A List Implementation that Uses An Array

Chapter 13

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

Array to Implement the ADT List

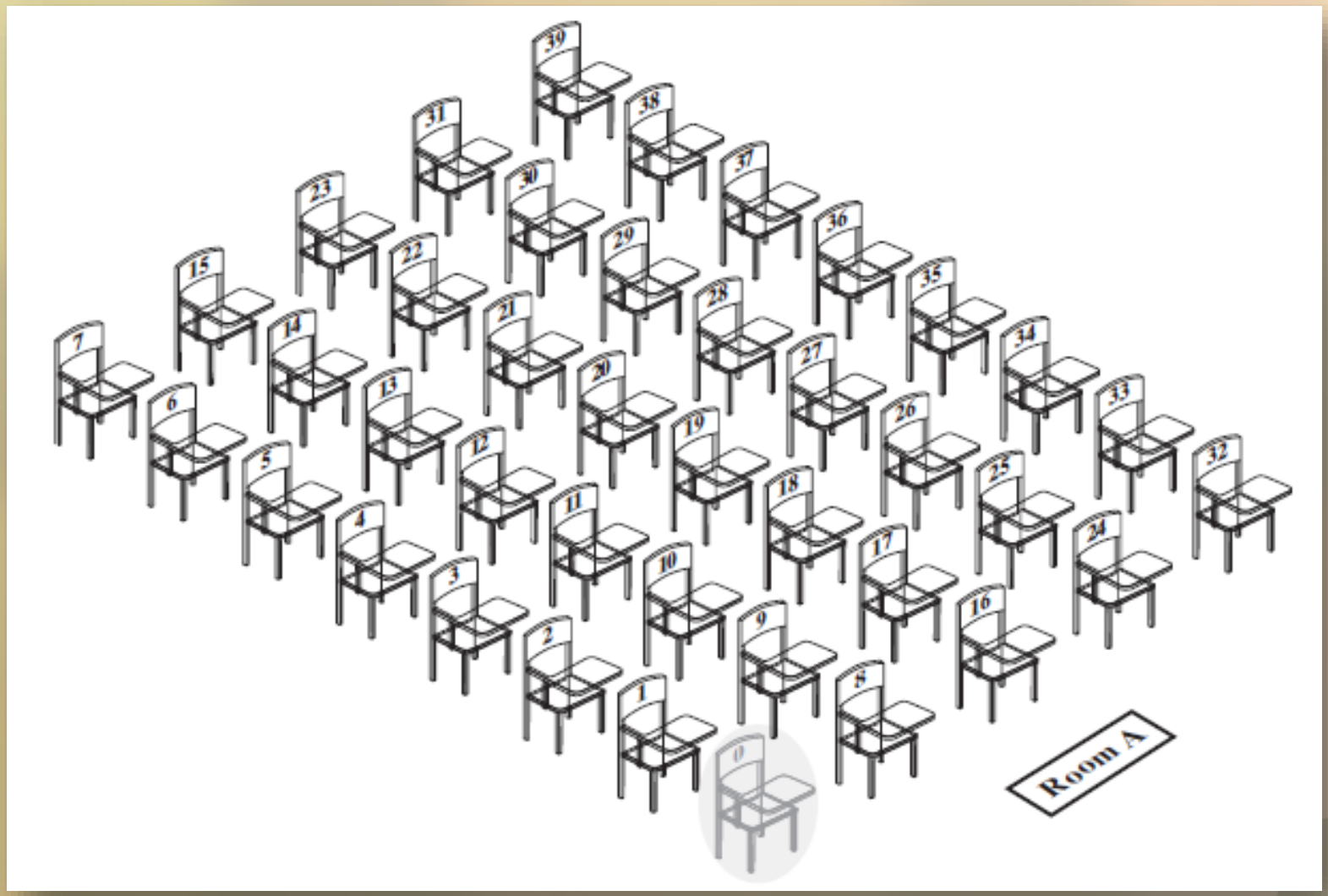

FIGURE 13-1 A classroom that contains desks in fixed positions

Array to Implement the ADT List

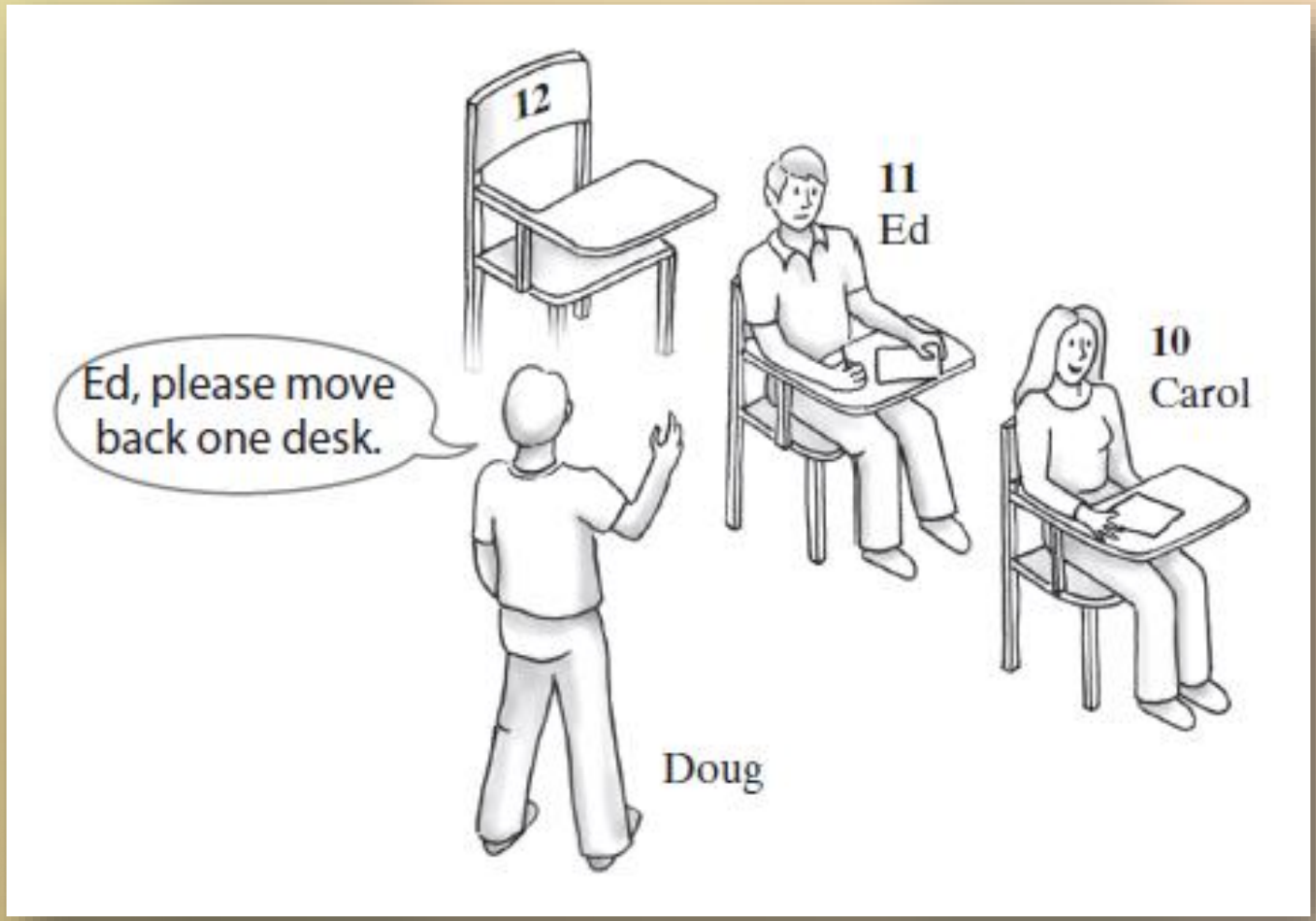

FIGURE 13-2 Seating a new student between two existing students: At least one other student must move

Array to Implement the ADT List

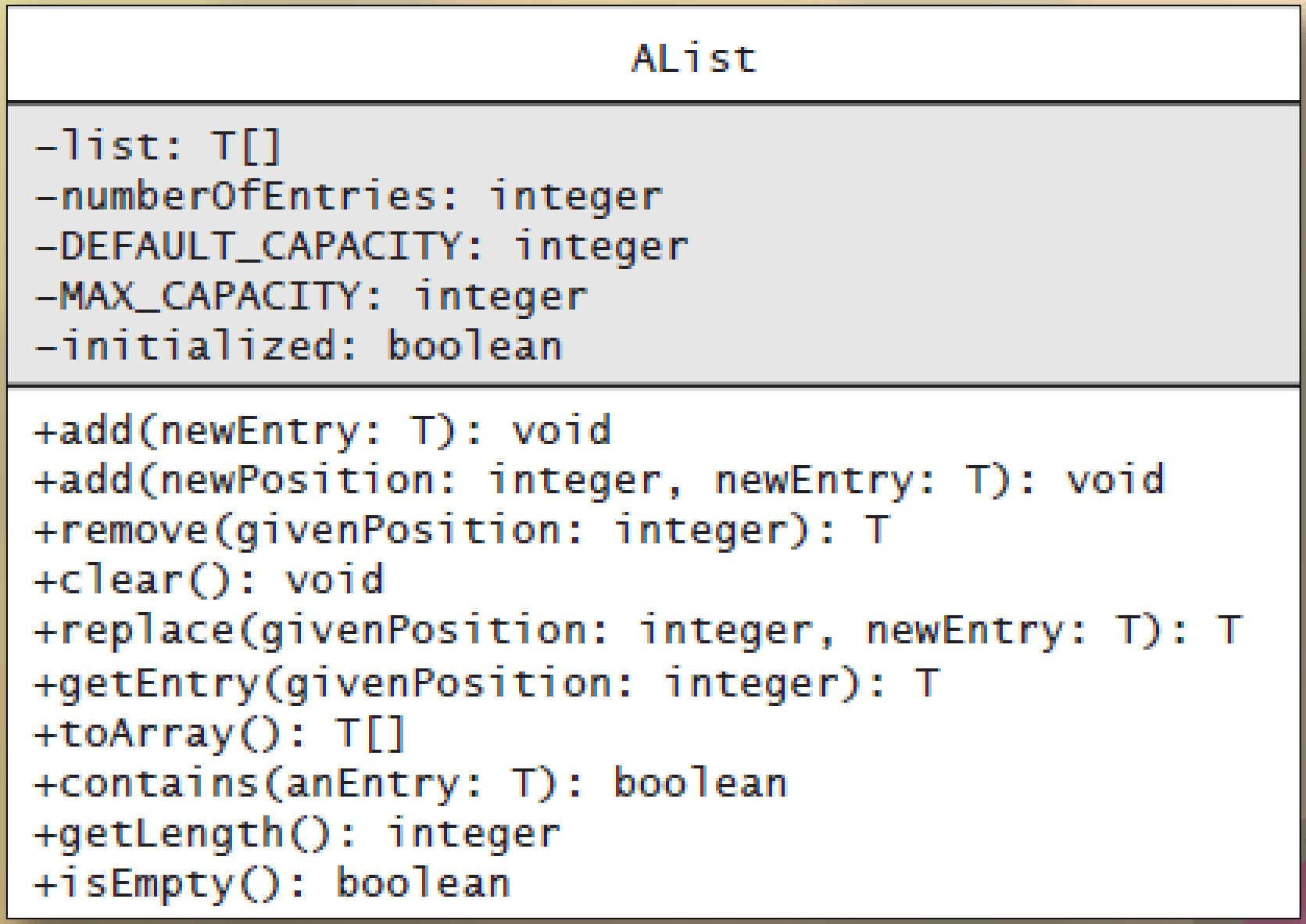

FIGURE 13-3 UML notation for the class **AList**

Array to Implement the ADT List

```
1  import java.util.Arrays;
2  /**
3      A class that implements a list of objects by using an array.
4      Entries in a list have positions that begin with 1.
5      Duplicate entries are allowed.
6      @author Frank M. Carrano
7  */
8  public class AList<T> implements ListInterface<T>
9  {
10     private T[] list;   // Array of list entries; ignore list[0]
11     private int numberOfEntries;
12     private boolean initialized = false;
13     private static final int DEFAULT_CAPACITY = 25;
14     private static final int MAX_CAPACITY = 10000;
15
16     public AList()
17     {
18        this(DEFAULT_CAPACITY); // Call next constructor
19     } // end default constructor
20
21     public AList(int initialCapacity)
```


LISTING 13-1 The class **AList**

Array to Implement the ADT List

```
19      } // end default constructor
20
21      public AList(int initialCapacity)
22      {
23         // Is initialCapacity too small?
24         if (initialCapacity < DEFAULT_CAPACITY)
25            initialCapacity = DEFAULT_CAPACITY;
26         else // Is initialCapacity too big?
27            checkCapacity(initialCapacity);
28
29         // The cast is safe because the new array contains null entries
30         @SuppressWarnings("unchecked")
31         T[] tempList = (T[])new Object[initialCapacity + 1];
32         list = tempList;
33         numberOfEntries = 0;
34         initialized = true;
35      } // end constructor
36
```


LISTING 13-1 The class **AList**

Array to Implement the ADT List

```
36
37      public void add(T newEntry)
38      {
39         checkInitialization();
40         list[numberOfEntries + 1] = newEntry;
41         numberOfEntries++;
42         ensureCapacity();
44      } // end add
45
46      public void add(int newPosition, T newEntry)
47      { < Implementation deferred >
59      } // end add
60
61      public T remove(int givenPosition)
62      { < Implementation deferred >
80      } // end remove
```


LISTING 13-1 The class **AList**

Array to Implement the ADT List

```
81
82      public void clear()
83      { < Implementation deferred >
91      } // end clear
92
93      public T replace(int givenPosition, T newEntry)
94      { < Implementation deferred >
106     } // end replace
107
108     public T getEntry(int givenPosition)
109     { < Implementation deferred >
119     } // end getEntry
120
121     public T[] toArray()
122     {
123        checkInitialization();
124
125        // The cast is safe because the new array contains null entries
126        @SuppressWarnings("unchecked")
127        T[] result = (T[])new Object[numberOfEntries];
128        for (int index = 0; index < numberOfEntries; index++)
129        {
130           result[index] = list[index + 1];
131        } // end for
```

Array to Implement the ADT List

```
130             result[index] = list[index + 1];
131          } // end for
132
133          return result;
134       } // end toArray
135
136       public boolean contains(T anEntry)
137       { < Implementation deferred >
149       } // end contains
150
151       public int getLength()
152       {
153          return numberOfEntries;
154       } // end getLength
155
156       public boolean isEmpty()
157       {
158          return numberOfEntries == 0; // Or getLength() == 0
159       } // end isEmpty
```


LISTING 13-1 The class **AList**

Array to Implement the ADT List

```
    } // end isEmpty

    // Doubles the capacity of the array list if it is full.
    // Precondition: checkInitialization has been called.
    private void ensureCapacity()
    {
       int capacity = list.length - 1;
       if (numberOfEntries >= capacity)
       {
          int newCapacity = 2 * capacity;
          checkCapacity(newCapacity); // Is capacity too big?
          list = Arrays.copyOf(list, newCapacity + 1);
       } // end if
    } // end ensureCapacity

    < This class will define checkCapacity, checkInitialization, and two more private
      methods that will be discussed later. >
} // end AList
```


LISTING 13-1 The class **AList**

Array to Implement the ADT List

```
// Precondition: The array list has room for another entry.
public void add(int newPosition, T newEntry)
{
   checkInitialization();
   if ((newPosition >= 1) && (newPosition <= numberOfEntries + 1))
   {
      if (newPosition <= numberOfEntries)
         makeRoom(newPosition);
      list[newPosition] = newEntry;
      numberOfEntries++;
      ensureCapacity(); // Ensure enough room for next add
   }
   else
      throw new IndexOutOfBoundsException(
                "Given position of add's new entry is out of bounds.");
} // end add
```

Implementation of **`add`** uses a private method **`makeRoom`** to handle the details of moving data within the array

Array to Implement the ADT List

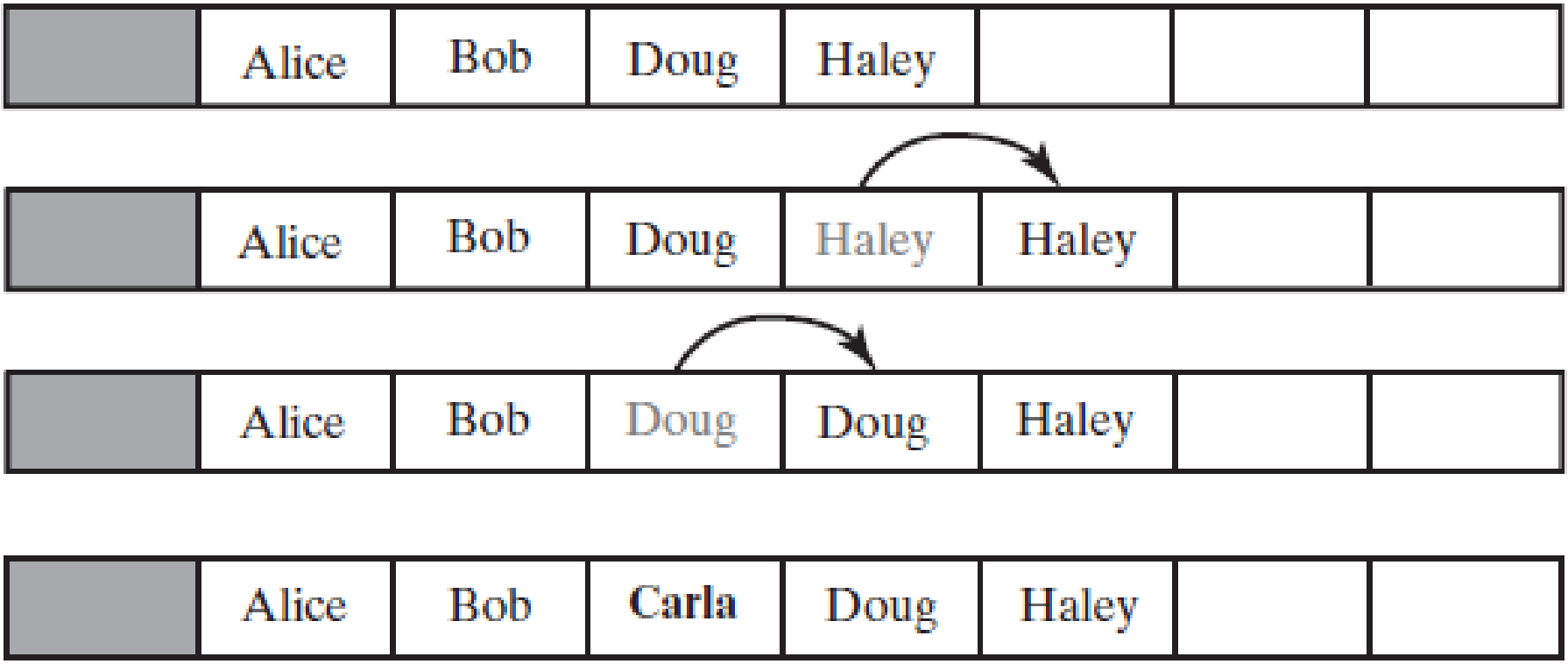

FIGURE 13-4 Making room to insert Carla as the third entry in an array

Array to Implement the ADT List

```
// Makes room for a new entry at newPosition.
// Precondition: 1 <= newPosition <= numberOfEntries + 1;
//               numberOfEntries is list's length before addition;
//               checkInitialization has been called.
private void makeRoom(int newPosition)

{
   assert (newPosition >= 1) && (newPosition <= numberOfEntries + 1);

   int newIndex = newPosition;
   int lastIndex = numberOfEntries;

   // Move each entry to next higher index, starting at end of
   // list and continuing until the entry at newIndex is moved
   for (int index = lastIndex; index >= newIndex; index--)
      list[index + 1] = list[index];
} // end makeRoom
```

Implement the private method **makeRoom**

Array to Implement the ADT List

```
public T remove(int givenPosition)
{
   checkInitialization();
   if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
   {
      assert !isEmpty();
      T result = list[givenPosition]; // Get entry to be removed
      // Move subsequent entries toward entry to be removed,
      // unless it is last in list
      if (givenPosition < numberOfEntries)
         removeGap(givenPosition);
      numberOfEntries--;
      return result; // Return reference to removed entry
   }
   else
      throw new IndexOutOfBoundsException(
                "Illegal position given to remove operation.");
} // end remove
```

Implementation uses a private method **`removeGap`** to handle the details of moving data within the array.

Array to Implement the ADT List

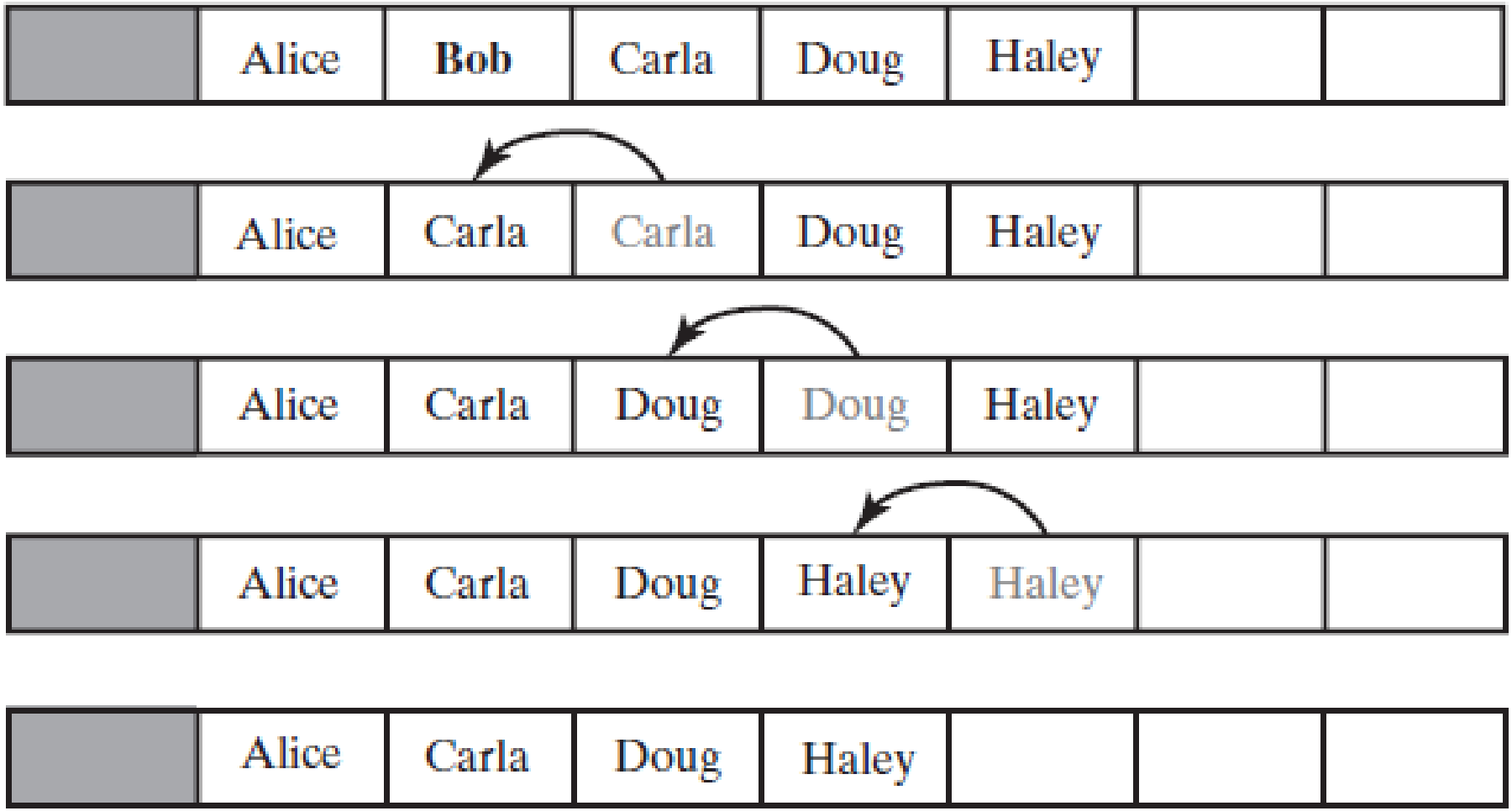

FIGURE 13-5 Removing Bob by shifting array entries

Array to Implement the ADT List

```
// Shifts entries that are beyond the entry to be removed to the
// next lower position.
// Precondition: 1 <= givenPosition < numberOfEntries;
//               numberOfEntries is list's length before removal;
//               checkInitialization has been called.
private void removeGap(int givenPosition)
{
   assert (givenPosition >= 1) && (givenPosition < numberOfEntries);

   int removedIndex = givenPosition;
   int lastIndex = numberOfEntries;

   for (int index = removedIndex; index < lastIndex; index++)

      list[index] = list[index + 1];
} // end removeGap
```

Method **`removeGap`** shifts list entries within the array

Array to Implement the ADT List

```
public boolean replace(int givenPosition, T newEntry)
{
   checkInitialization();
   if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
   {
      assert !isEmpty();
      T originalEntry = list[givenPosition];
      list[givenPosition] = newEntry;
      return originalEntry;
   }
      else
         throw new IndexOutOfBoundsException(
                   "Illegal position given to replace operation.");
} // end replace
```

Method **replace**

Array to Implement the ADT List

```
public T getEntry(int givenPosition)
{
   checkInitialization();
   if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
   {
      assert !isEmpty();
      return list[givenPosition];
   }
   else
      throw new IndexOutOfBoundsException(
                "Illegal position given to getEntry operation.");
} // end getEntry
```

Method **getEntry**

Array to Implement the ADT List

```
public boolean contains(T anEntry)
{
   checkInitialization();
   boolean found = false;
   int index = 1;
   while (!found && (index <= numberOfEntries))
   {
      if (anEntry.equals(list[index]))
         found = true;
      index++;
   } // end while

   return found;
} // end contains
```

Method **`contains`** uses a local boolean variable to terminate the loop when we find the desired entry.

End

Chapter 13