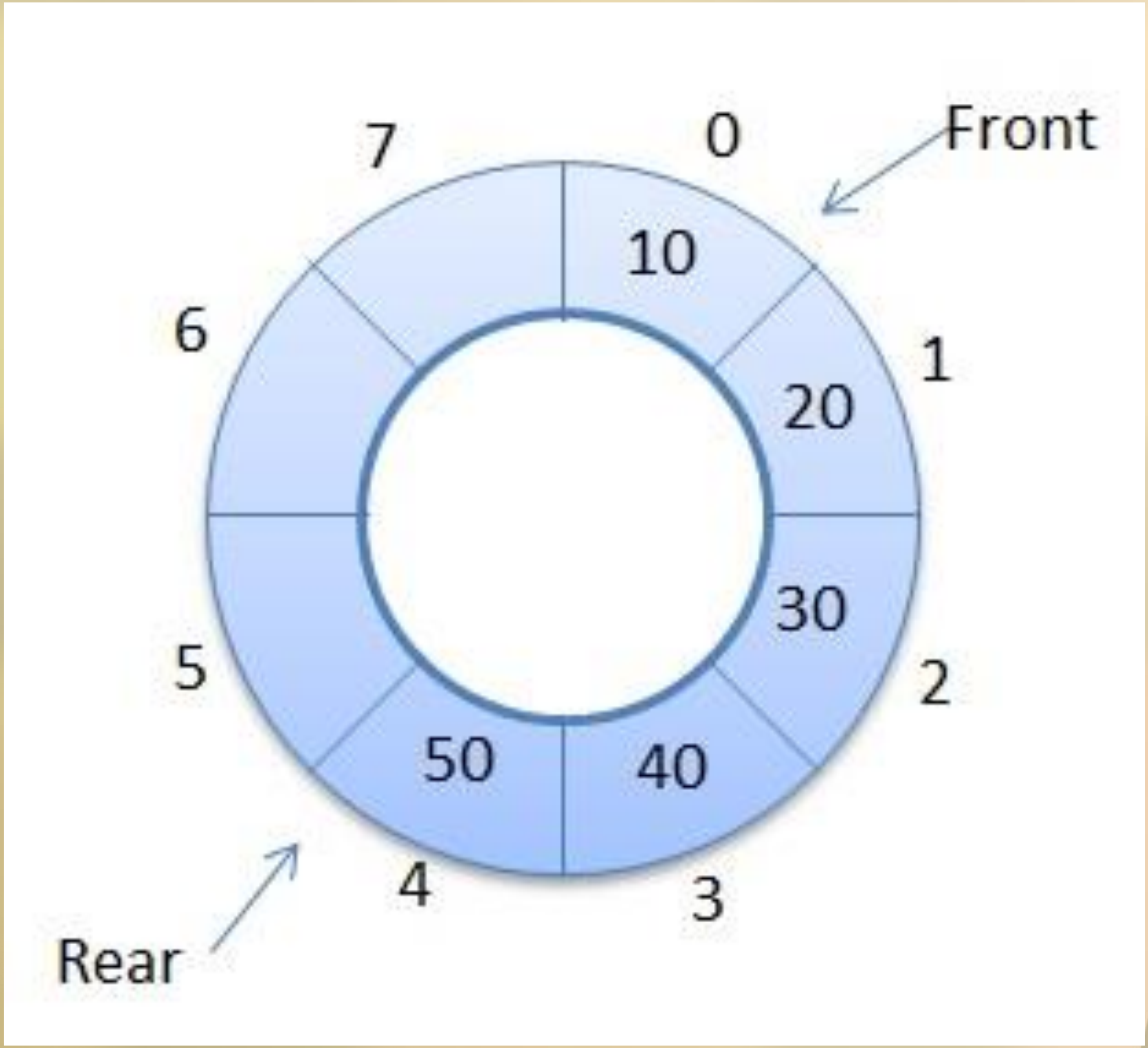
Data Structures and Abstractions with Java, 4e, Global Edition
Frank Carrano

Array:



Pretend array is a circle:





Front
0
1
2
3
4
5
6
7
10
20
30
40
50
Rear

Array-Based Implementation of a Queue: Circular Array

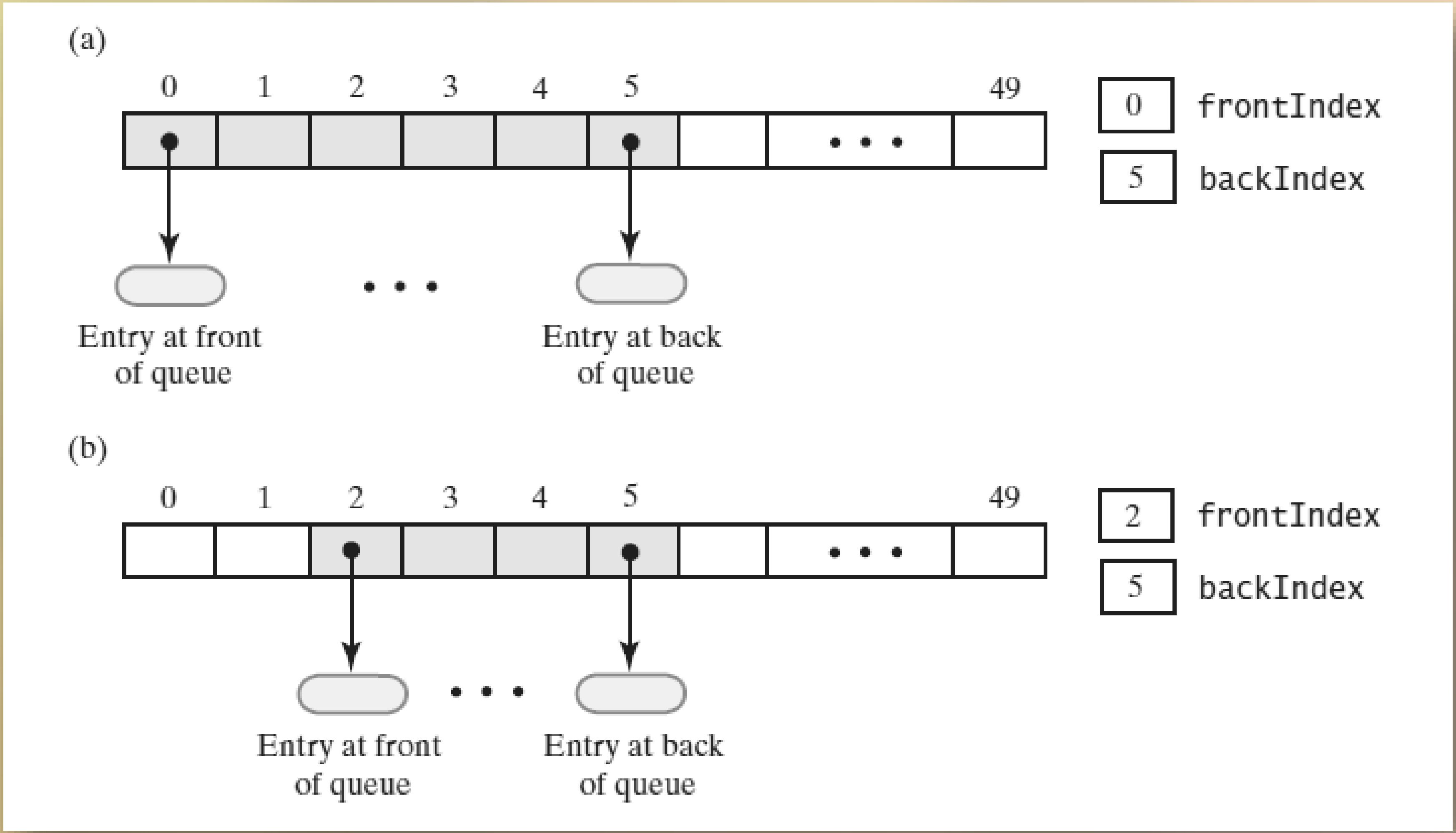


FIGURE 11-6 An array that represents a queue without moving any entries: (a) initially; (b) after removing the entry at the front twice;

Circular Array

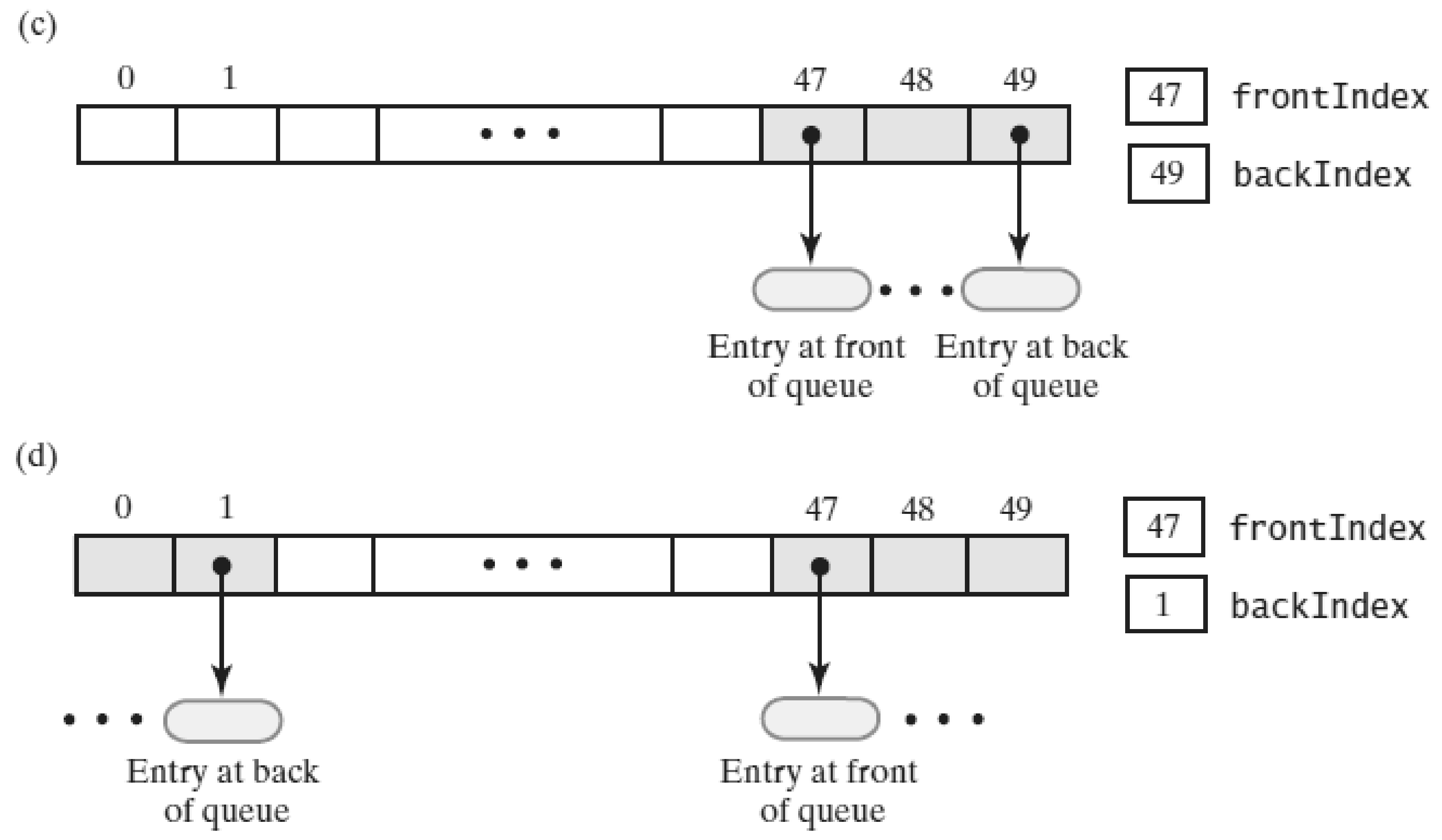


FIGURE 11-6 An array that represents a queue without moving any entries: (c) several more additions, removals; (d) after two additions that wrap around to beginning of array

Circular Array

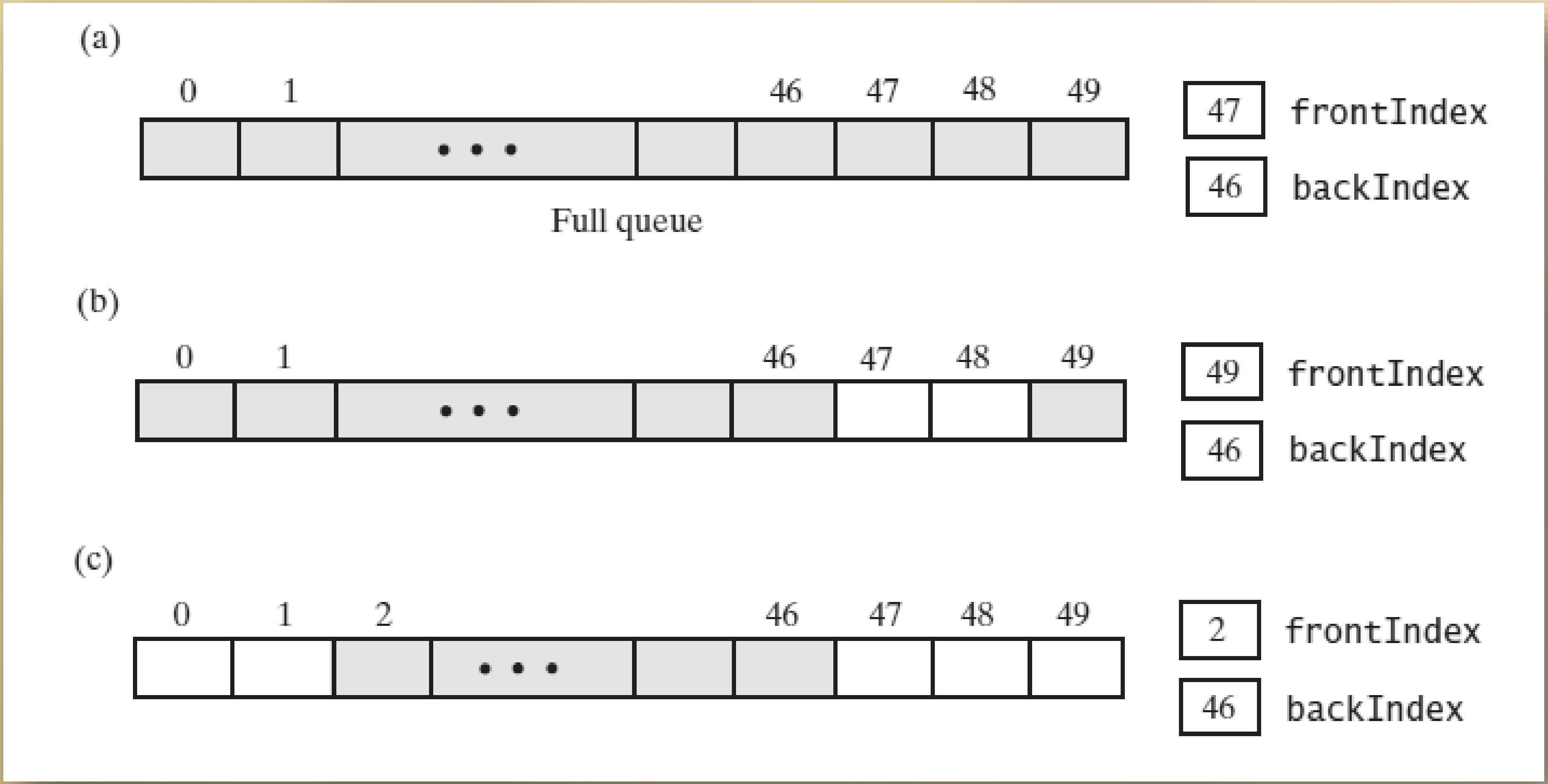


FIGURE 11-7 A circular array that represents a queue:
(a) when full; (b) after removing two entries; (c) after removing three more entries;

Circular Array

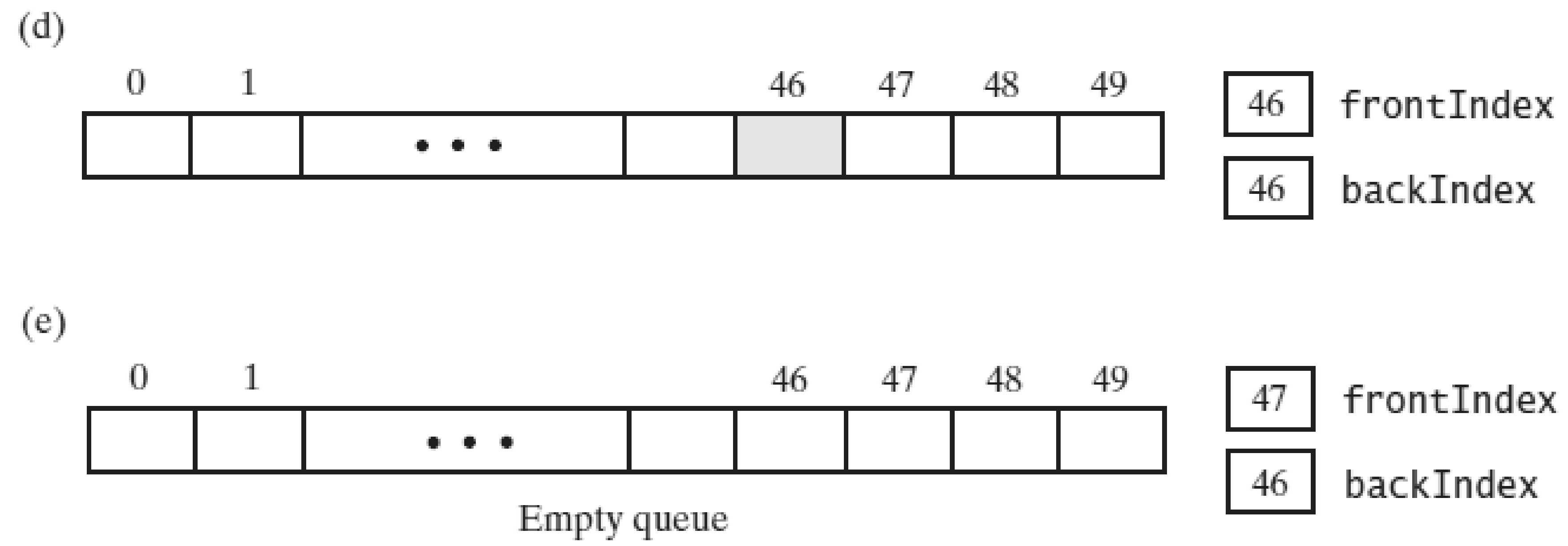


FIGURE 11-7 A circular array that represents a queue: (d) after removing all but one entry; (e) after removing the remaining entry

Circular Array with One Unused Location

Allows us to distinguish between empty and full queue by examining frontIndex and backIndex

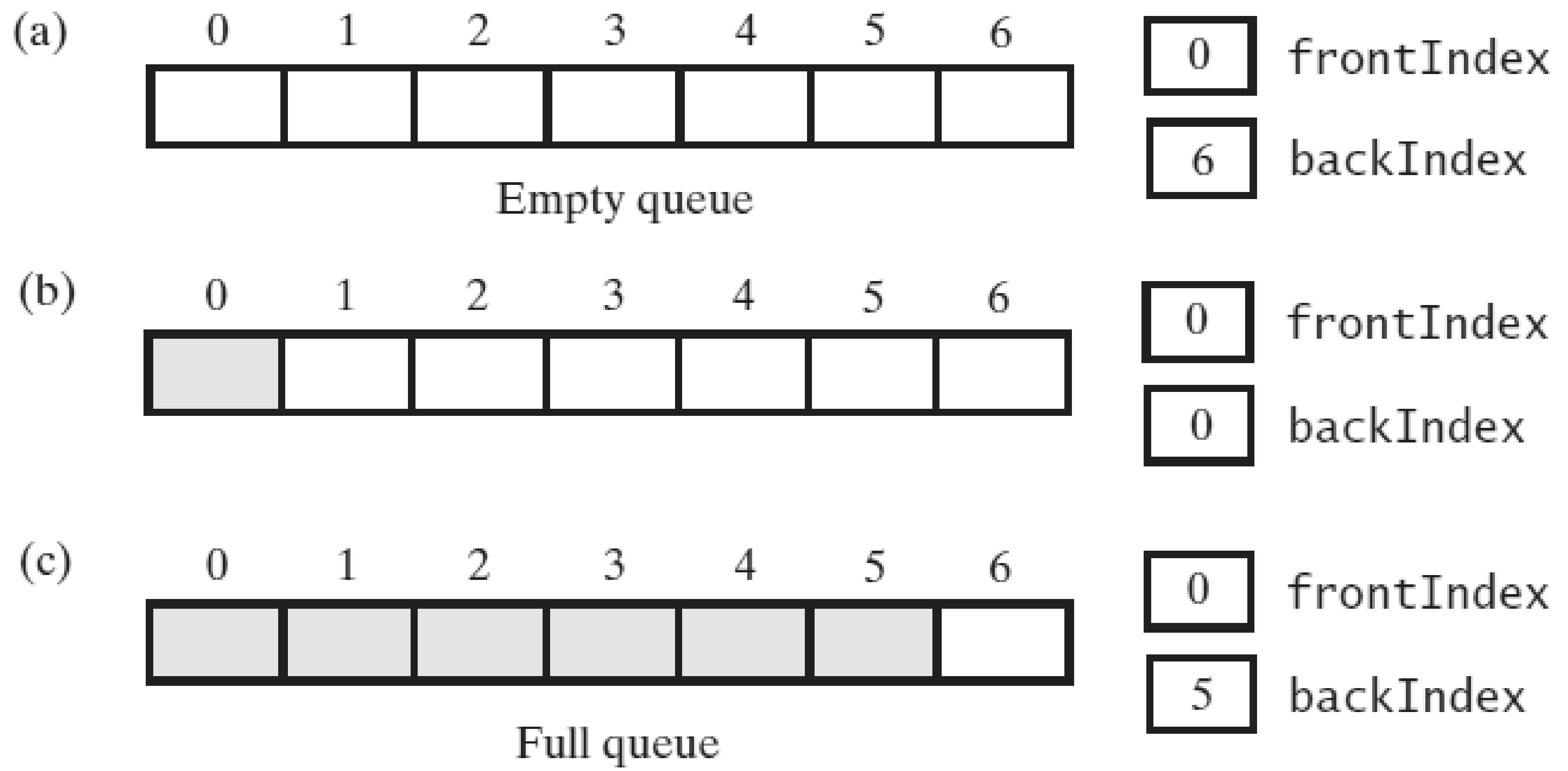


FIGURE 11-8 A seven-location circular array that contains at most six entries of a queue

Circular Array with One Unused Location

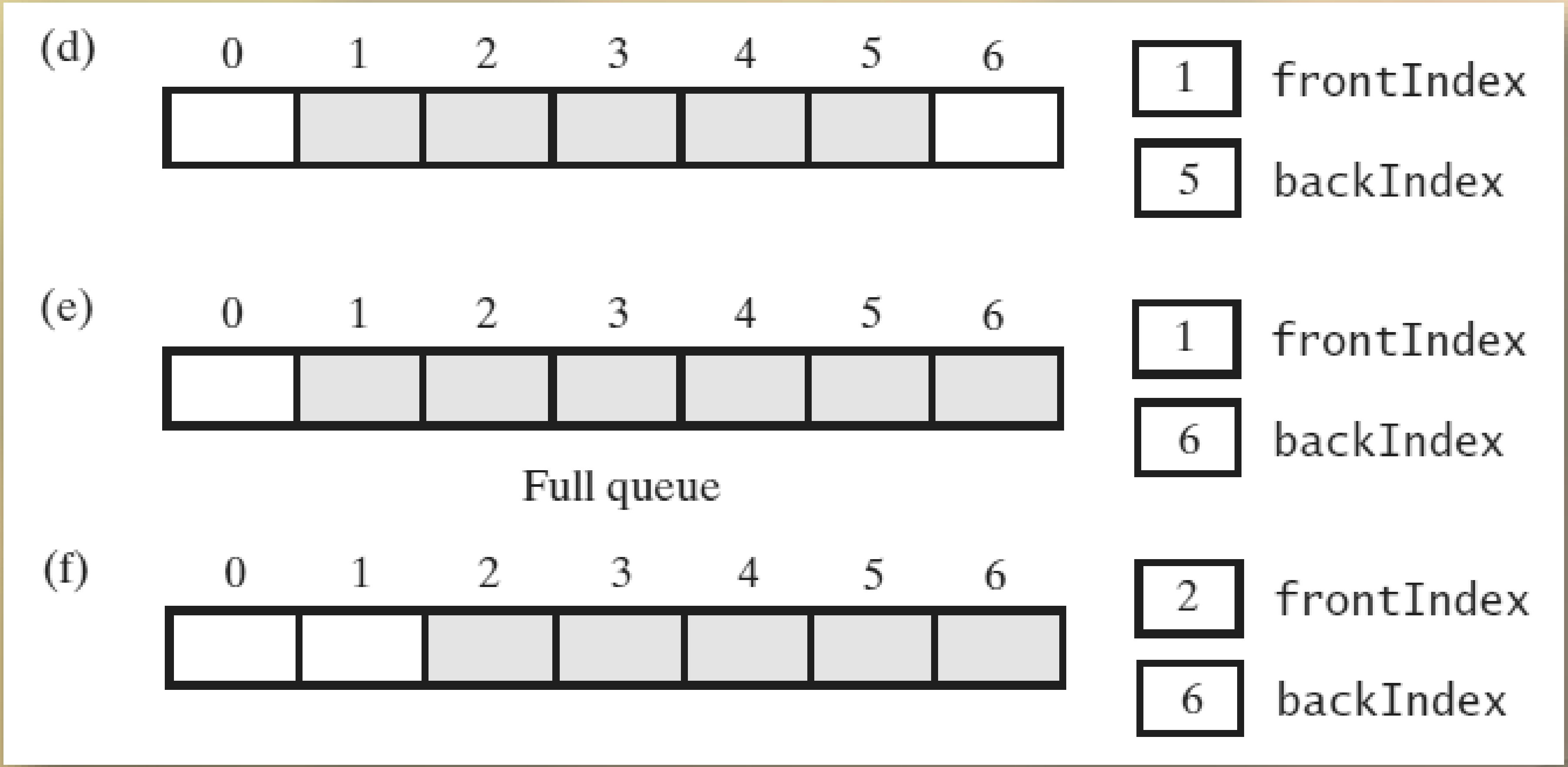
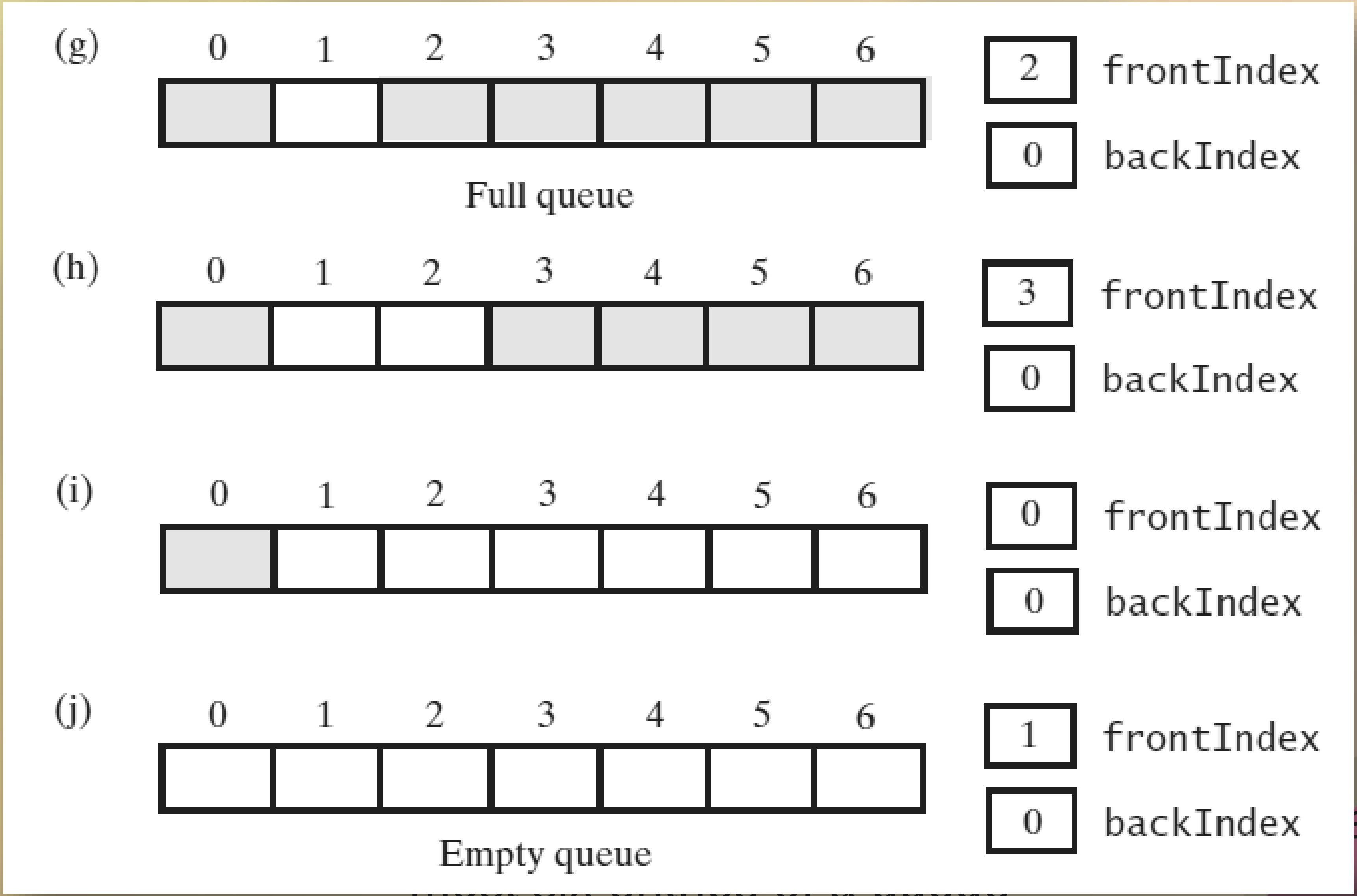


FIGURE 11-8 A seven-location circular array that contains at most six entries of a queue

Circular Array with One Unused Location



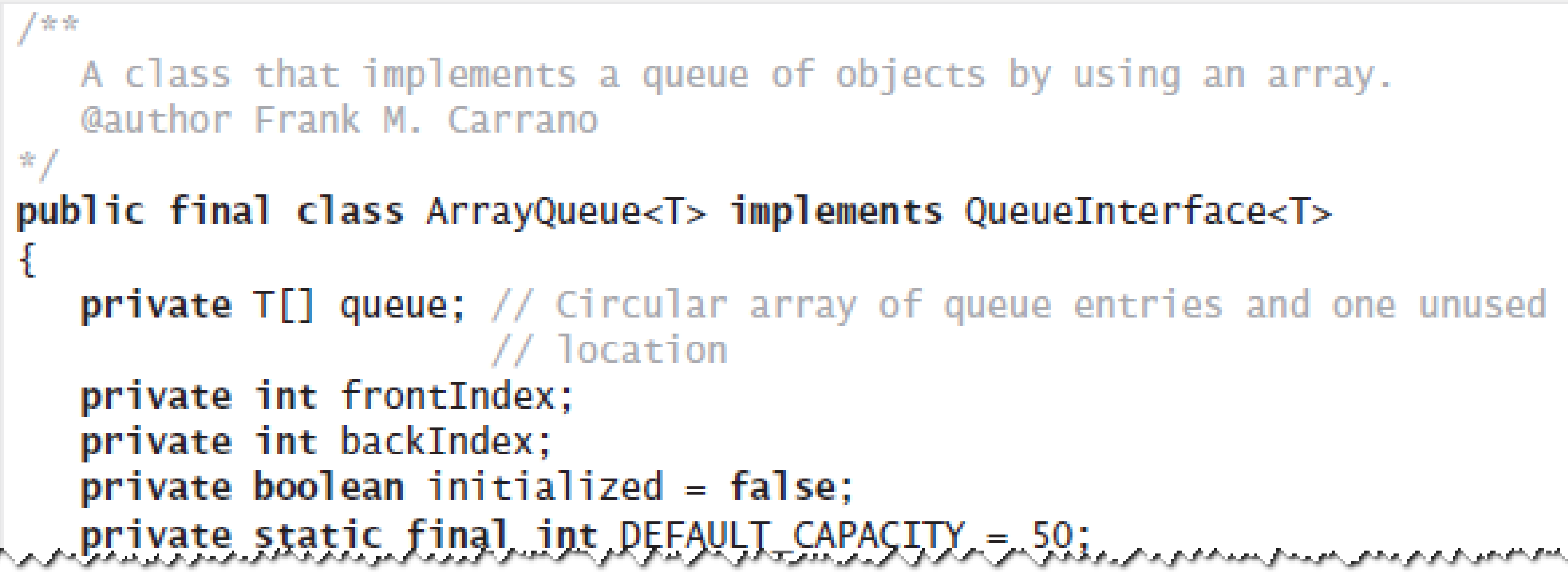
Circular Array with One Unused Location

When the array is full, the index of the unused location is 1 more than backIndex and 1 less than frontIndex.

frontIndex == (backIndex +2) % queue.length

When the array is empty,
frontIndex == (backIndex + 1) % queue.length

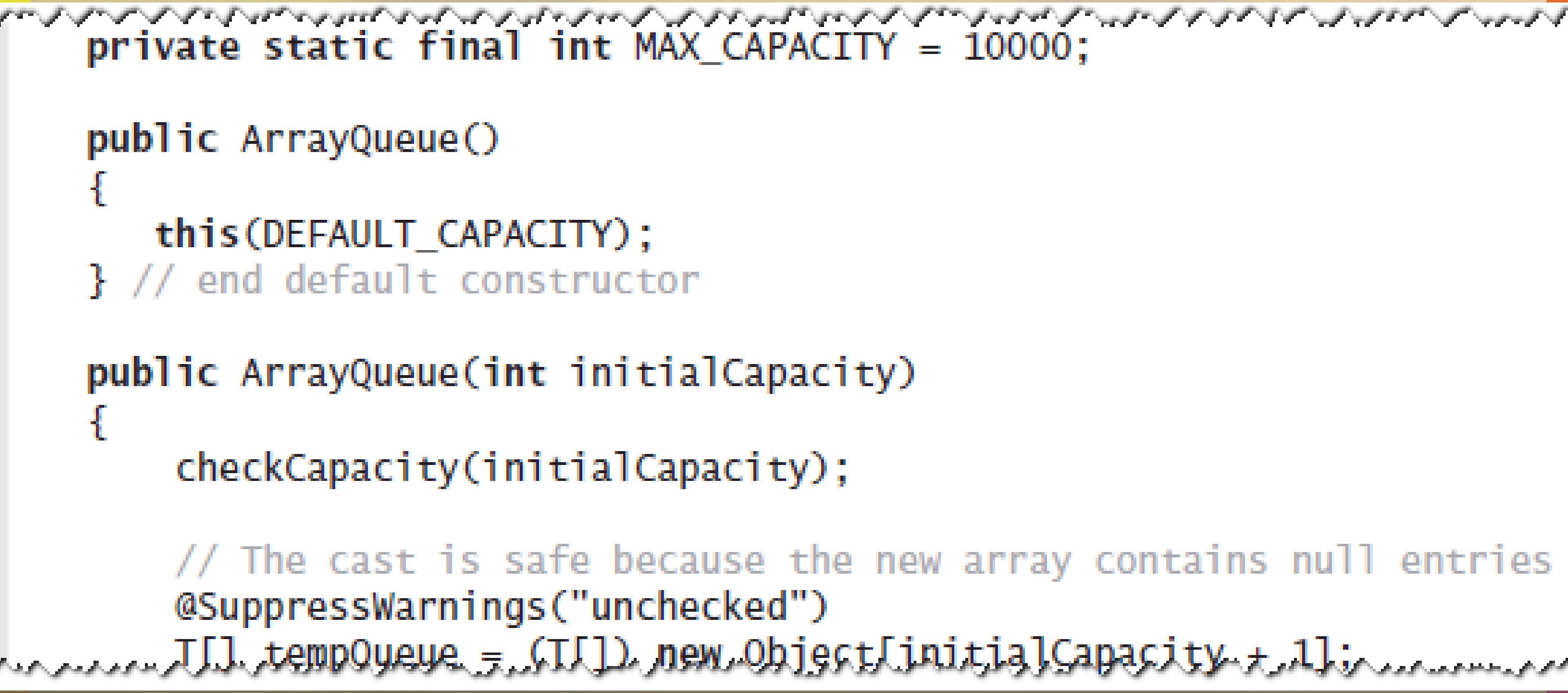
Circular Array with One Unused Location



```
/**
   A class that implements a queue of objects by using an array.
   @author Frank M. Carrano
*/
public final class ArrayQueue<T> implements QueueInterface<T>
{
   private T[] queue; // Circular array of queue entries and one unused
                      // location
   private int frontIndex;
   private int backIndex;
   private boolean initialized = false;
   private static final int DEFAULT_CAPACITY = 50;
```

LISTING 11-2 An outline of an array-based implementation of the ADT queue

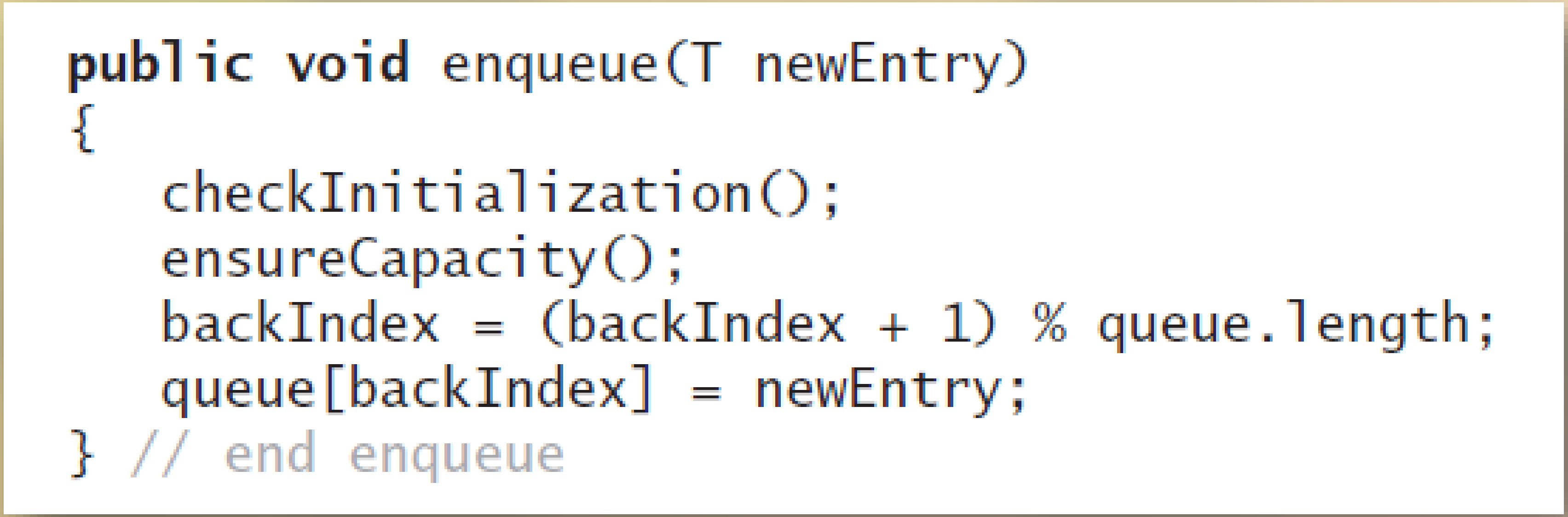
Circular Array with One Unused Location



```
private static final int MAX_CAPACITY = 10000;

public ArrayQueue()
{
   this(DEFAULT_CAPACITY);
} // end default constructor

public ArrayQueue(int initialCapacity)
{
    checkCapacity(initialCapacity);

    // The cast is safe because the new array contains null entries
    @SuppressWarnings("unchecked")
    T[] tempQueue = (T[]) new Object[initialCapacity + 1];
```

LISTING 11-2 An outline of an array-based implementation of the ADT queue

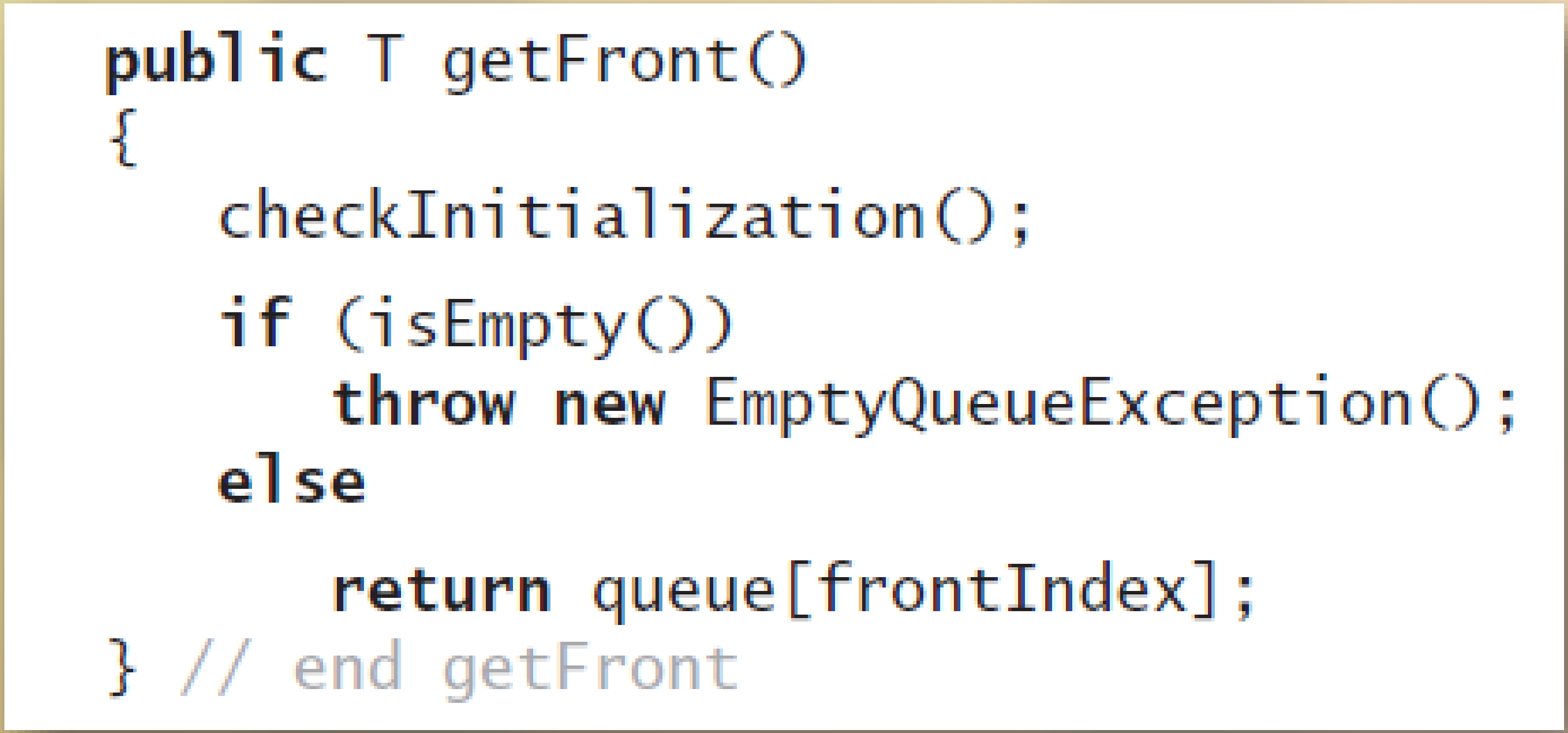
Circular Array with One Unused Location



```
public void enqueue(T newEntry)
{
   checkInitialization();
   ensureCapacity();
   backIndex = (backIndex + 1) % queue.length;
   queue[backIndex] = newEntry;
} // end enqueue
```

Adding to the back

Circular Array with One Unused Location



```
public T getFront()
{
   checkInitialization();
   if (isEmpty())
      throw new EmptyQueueException();
   else
      return queue[frontIndex];
} // end getFront
```

Retrieving the front entry

Circular Array with One Unused Location

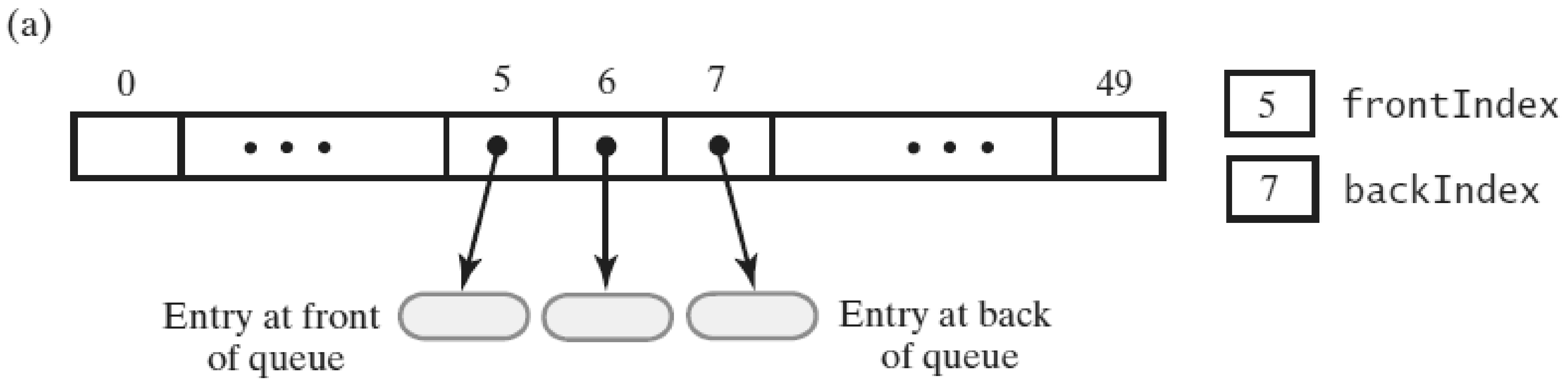


FIGURE 11-9 An array-based queue: (a) initially;

Circular Array with One Unused Location

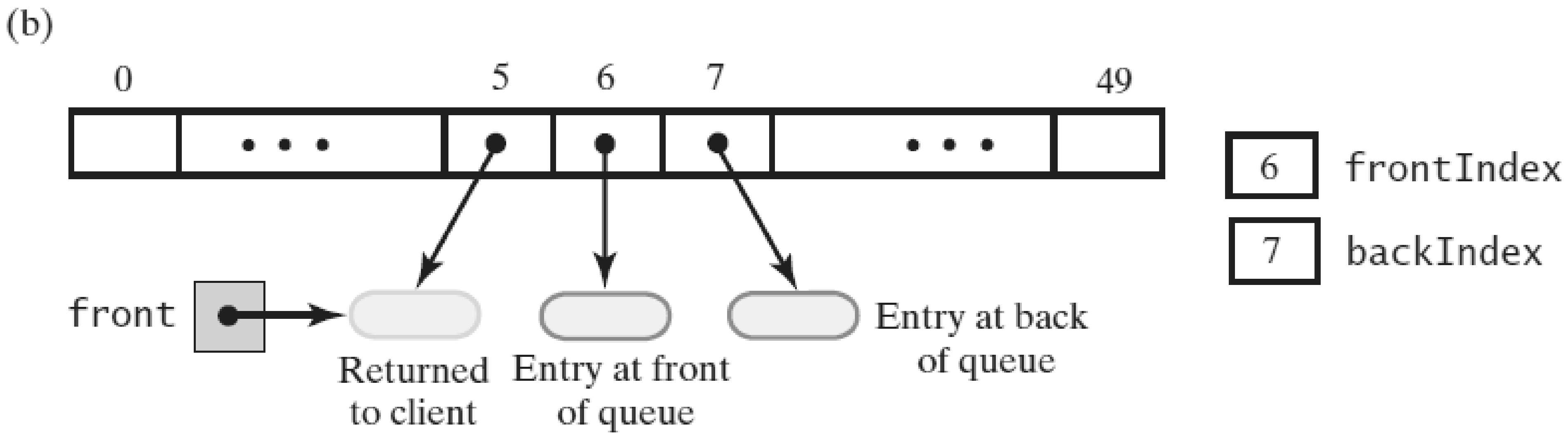


FIGURE 11-9 An array-based queue: (b) after removing its front entry by incrementing `frontIndex`;

Circular Array with One Unused Location

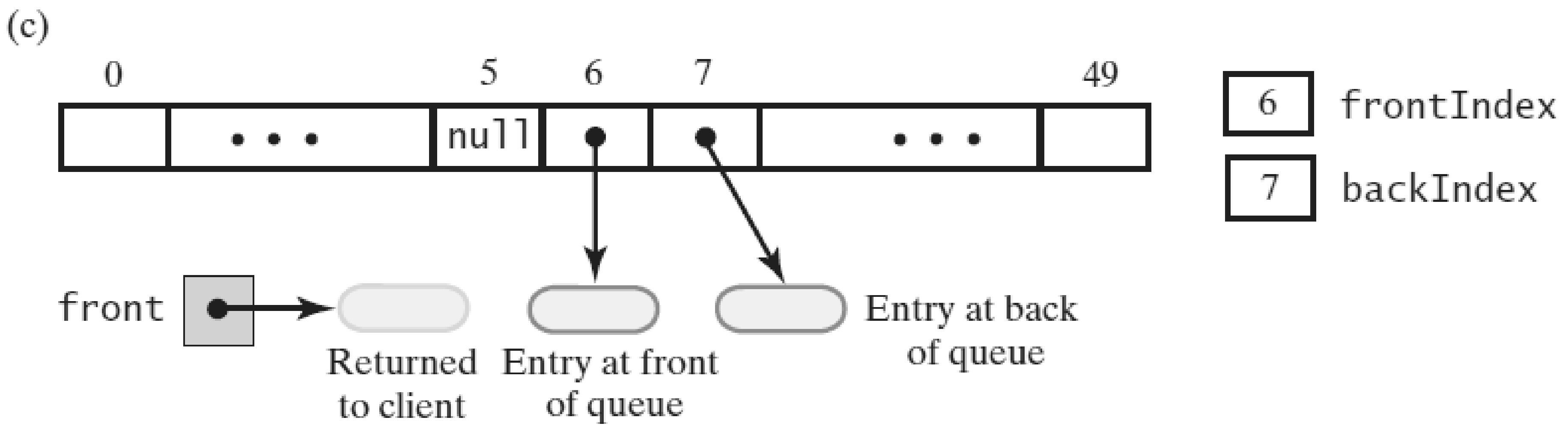
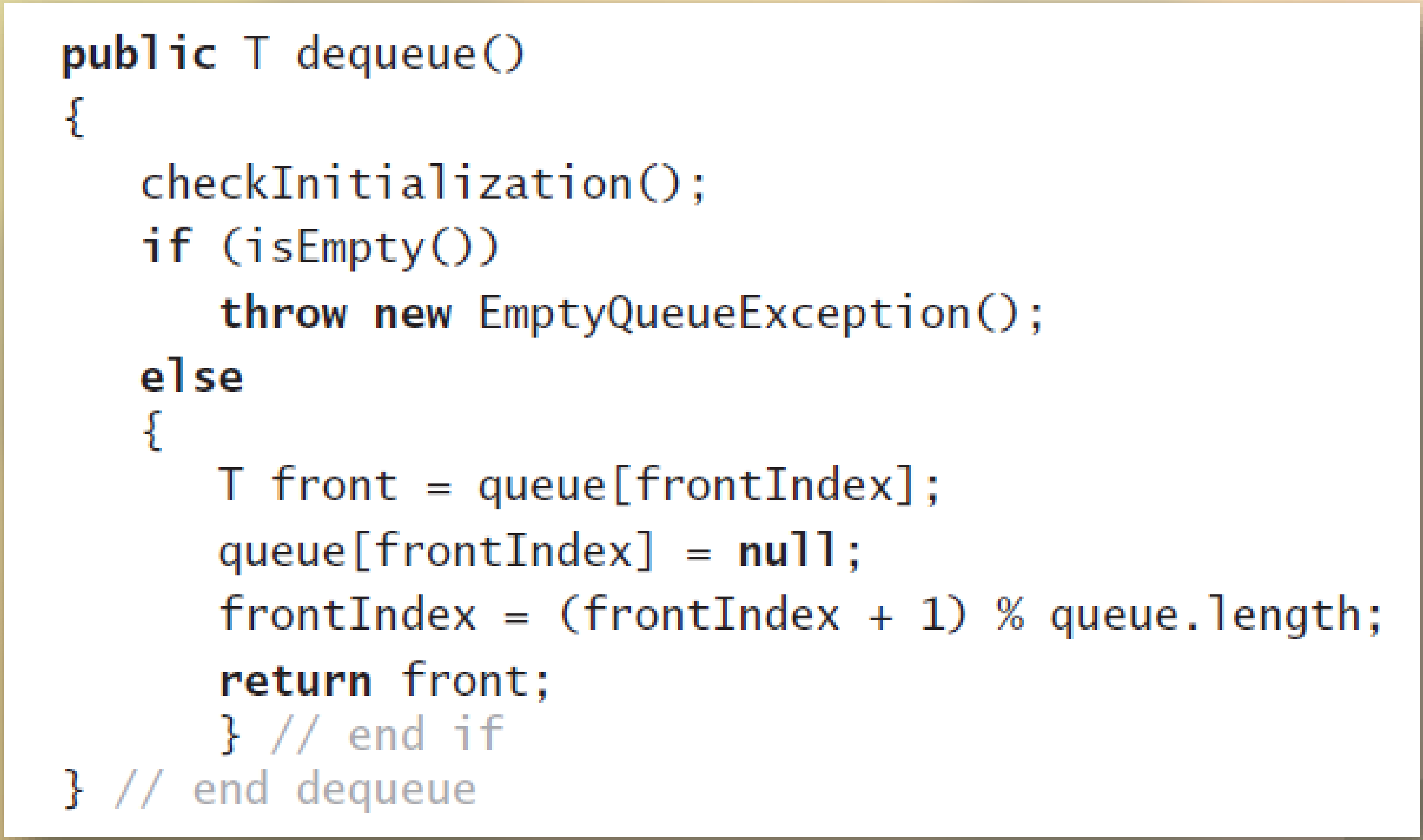


FIGURE 11-9 An array-based queue: (c) after removing its front entry by setting `queue[frontIndex]` to null and then incrementing `frontIndex`

Circular Array with One Unused Location



```
public T dequeue()
{
   checkInitialization();
   if (isEmpty())
      throw new EmptyQueueException();
   else
   {
      T front = queue[frontIndex];
      queue[frontIndex] = null;
      frontIndex = (frontIndex + 1) % queue.length;
      return front;
      } // end if
} // end dequeue
```

Implementation of **dequeue**

Circular Array with One Unused Location

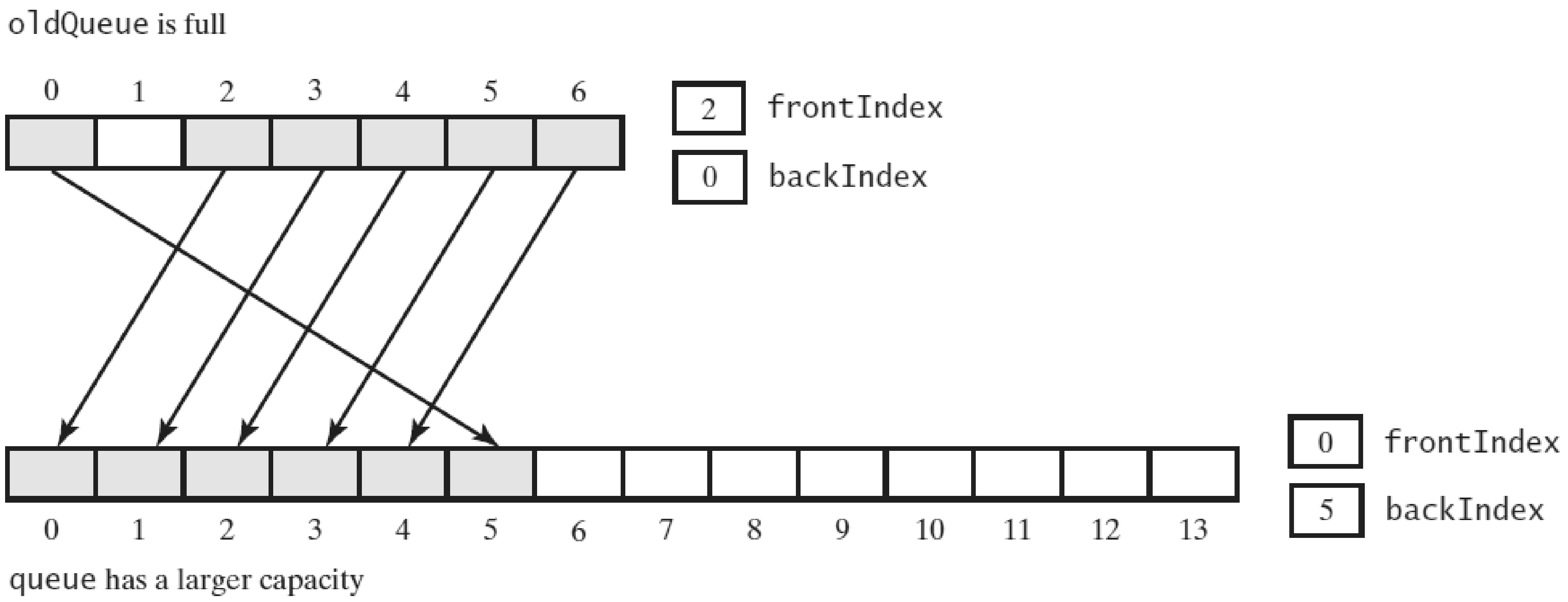
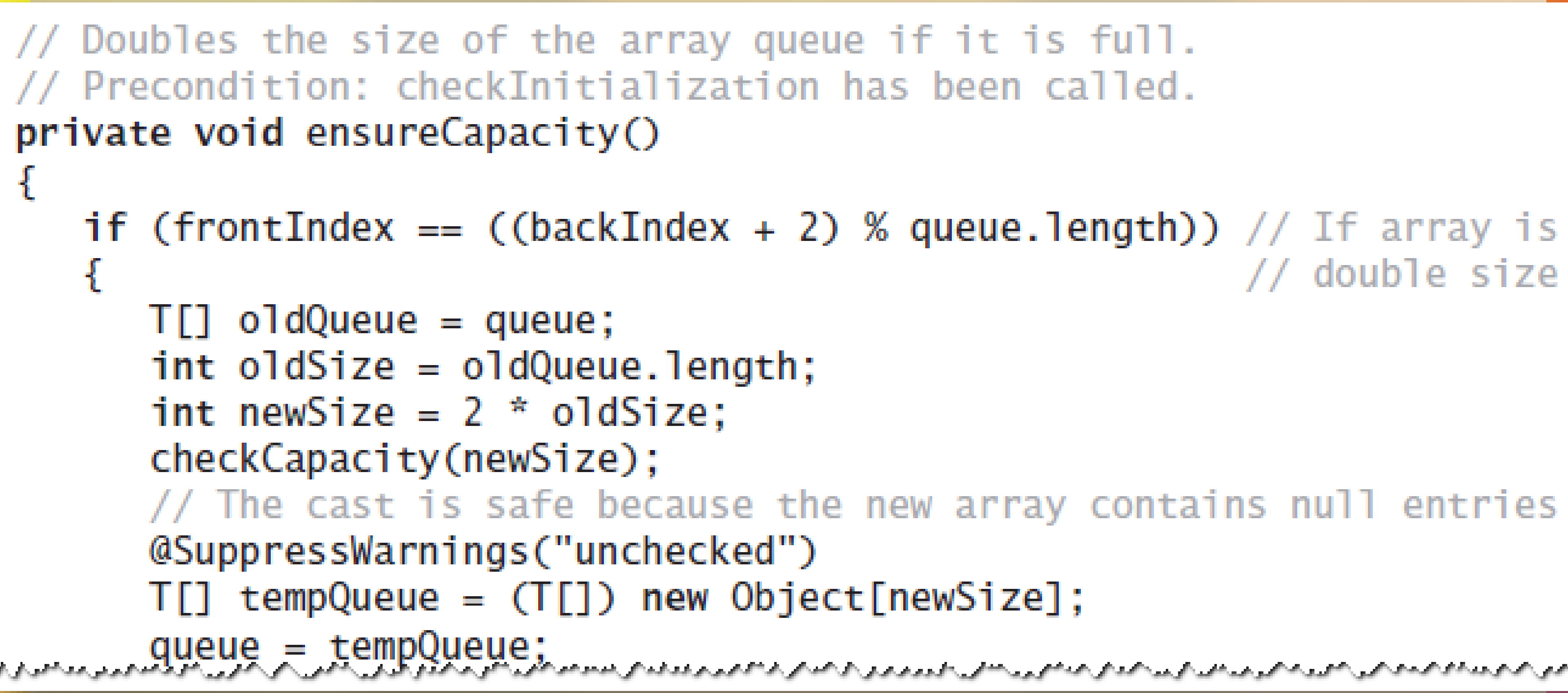


FIGURE 11-10 Doubling the size of an array-based queue

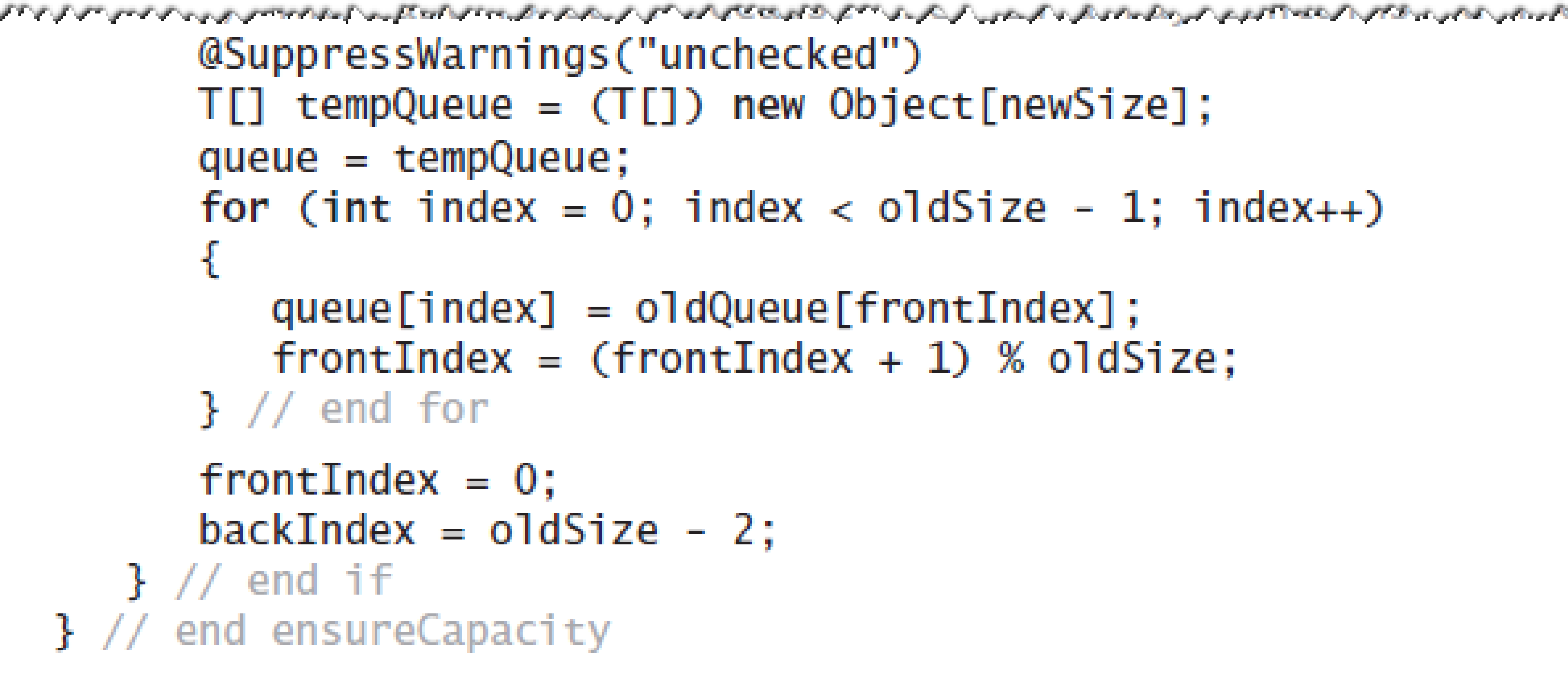
Circular Array with One Unused Location



```
// Doubles the size of the array queue if it is full.
// Precondition: checkInitialization has been called.
private void ensureCapacity()
{
   if (frontIndex == ((backIndex + 2) % queue.length)) // If array is
   {                                                    // double size
      T[] oldQueue = queue;
      int oldSize = oldQueue.length;
      int newSize = 2 * oldSize;
      checkCapacity(newSize);
      // The cast is safe because the new array contains null entries
      @SuppressWarnings("unchecked")
      T[] tempQueue = (T[]) new Object[newSize];
      queue = tempQueue;
```

Definition of `ensureCapacity`

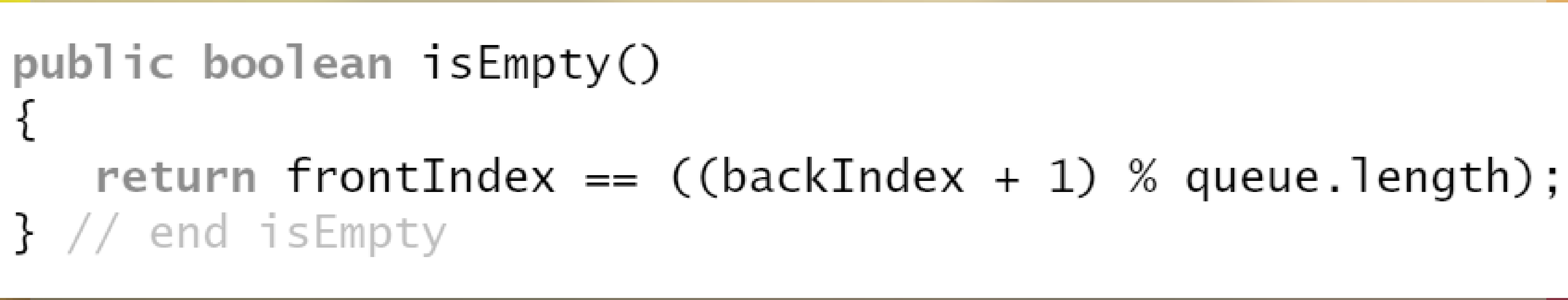
Circular Array with One Unused Location



```
        @SuppressWarnings("unchecked")
        T[] tempQueue = (T[]) new Object[newSize];
        queue = tempQueue;
        for (int index = 0; index < oldSize - 1; index++)
        {
            queue[index] = oldQueue[frontIndex];
            frontIndex = (frontIndex + 1) % oldSize;
        } // end for

        frontIndex = 0;
        backIndex = oldSize - 2;
    } // end if
} // end ensureCapacity
```

Definition of `ensureCapacity`

Circular Array with One Unused Location



```java
public boolean isEmpty()
{
   return frontIndex == ((backIndex + 1) % queue.length);
} // end isEmpty
```

Implementation of `isEmpty`