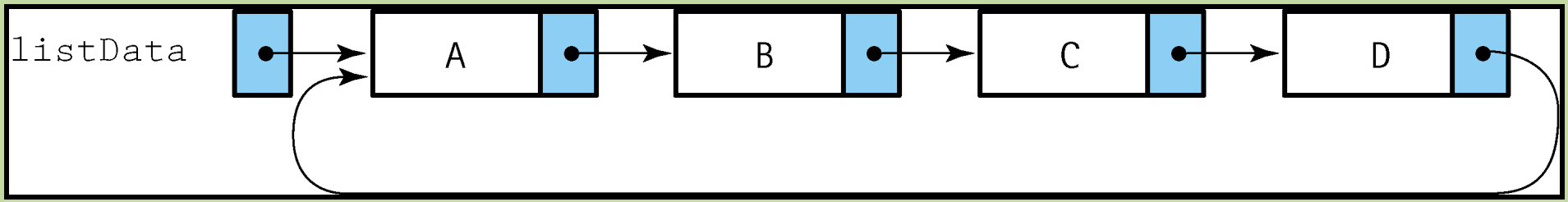
Object-Oriented Data Structures Using Java

Chapter 6

Lists Plus

Circular Linked List

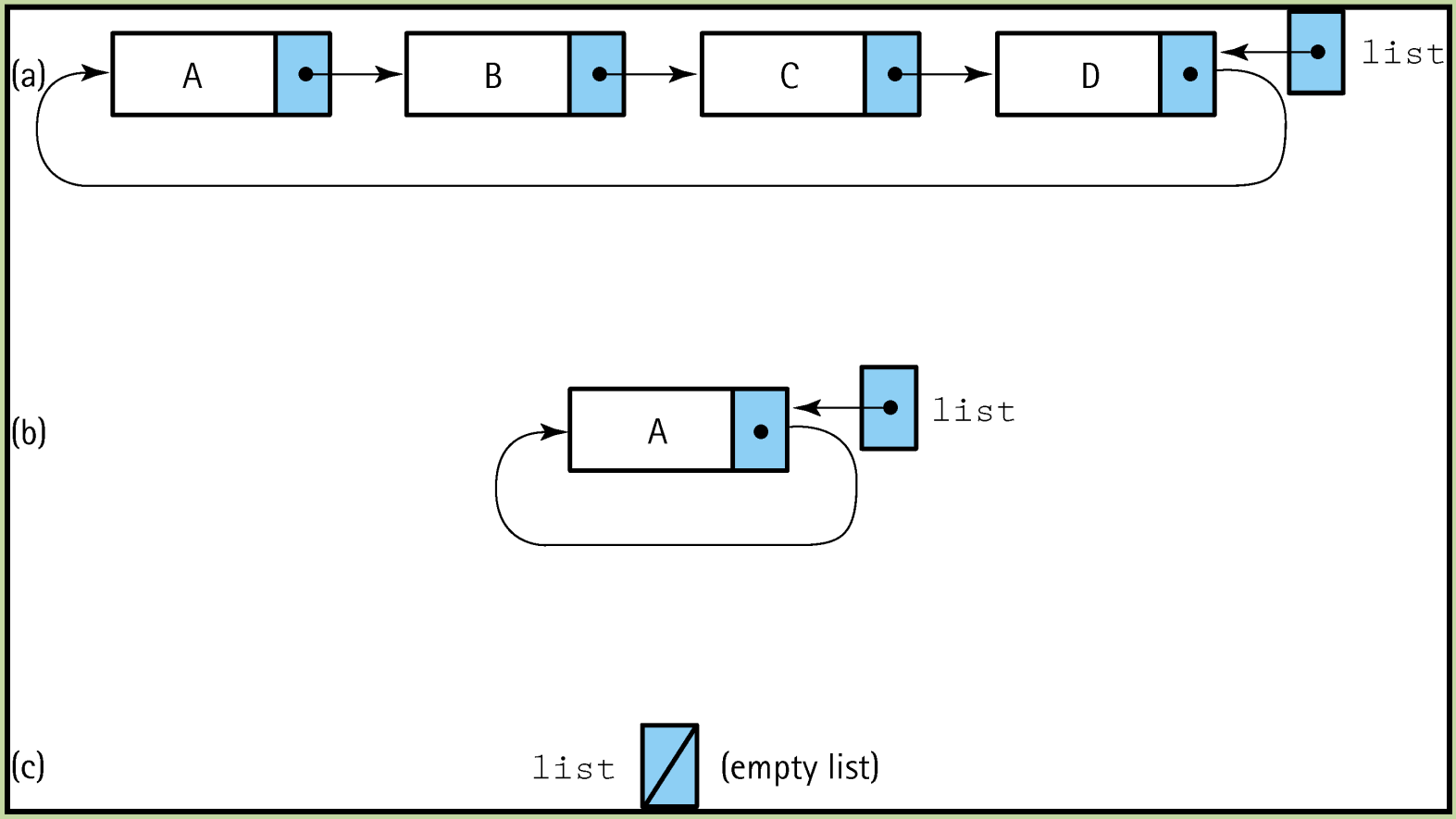
A list in which every node has a successor; the "last" element is succeeded by the "first" element.



An Alternate Approach

1. Circular linked lists with the external pointer pointing to the rear element
2. Now we have direct access to both the first and the last elements in the list.

An Alternate Approach *(cont'd)*



The CircularSortedLinkedList Class

The beginning of the new class definition looks like this:

```
//-----------------------------------------------------------------------------
// CircularSortedLinkedList.java      by Dale/Joyce/Weems            Chapter 6
//
// Completes the definition of a link-based list under the assumption
//  that the list is circular and is kept sorted
//-----------------------------------------------------------------------------

package ch06.genericLists;

import ch04.genericLists.*;

public class CircularSortedLinkedList extends LinkedList
{
  public CircularSortedLinkedList()
  // Instantiates an empty list object
  {
    super();
  }
```

The Iterator Methods

Linear

```
public void reset()
// Initializes current position for
//  an iteration through this list
{
  currentPos = list;
}

public Listable getNextItem ()
// Returns copy of the next element
{
  Listable nextItemInfo =
     currentPos.info.copy();
  if (currentPos.next == null)
    currentPos = list;
  else
    currentPos = currentPos.next;

  return nextItemInfo;
}
```

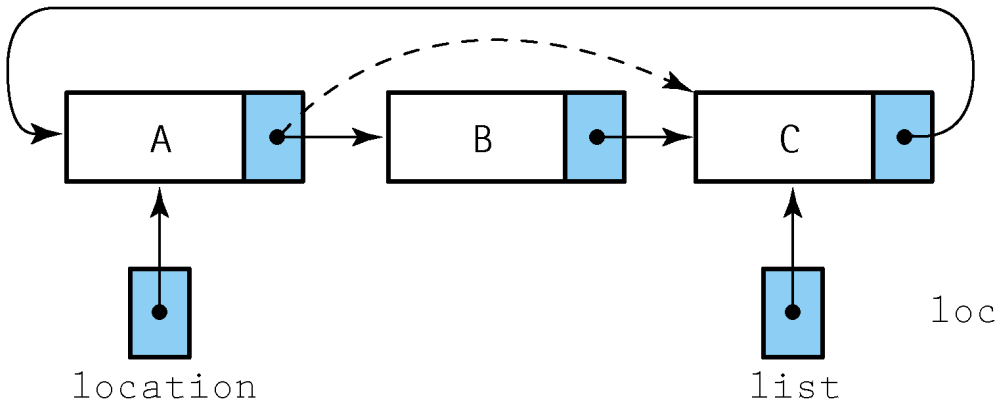
Circular

```
public void reset()
// Initializes current position for
//  an iteration through this list
{
  if (list == null)
    currentPos = list;
  else
    currentPos = list.next;
}

public Listable getNextItem ()
// Returns copy of the next element
{
  Listable nextItemInfo =
   currentPos.info.copy();
  currentPos = currentPos.next;

  return nextItemInfo;
}
```

The `isThere` Method

Deleting from a Circular List

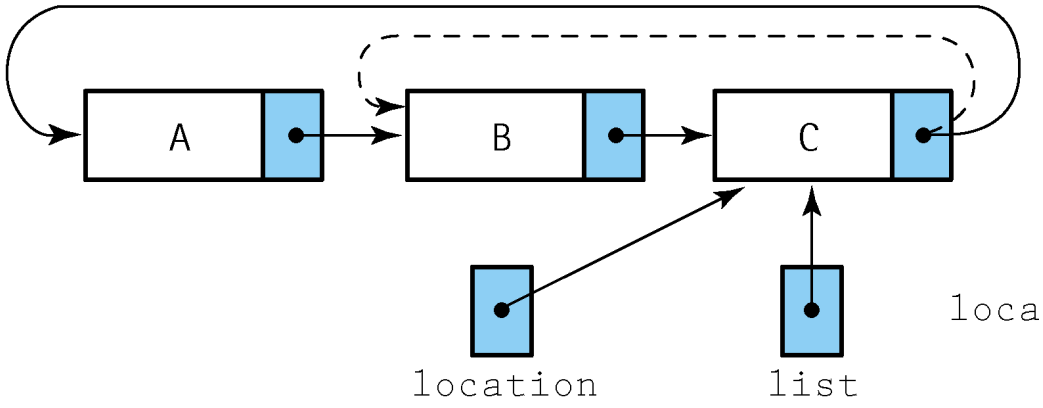
(a) The general case (delete B)




```
location.next = location.next.next;
```

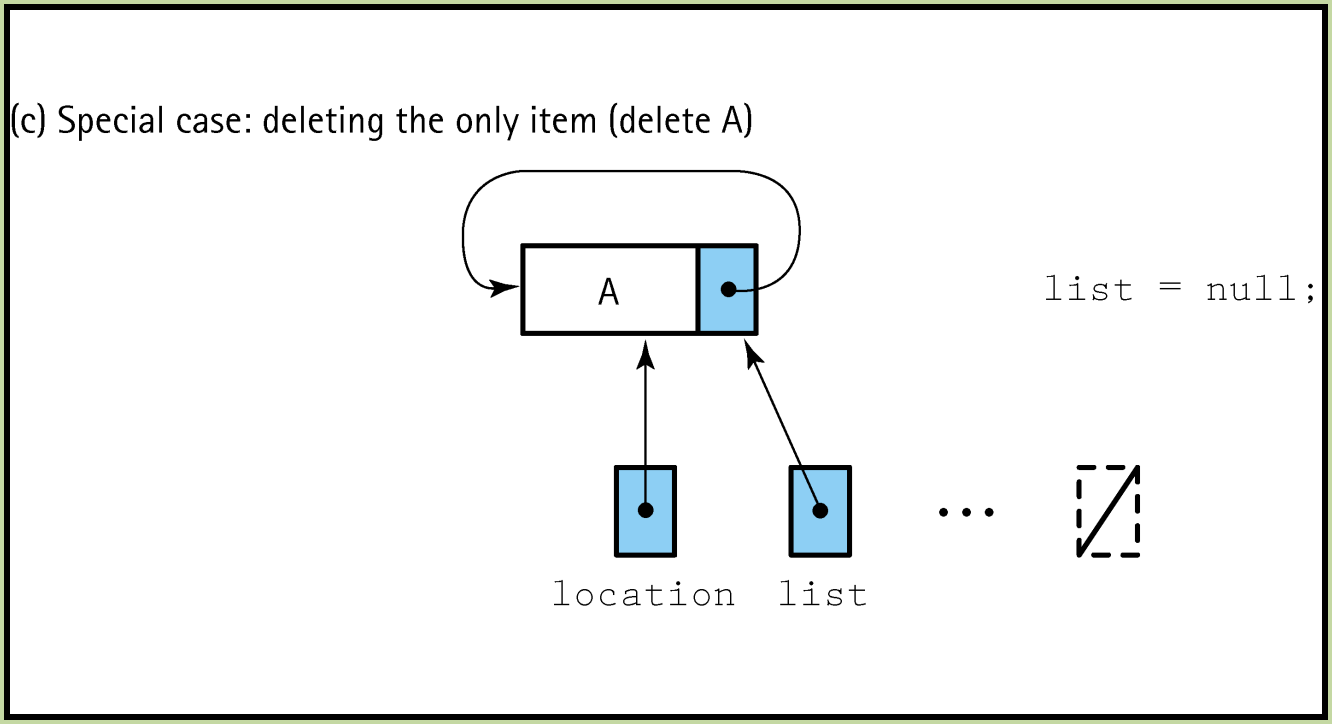
Deleting from a Circular List *(cont'd)*

(b) Special case (?): deleting the smallest item (delete A)



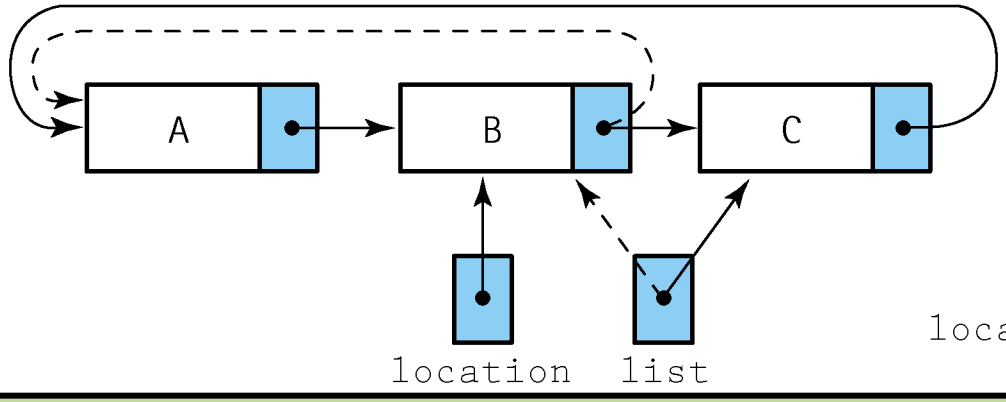

```
location.next = location.next.next;
```

Deleting from a Circular List *(cont'd)*



Deleting from a Circular List *(cont'd)*

(d) Special case: deleting the largest item (delete C)




```
list = location;
```

(the general case PLUS:)

```
location.next = location.next.next;
```

Deleting from a Circular List *(cont'd)*

- The code for the delete method:

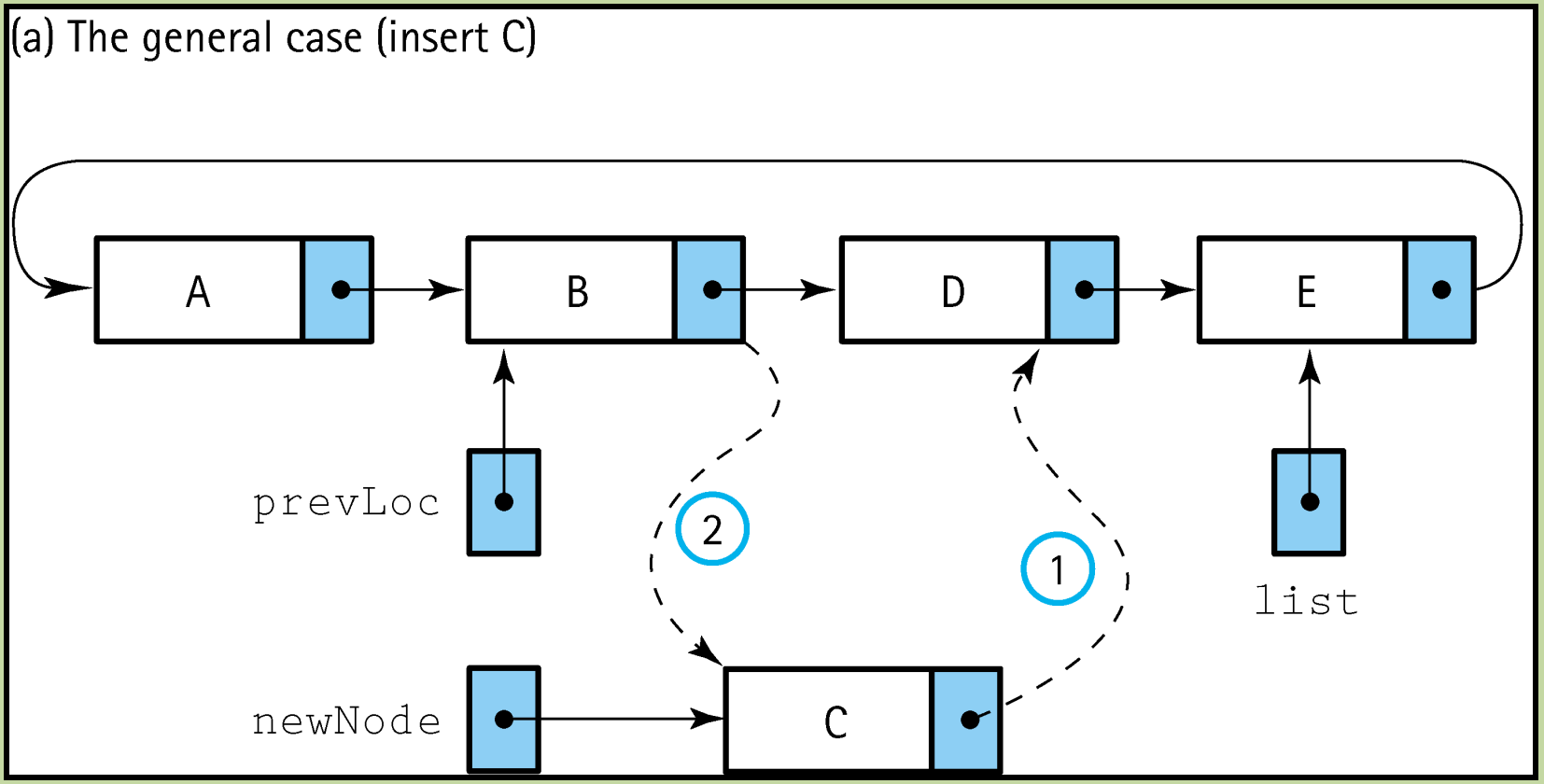
```
public void delete (Listable item)
// Deletes the element of this list whose key matches item's key
{
  ListNode location = list;

  if (location == location.next)    // Single element list
    list = null;
  else
    {
    while (item.compareTo(location.next.info) != 0)
      location = location.next;
    if (location.next == list)          // Deleting last element
      list = location;
    // Delete node at location.next
    location.next = location.next.next;
    }
  numItems--;
}
```

The `insert` Method

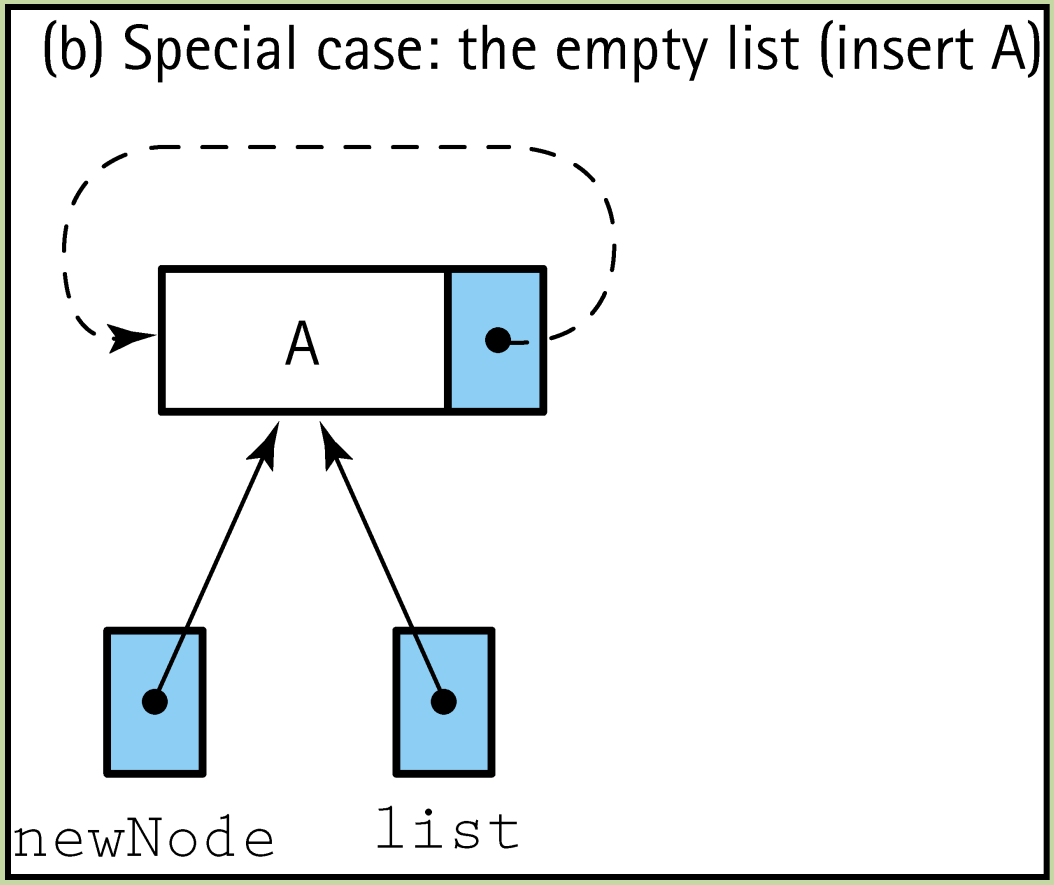
insert (item)

Create a node for the new list element
Find the place where the new element belongs
Put the new element into the list
Increment the number of items

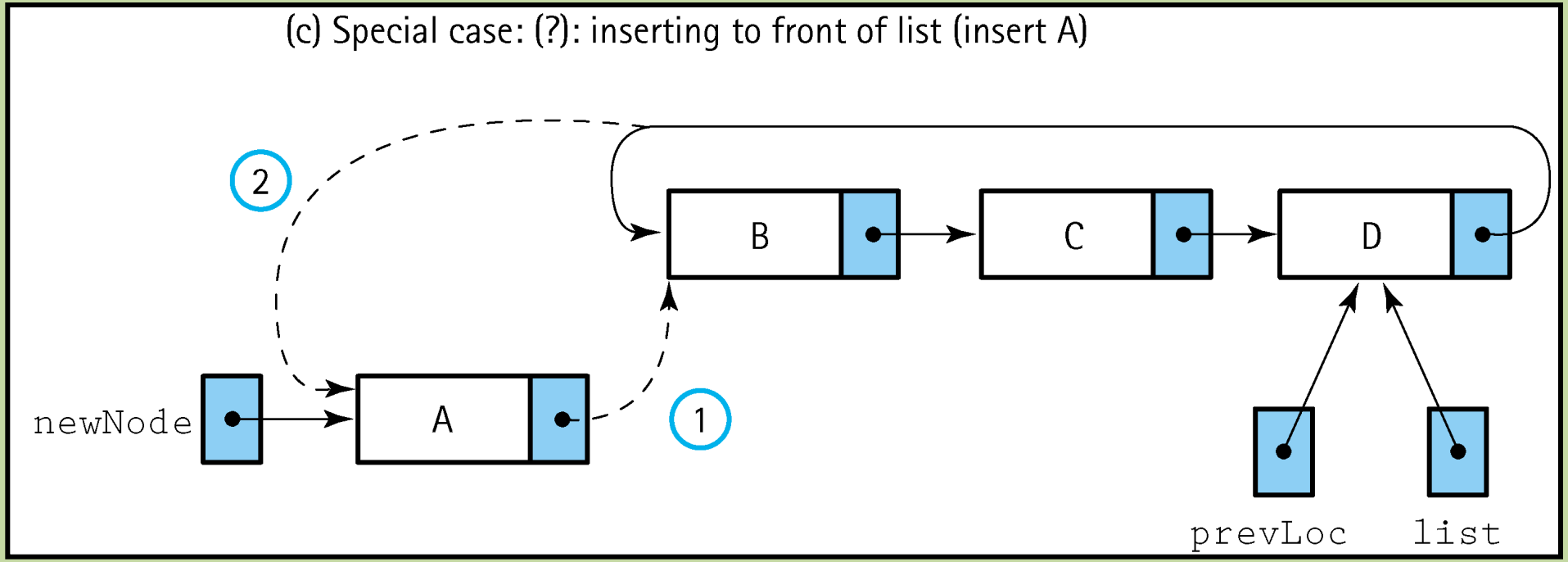
The `insert` Method



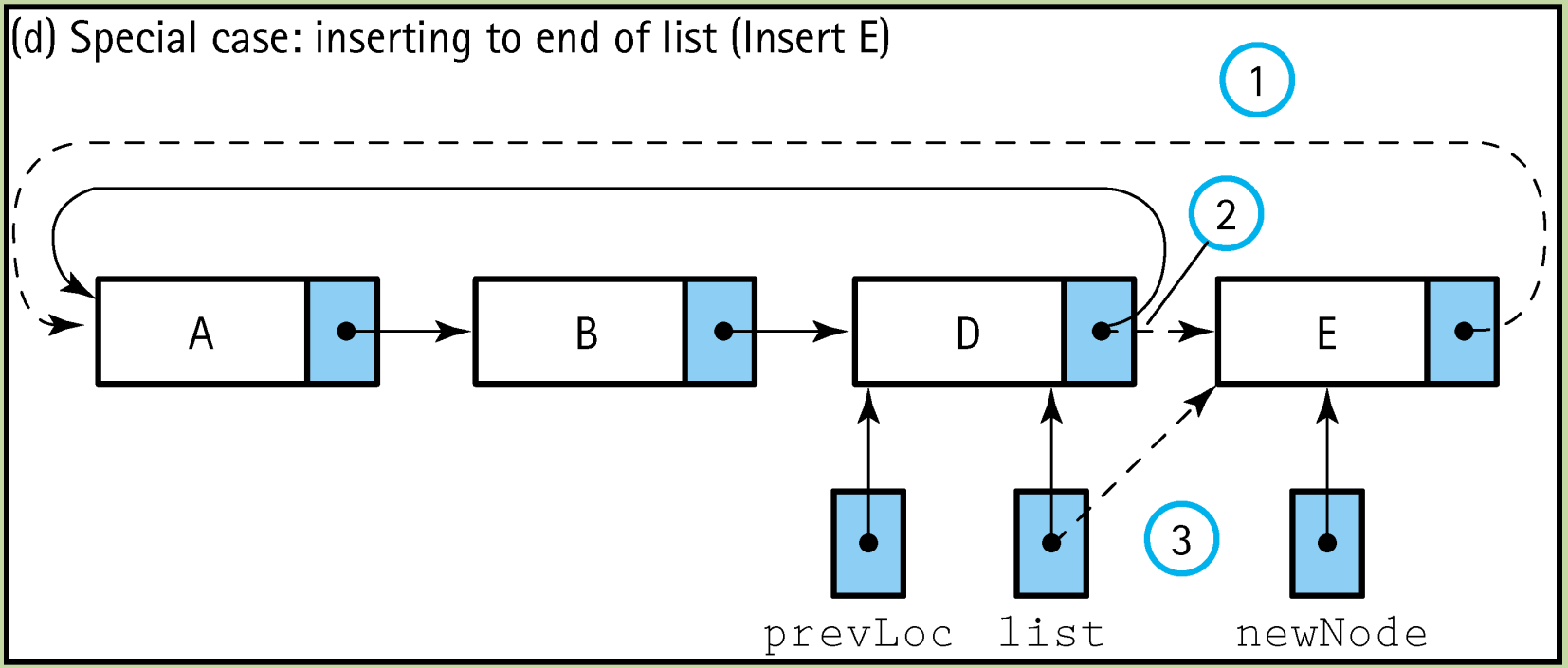
The `insert` Method



The `insert` Method



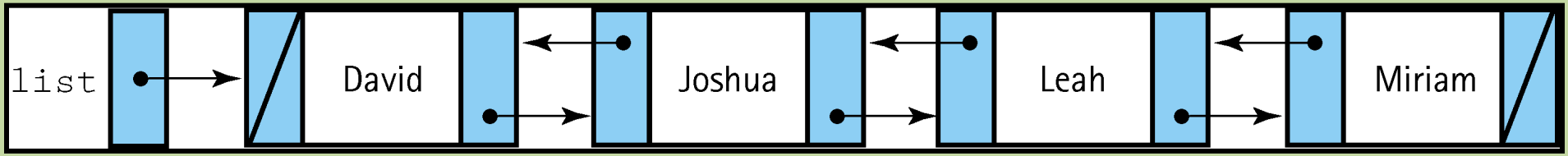
The `insert` Method



The `insert` Method

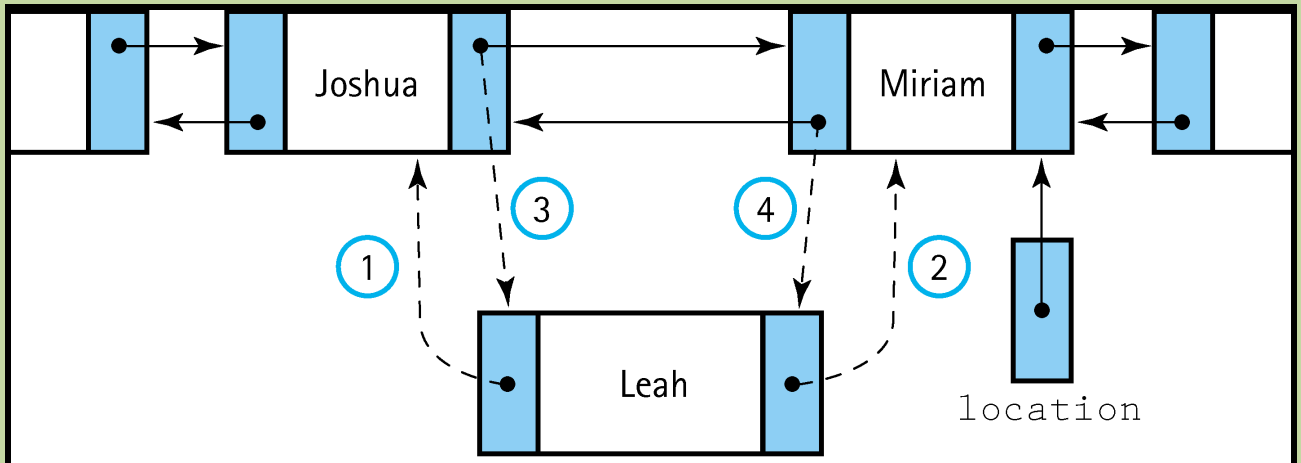
Doubly Linked List

- A linked list in which each node is linked to both its successor and its



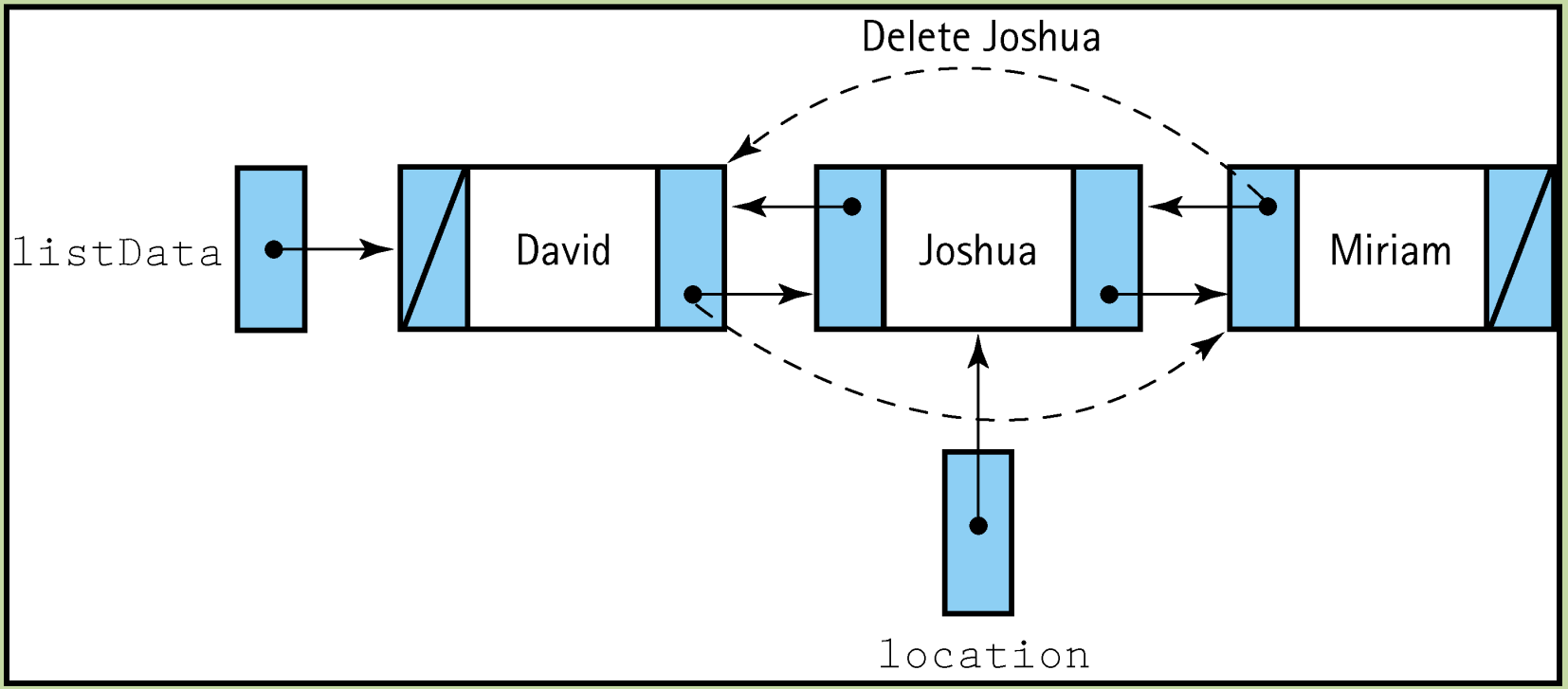
```
protected class DLListNode
{
  // List nodes for the doubly linked list implementation
  protected Listable info;     // The info in a list node
  protected DLListNode next;   // A link to the next node on the list
  protected DLListNode back;   // A link to the next node on the list
}
```

Inserting into a Doubly Linked List




```
newNode.back = location.back;
newNode.next = location;
location.back.next = newNode;
location.back = newNode;
```

Deleting from a Doubly Linked List



How do doubly-linked interfaces fit our framework?

- Define a new interface that extends `ListInterface` and adds a `getPreviousItem` method.
- **Inheritance of interfaces** A Java interface can extend another Java interface, inheriting its requirements. If interface B extends interface A, then classes that implement interface B must also implement interface A. Usually, interface B adds additional abstract methods to those required by interface A.

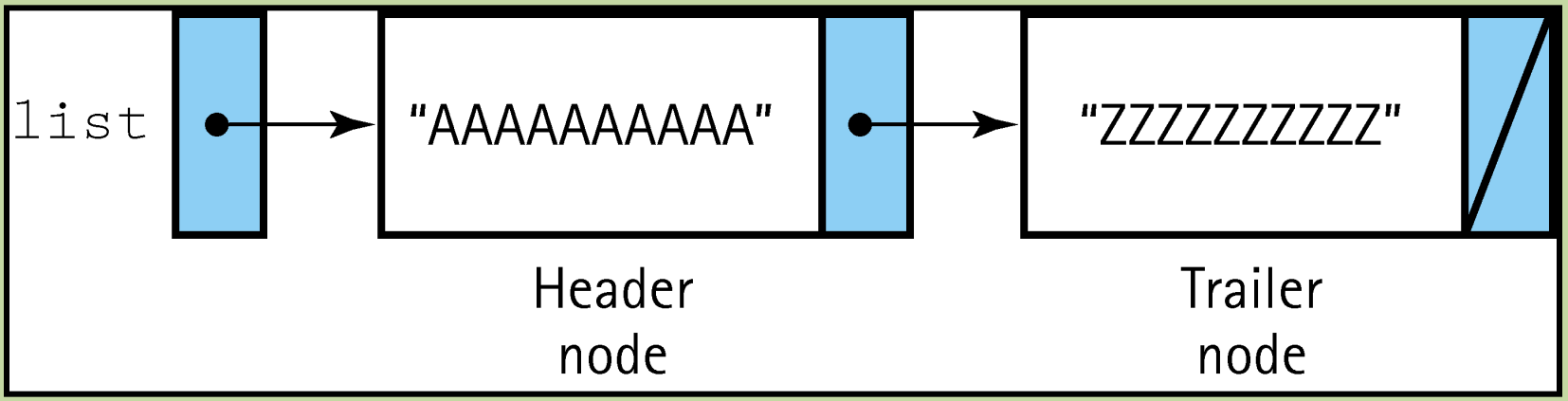
The New Interface:

```
package ch06.genericLists;

import ch04.genericLists.*;

public interface TwoWayListInterface extends ListInterface

// Interface for a class that implements a list of elements as defined
//  by ListInterface, with the additional operation(s) defined below

{
  public Listable getPreviousItem ();
  // Effect:          Returns a copy of the element preceding the current
  //                  position on this list and moves back the value of the
  //                  current position
  //                  The last element precedes the first element
  // Preconditions:   Current position is defined
  //                  There exists a list element preceding current
  //                  position. No list transformers have been called since
  //                  most recent reset
  // Postcondition:   Return value = (a copy of element previous to current
  //                  position)
}
```

Linked Lists with Headers and Trailers

- **Header node** A placeholder node at the beginning of a list; used to simplify list processing
- **Trailer node** A placeholder node at the end of a list; used to simplify list processing
- Reduces need for special processing

An "Empty" List with a Header and a Trailer



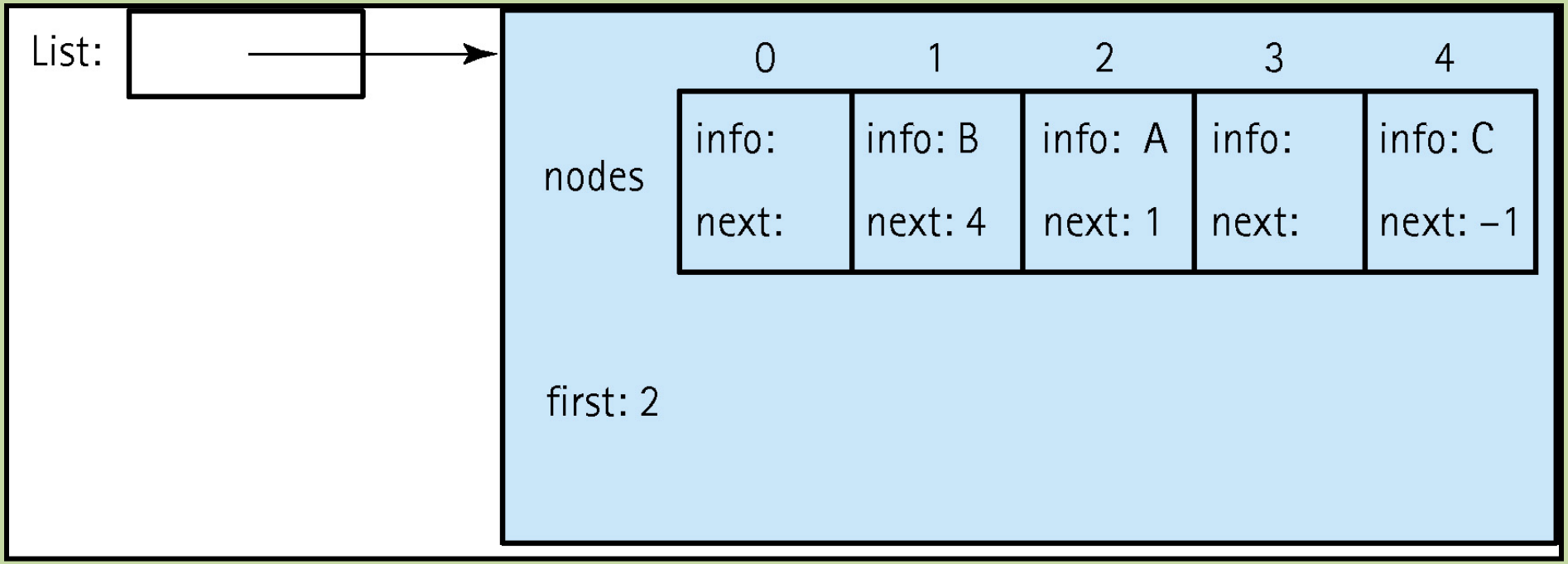
A Linked List as an Array of Nodes

- A linked list could be implemented in an array.
- The elements might be stored in an array in any order, and "linked" by their indexes.
- We keep a separate list of available nodes.

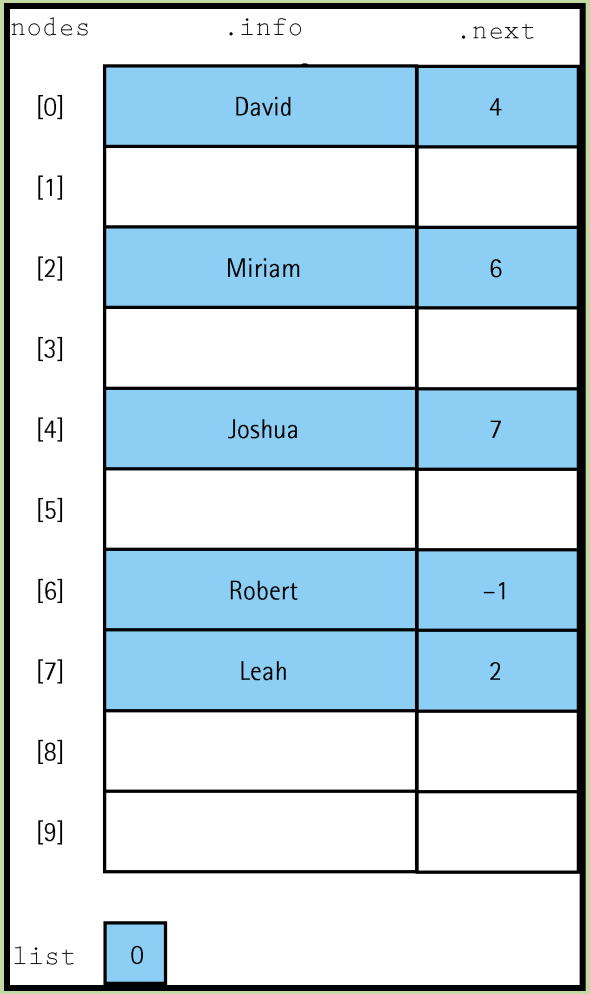
Linked List in Static Storage

```
class AListNode
{
 protected Listable info;
 protected int Next;
}

class ArrayLinkedList
{
 AListNode[] nodes = new AListNode[5];
 int first;
}

ArrayLinkedList list = new ArrayLinkedList();
```

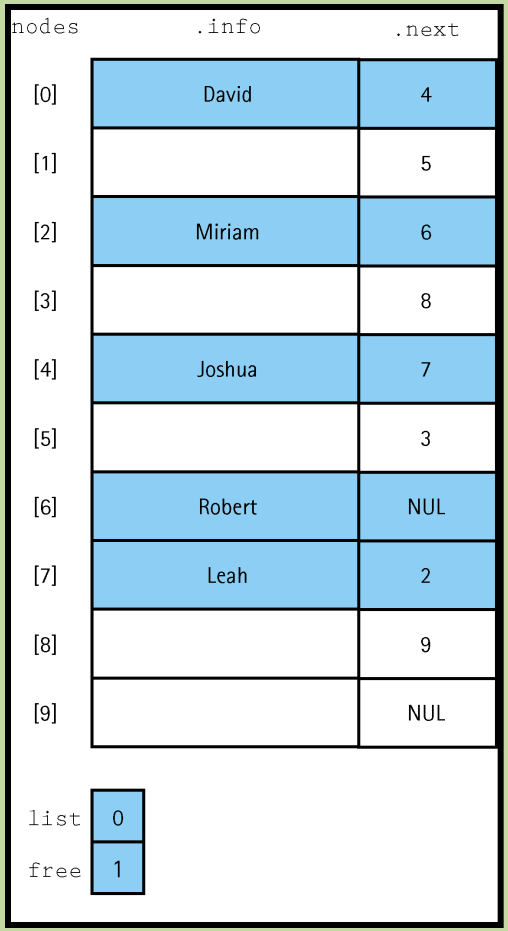
Linked List in Static Storage



A Sorted list Stored in an Array of Nodes



An Array with Linked List of Values and Free Space



An Array with Three Lists (Including the Free List)