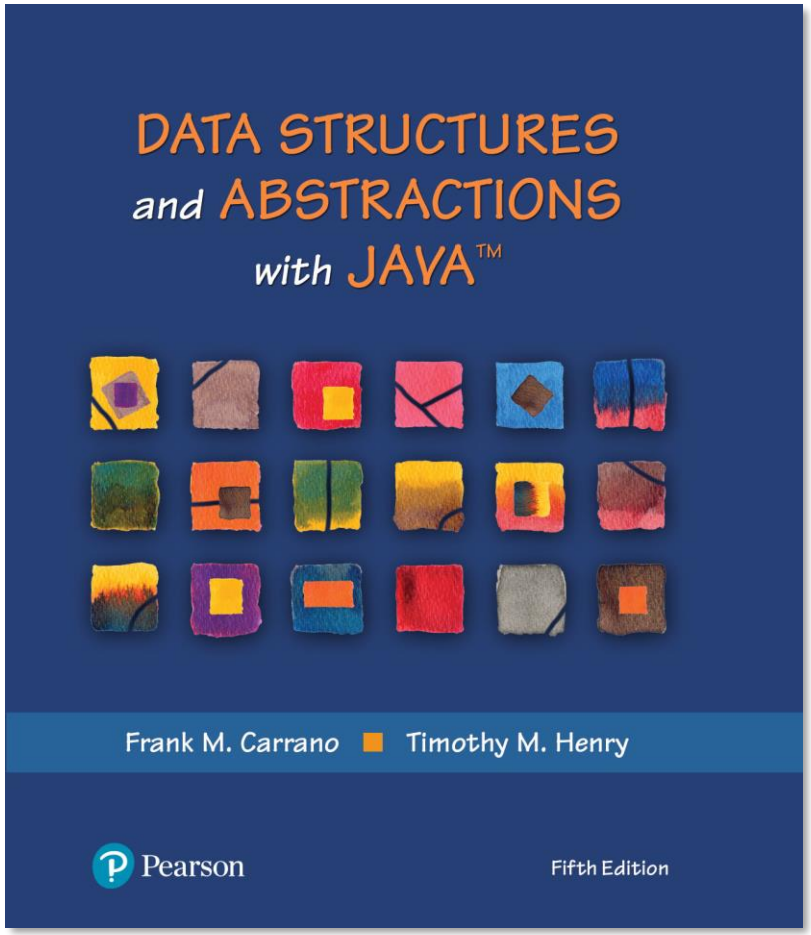
Data Structures and Abstractions with Java™

5th Edition



Chapter 11

A List Implementation That Uses an Array

Using an Array to Implement the ADT List

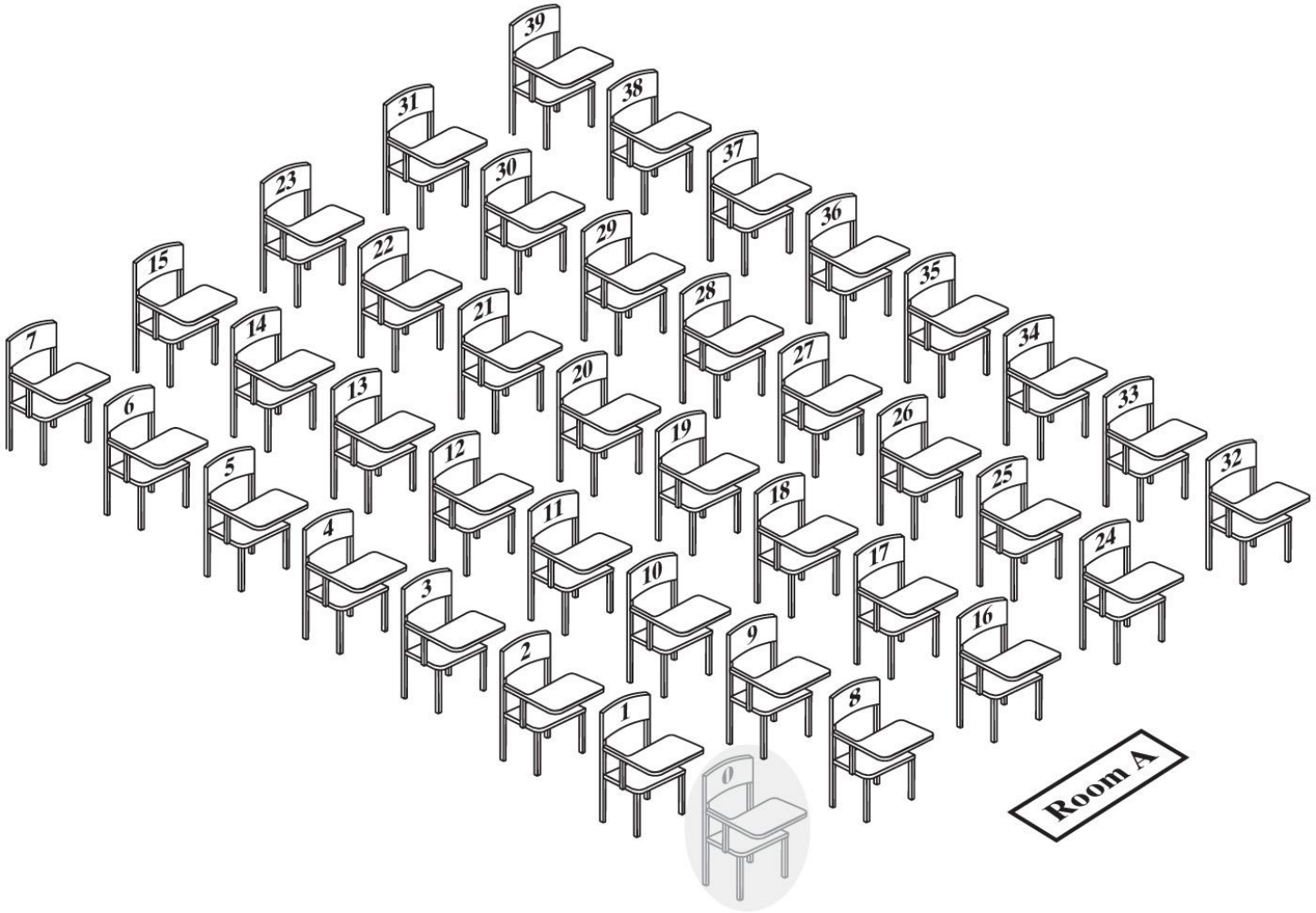


FIGURE 11-1 A classroom that contains desks in fixed positions

Using an Array to Implement the ADT List

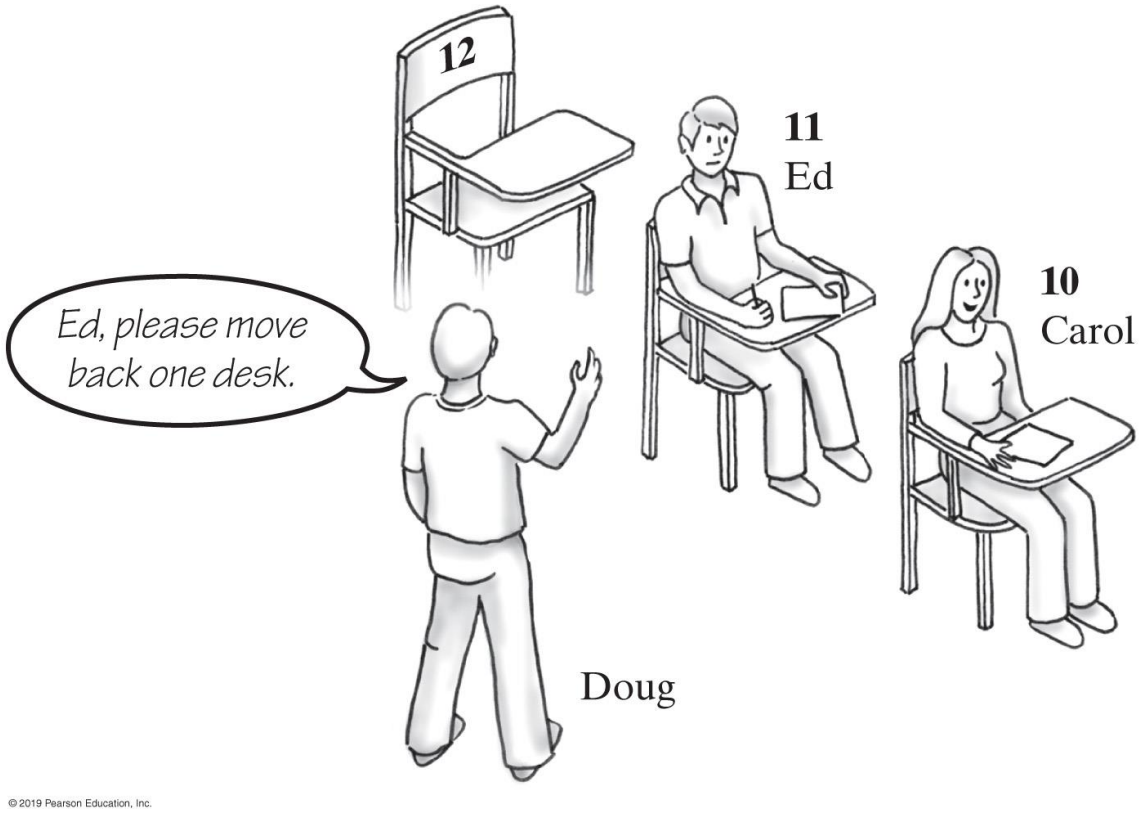


FIGURE 11-2 Seating a new student between two existing students: At least one other student must move

Using an Array to Implement the ADT List

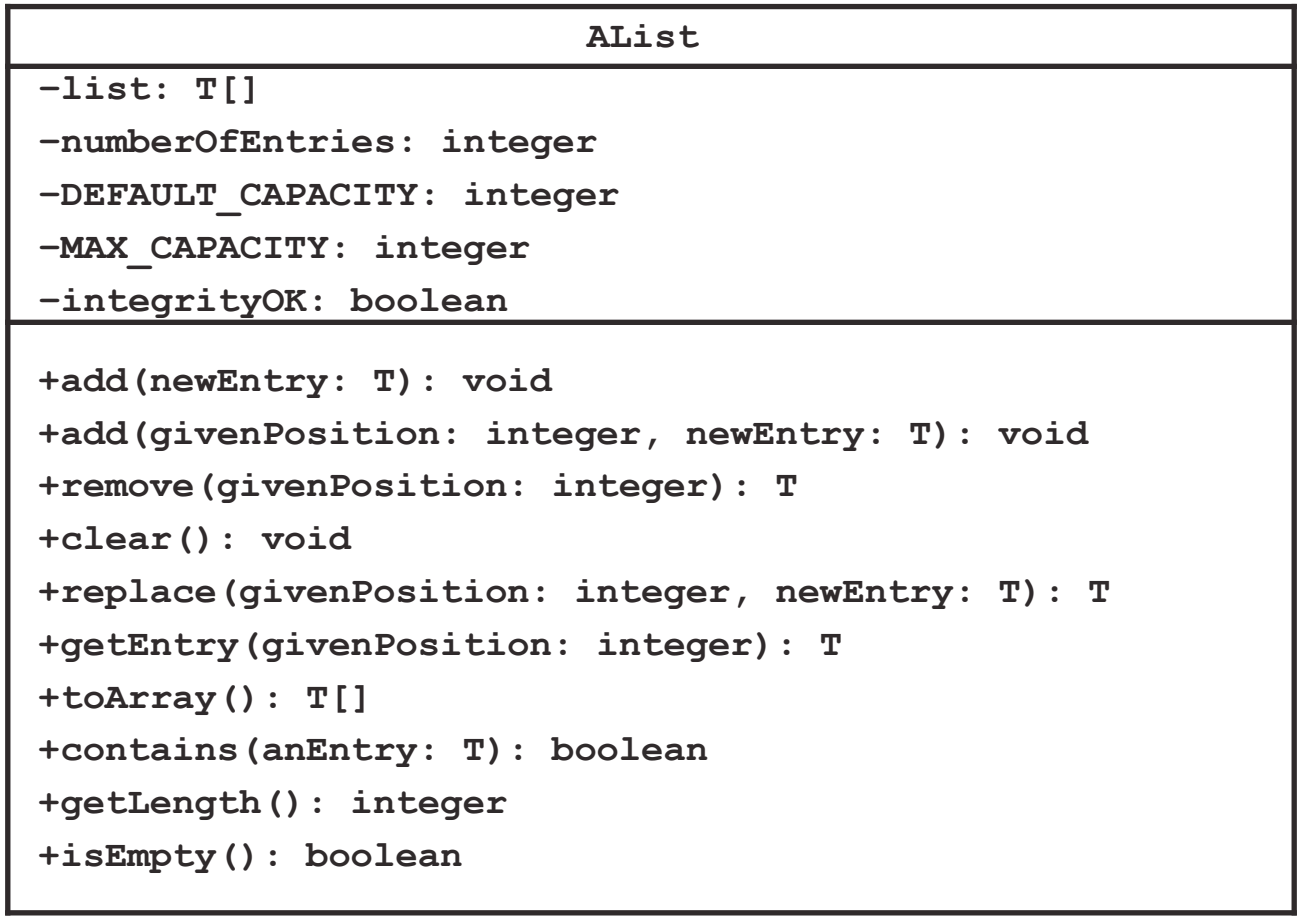


FIGURE 11-3 UML notation for the class AList

An Array List Implementation (Part 1)

```
/**
  A class that implements a list of objects by using an array.
  Entries in a list have positions that begin with 1.
  Duplicate entries are allowed. */
public class AList<T> implements ListInterface<T>
{
      private T[] list;   // Array of list entries; ignore list[0]
      private int numberOfEntries;
  private boolean integrityOK;
      private static final int DEFAULT_CAPACITY = 25;
      private static final int MAX_CAPACITY = 10000;

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

LISTING 11-1 The class `AList`

An Array List Implementation (Part 2)

```
  public AList(int initialCapacity)
  {
integrityOK = false;

// Is initialCapacity too small?
if (initialCapacity < DEFAULT_CAPACITY)
  initialCapacity = DEFAULT_CAPACITY;
else // Is initialCapacity too big?
  checkCapacity(initialCapacity);

// The cast is safe because the new array contains null entries
@SuppressWarnings("unchecked")
T[] tempList = (T[])new Object[initialCapacity + 1];
list = tempList;
numberOfEntries = 0;
integrityOK = true;
  } // end constructor

  public void add(T newEntry)
  {
 checkIntegrity();
 list[numberOfEntries + 1] = newEntry;
 numberOfEntries++;
 ensureCapacity();
//   add(numberOfEntries + 1, newEntry);  // ALTERNATE CODE
  } // end add
```

LISTING 1-1 The class `AList`

An Array List Implementation (Part 3)

```
public void add(int newPosition, T newEntry)
{  /* < Implementation deferred > */
} // end add

public T remove(int givenPosition)
{ /* < Implementation deferred > */
} // end remove

public void clear()
{ /* < Implementation deferred > */
} // end clear

public T replace(int givenPosition, T newEntry)
{ /* < Implementation deferred > */
} // end replace

public T getEntry(int givenPosition)
{ /* < Implementation deferred > */
} // end getEntry
```

LISTING 1-1 The class `AList`

An Array List Implementation (Part 4)

```
public T[] toArray()
{
    checkIntegrity();

  // The cast is safe because the new array contains null entries
  @SuppressWarnings("unchecked")
  T[] result = (T[])new Object[numberOfEntries]; // Unchecked cast
  for (int index = 0; index < numberOfEntries; index++)
  {
    result[index] = list[index + 1];
  } // end for

  return result;
} // end toArray

    public boolean contains(T anEntry)
    { /* < Implementation deferred > */
    } // end contains

    public int getLength()
    {
    return numberOfEntries;
    } // end getLength
```

LISTING 11-1 The class `AList`

An Array List Implementation (Part 5)

```
    public int getLength()
    {
    return numberOfEntries;
    } // end getLength

    public boolean isEmpty()
    {
    return numberOfEntries == 0; // Or getLength() == 0
    } // end isEmpty

// Doubles the capacity of the array list if it is full.
// Precondition: checkIntegrity 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
```

LISTING 1-1 The class `AList`

Using an Array to Implement the ADT List

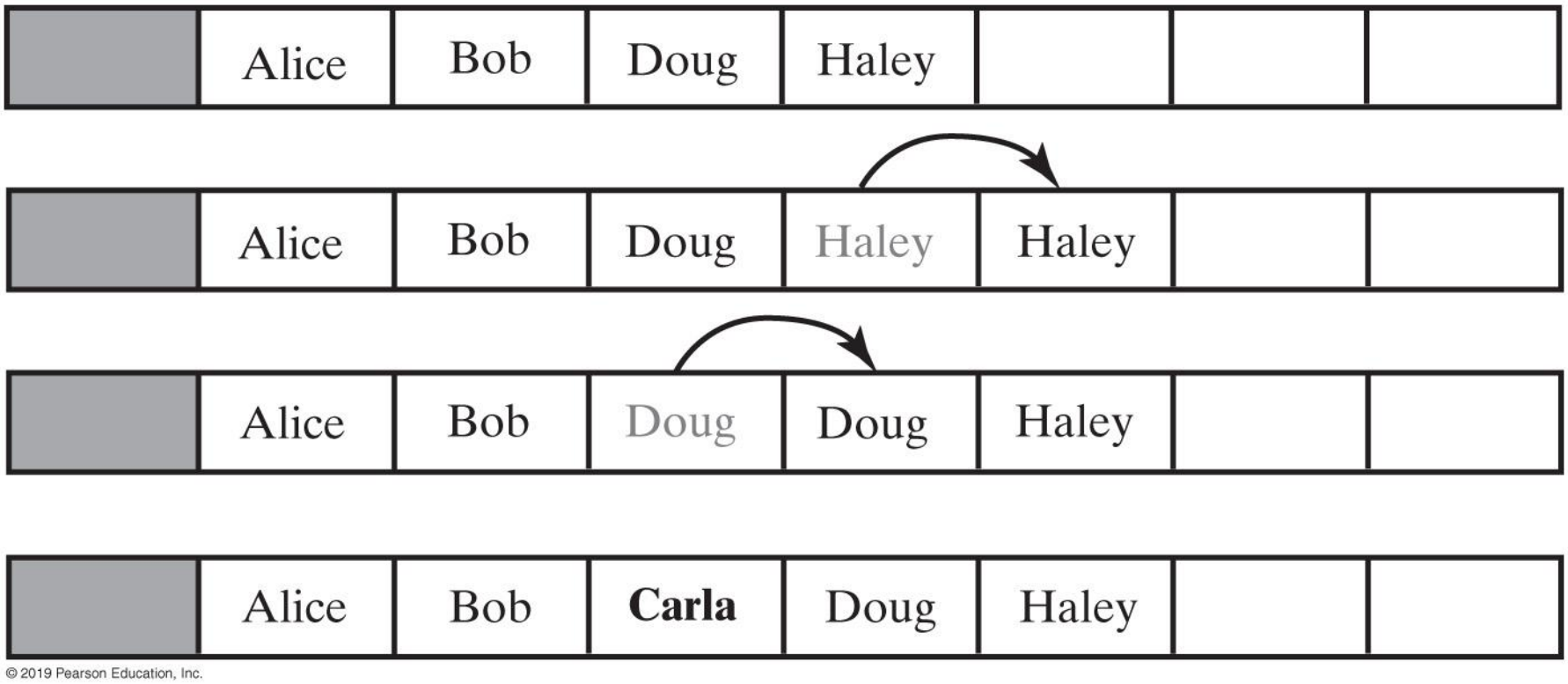


FIGURE 11-4 Making room to insert Carla as the third entry in an array

Using an Array to Implement the ADT List

```
// Precondition: The array list has room for another entry.
public void add(int newPosition, T newEntry)
{
  checkIntegrity();
  // Assertion: The array list has room for another entry.
  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
```

Implementing add at a specific position

Using an Array to Implement the ADT List

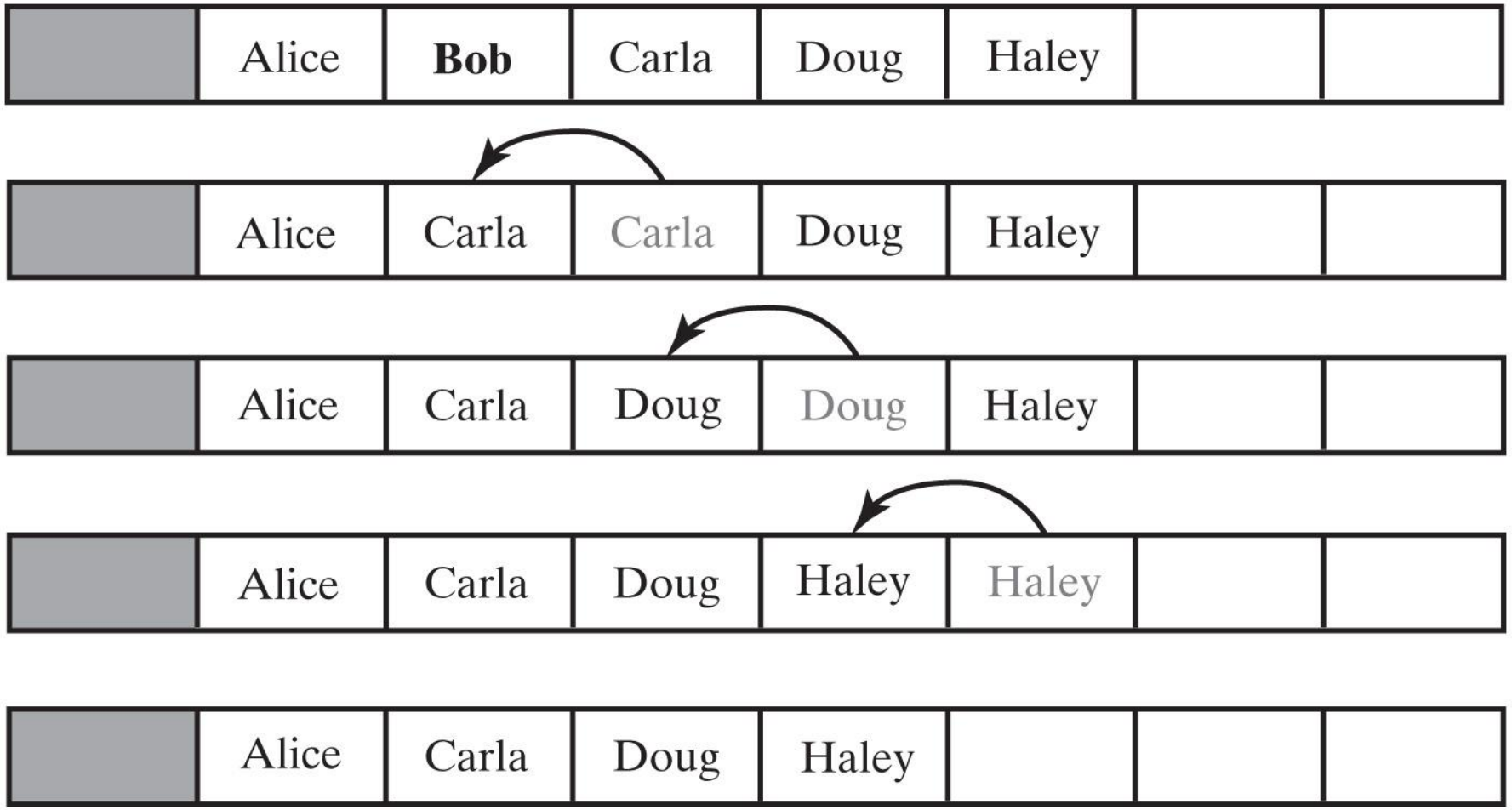


FIGURE 11-5 Removing Bob by shifting array entries

Using an Array to Implement the ADT List

```
public T remove(int givenPosition)
{
  checkIntegrity();
  if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
  {
    // Assertion: The list is not empty
    T result = list[givenPosition]; // Get entry to be removed
    // Move subsequent entries towards entry to be removed,
    // unless it is last in list
    if (givenPosition < numberOfEntries)
      removeGap(givenPosition);
    list[numberOfEntries] = null;
    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.

Using an 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;
//               checkIntegrity has been called.
private void removeGap(int givenPosition)
{
  int removedIndex = givenPosition;
  for (int index = removedIndex; index < numberOfEntries; index++)
    list[index] = list[index + 1];
} // end removeGap
```

Method `removeGap` shifts list entries within the array

Using an Array to Implement the ADT List

```
public boolean contains(T anEntry)
{
  checkIntegrity();
  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.

Using an Array to Implement the ADT List

```
public void add(T newEntry)
{
  checkIntegrity();
  list[numberOfEntries + 1] = newEntry;
  numberOfEntries++;
  ensureCapacity();
} // end add
```

Operation that adds a new entry to the end of a list. Efficiency O(1) if new if array is not resized.

Using an Array to Implement the ADT List

```
public void add(int givenPosition, T newEntry)
{
  checkIntegrity();
  if ((givenPosition >= 1) && (givenPosition <= numberOfEntries + 1))
  {
    if (givenPosition <= numberOfEntries)
      makeRoom(givenPosition);
    list[givenPosition] = 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
```

Add a new entry to a list at a client-specified position.

Using an Array to Implement the ADT List

```
private void makeRoom(int givenPosition)
{
  int newIndex = givenPosition;
  int lastIndex = numberOfEntries;
  for (int index = lastIndex; index >= newIndex; index--)
    list[index + 1] = list[index];
} // end makeRoom
```

Method add uses method makeRoom.

End

Chapter 11