Sorted Lists

Chapter 16

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Contents

- Specifications for the ADT Sorted List
  - Using the ADT Sorted List
- A Linked Implementation
  - The Method **`add`**
  - The Efficiency of the Linked Implementation
- An Implementation That Uses the ADT List
  - Efficiency Issues

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Objectives

- Use sorted list in a program
- Describe differences between ADT list and ADT sorted list
- Implement ADT sorted list by using chain of linked nodes
- Implement ADT sorted list by using operations of ADT list

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Sorted Lists

- We extend the capability of a list
  - Previous example used list to organize names in alphabetical order
- Consider need to keep list sorted in numerical or alphabetic order after list established
  - We add or remove an element
  - The ADT handles keeping elements in order

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Specifications for the ADT Sorted List

- Possible operations
  - For simplicity, duplicate entries allowed
  - Must determine where in list element is added
  - Can ask if list contains specified entry
  - Must be able to remove an entry

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Abstract Data Type: Sorted List

- Data
  - A collection of objects in sorted order and having same data type
  - Number of objects in collection
- Operations
  - add(newEntry)
  - remove(anEntry)
  - getPosition(anEntry)

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Abstract Data Type: Sorted List

- Operations used from ADT list (Ch. 12)
  - getEntry(givenPosition)
  - contains(anEntry)
  - remove(givenPosition)
  - clear()
  - getLength()
  - isEmpty()
  - toArray()
- Interface, Listing 16-1

Note: Code listing files must be in same folder as PowerPoint files for links to work

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 1 Suppose that wordList is an unsorted list of words. Using the operations of the ADT list and the ADT sorted list, create a sorted list of these words.

Question 2 Assuming that the sorted list you created in the previous question is not empty, write Java statements that

a. Display the last entry in the sorted list.

b. Add the sorted list's first entry to the sorted list again.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 1 Suppose that wordList is an unsorted list of words. Using the operations of the ADT list and the ADT sorted list, create a sorted list of these words.

```
SortedListInterface<String> sortedWordList = new SortedList<String>();
int numberOfWords = wordList.getLength();
for (int position = 1; position <= numberOfWords; position++)
      sortedWordList.add(wordList.getEntry(position));
```

Question 2 Assuming that the sorted list you created in the previous question is not empty, write Java statements that

a. Display the last entry in the sorted list.

```
int length = sortedWordList.getLength();
String lastEntry = sortedWordList.getEntry(length);
System.out.println(lastEntry);
```

b. Add the sorted list's first entry to the sorted list again.

```
sortedList.add(sortedList.getEntry(1));
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

A Linked Implementation

- Linked implementation of the ADT sorted list, Listing 16-2
- Note different versions of method **`add`**
  - Iterative version
  - Recursive version

Copyright ©2012 by Pearson Education, Inc. All rights reserved

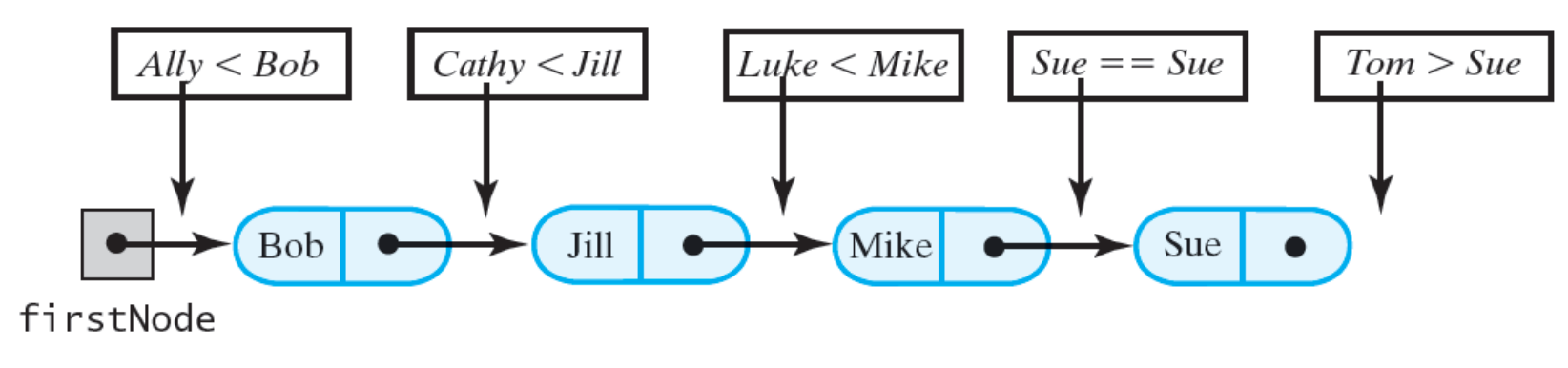


Figure 16-1 Places to insert names into a sorted chain of linked nodes

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 3 In the while statement of the method getNodeBefore , how important is the order of the two boolean expressions that the operator && joins? Explain.

while ((currentNode != null) && (anEntry.compareTo (currentNode.getData ()) > 0))

Question 4 What does getNodeBefore return if the sorted list is empty? How can you use this fact to simplify the implementation of the method add given in Segment 16.10?

Question 5 Suppose that you use the previous method add to add an entry to a sorted list. If the entry is already in the list, where in the list will add insert it? Before the first occurrence of the entry, after the first occurrence of the entry, after the last occurrence of the entry, or somewhere else?

Question 6 What would be the answer to the previous question if you changed > to >= in the while statement of the method getNodeBefore ?

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 3 In the while statement of the method getNodeBefore , how important is the order of the two boolean expressions that the operator && joins? Explain.

while ((currentNode != null) && (anEntry.compareTo (currentNode.getData ()) > 0))

The order is critical. When currentNode is null, currentNode != null is false. Thus, the en tire expression in the while statement is false without executing the call currentNode.getData() . If the latter call were to execute first when currentNode was null, an exception would occur. Thus, the while statement should remain as written.

Question 4 What does getNodeBefore return if the sorted list is empty? How can you use this fact to simplify the implementation of the method add given in Segment 16.10?

When the sorted list is empty, getNodeBefore returns null. Thus, in the definition of add, you can omit the call to isEmpty in the if statement.

Question 5 Suppose that you use the previous method add to add an entry to a sorted list. If the entry is already in the list, where in the list will add insert it? Before the first occurrence of the entry, after the first occurrence of the entry, after the last occurrence of the entry, or somewhere else?

Before the first occurrence of the entry.

Question 6 What would be the answer to the previous question if you changed > to >= in the while statement of the method getNodeBefore ?

After the last occurrence of the entry.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

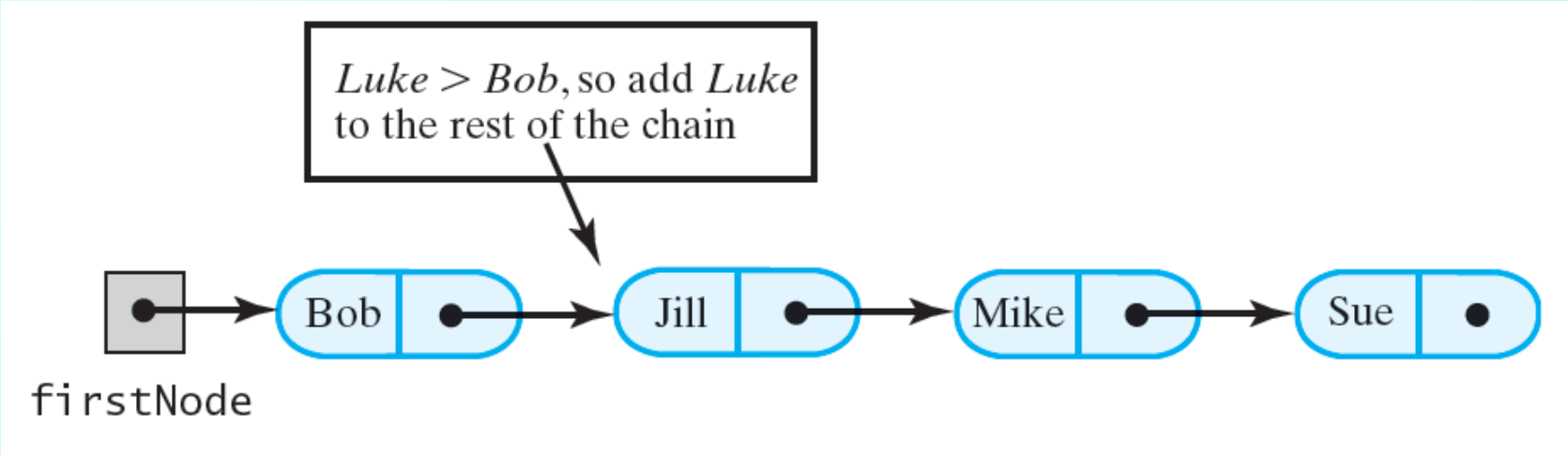


Figure 16-2 Recursively adding *Luke* to a sorted chain of names

Copyright ©2012 by Pearson Education, Inc. All rights reserved

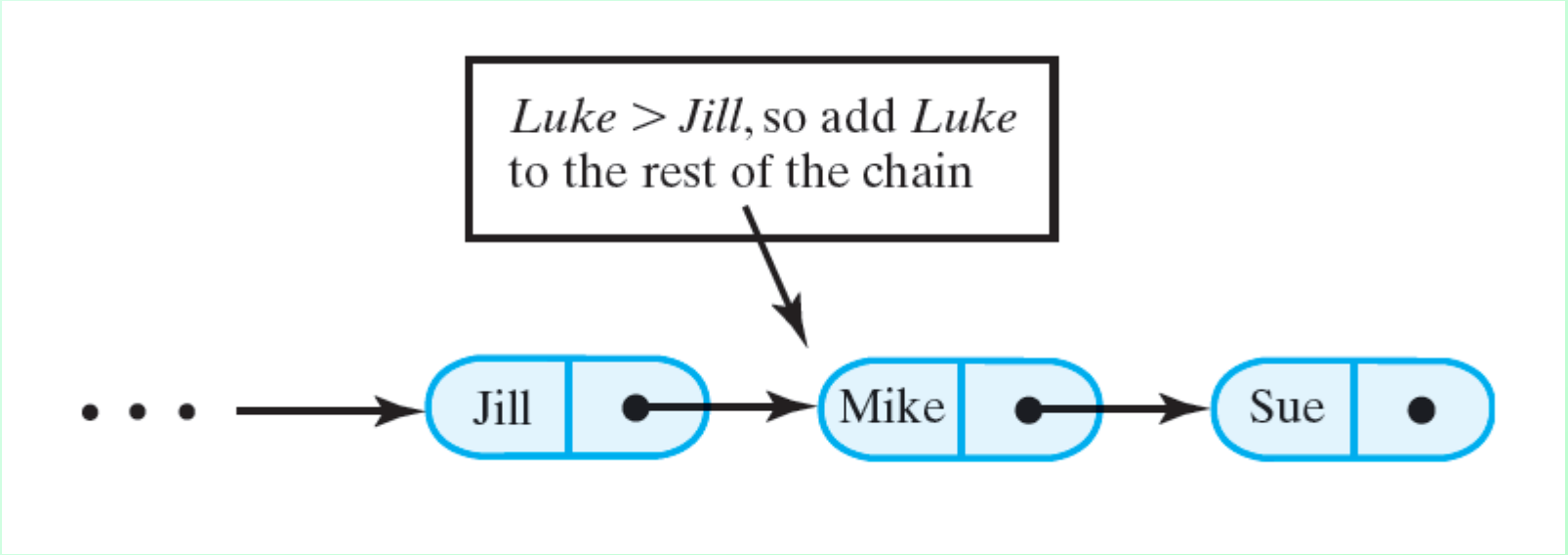


Figure 16-2 Recursively adding *Luke* to a sorted chain of names

Copyright ©2012 by Pearson Education, Inc. All rights reserved

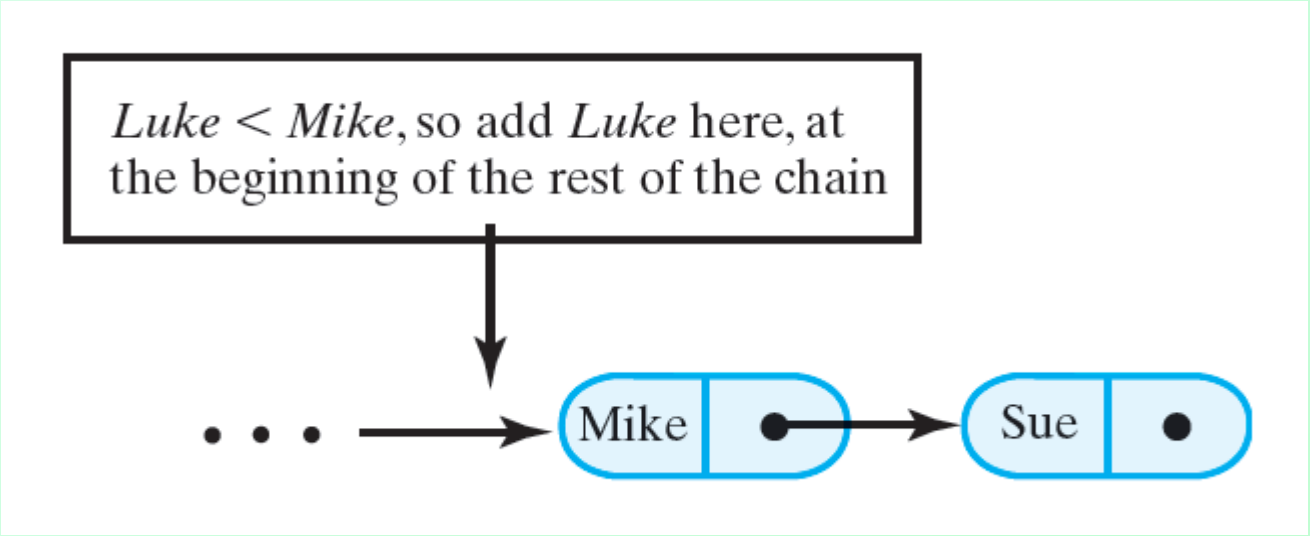


Figure 16-2 Recursively adding *Luke* to a sorted chain of names

Copyright ©2012 by Pearson Education, Inc. All rights reserved

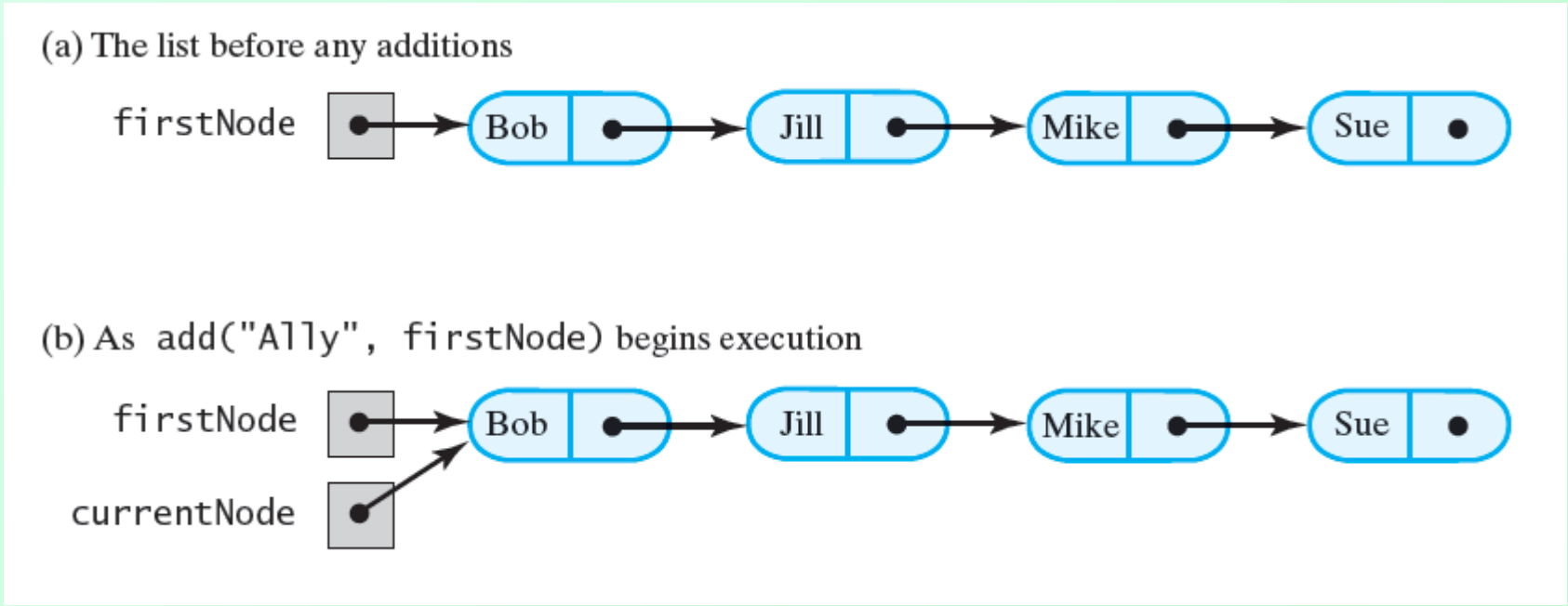


Figure 16-3 Recursively adding a node at the beginning of a chain

Copyright ©2012 by Pearson Education, Inc. All rights reserved

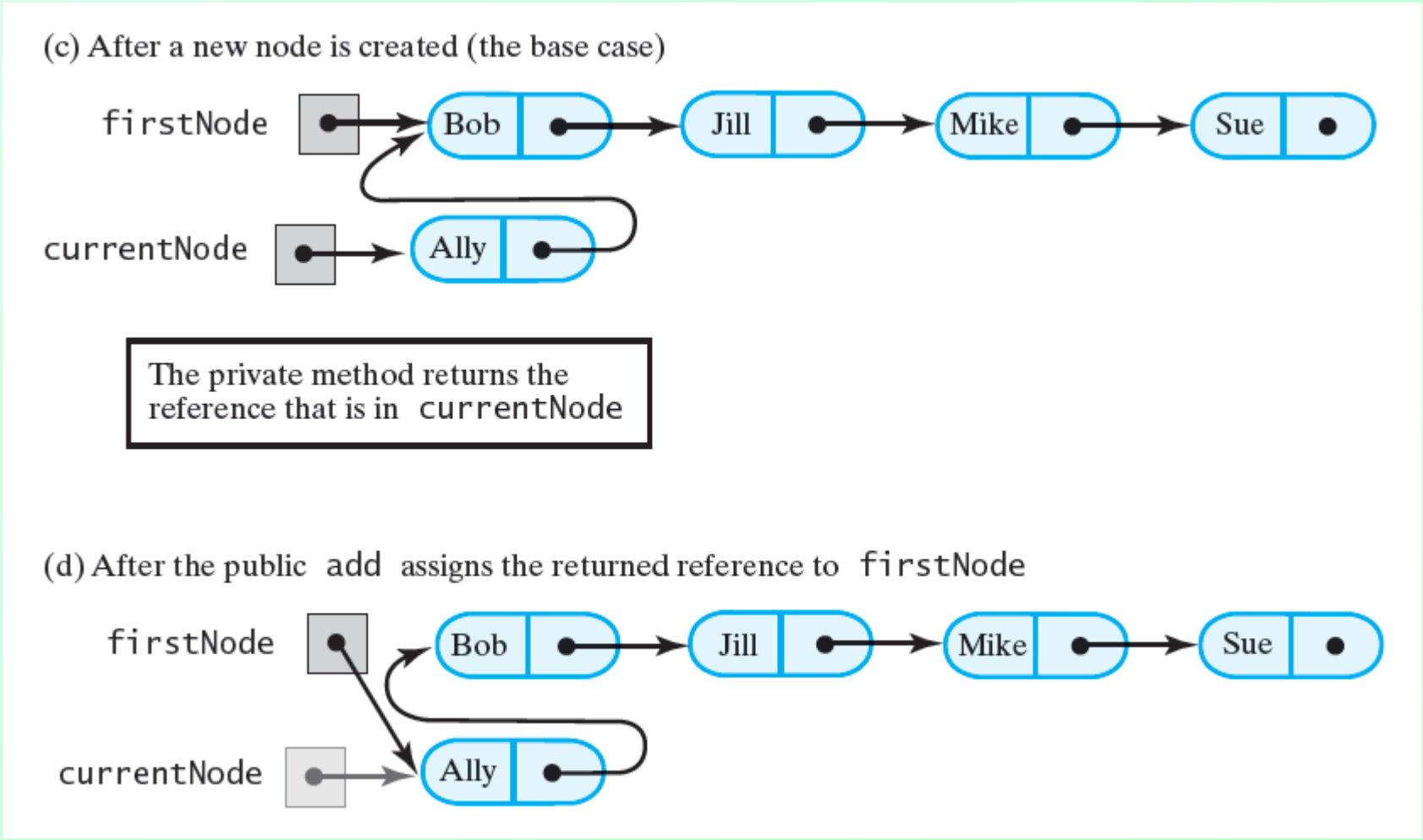


Figure 16-3 Recursively adding a node at the beginning of a chain

Copyright ©2012 by Pearson Education, Inc. All rights reserved

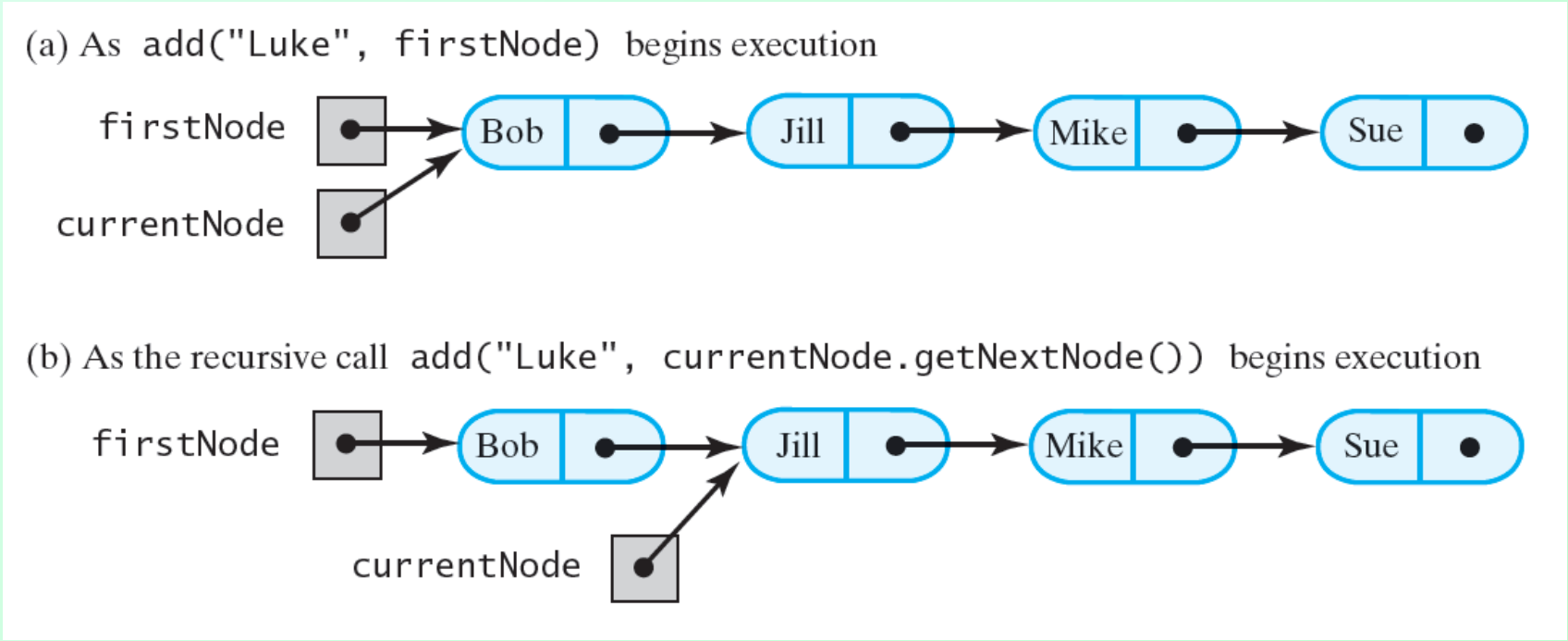
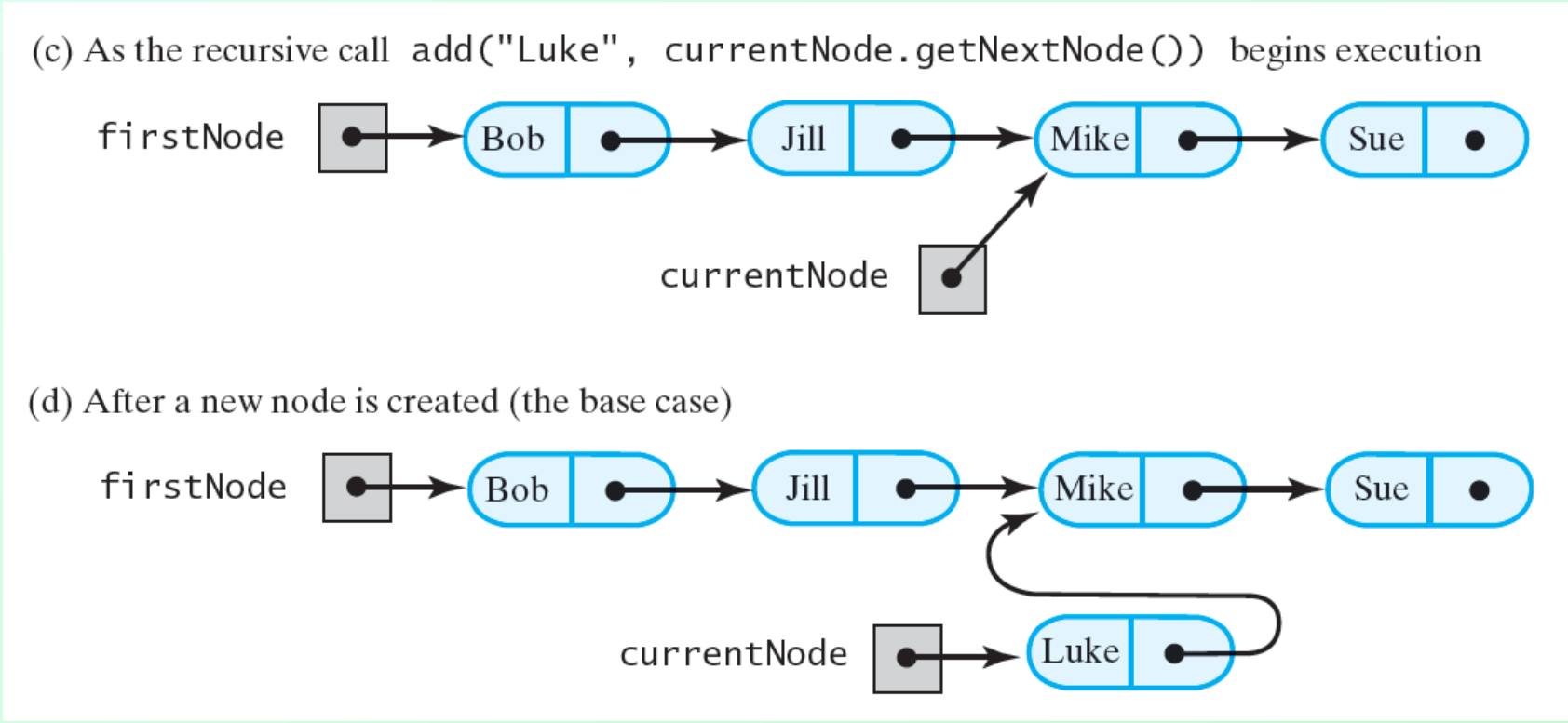


Figure 16-4 Recursively adding a node between existing nodes in a chain

Copyright ©2012 by Pearson Education, Inc. All rights reserved



The private method returns the reference that is in `currentNode`

Figure 16-4 Recursively adding a node between existing nodes in a chain

Copyright ©2012 by Pearson Education, Inc. All rights reserved

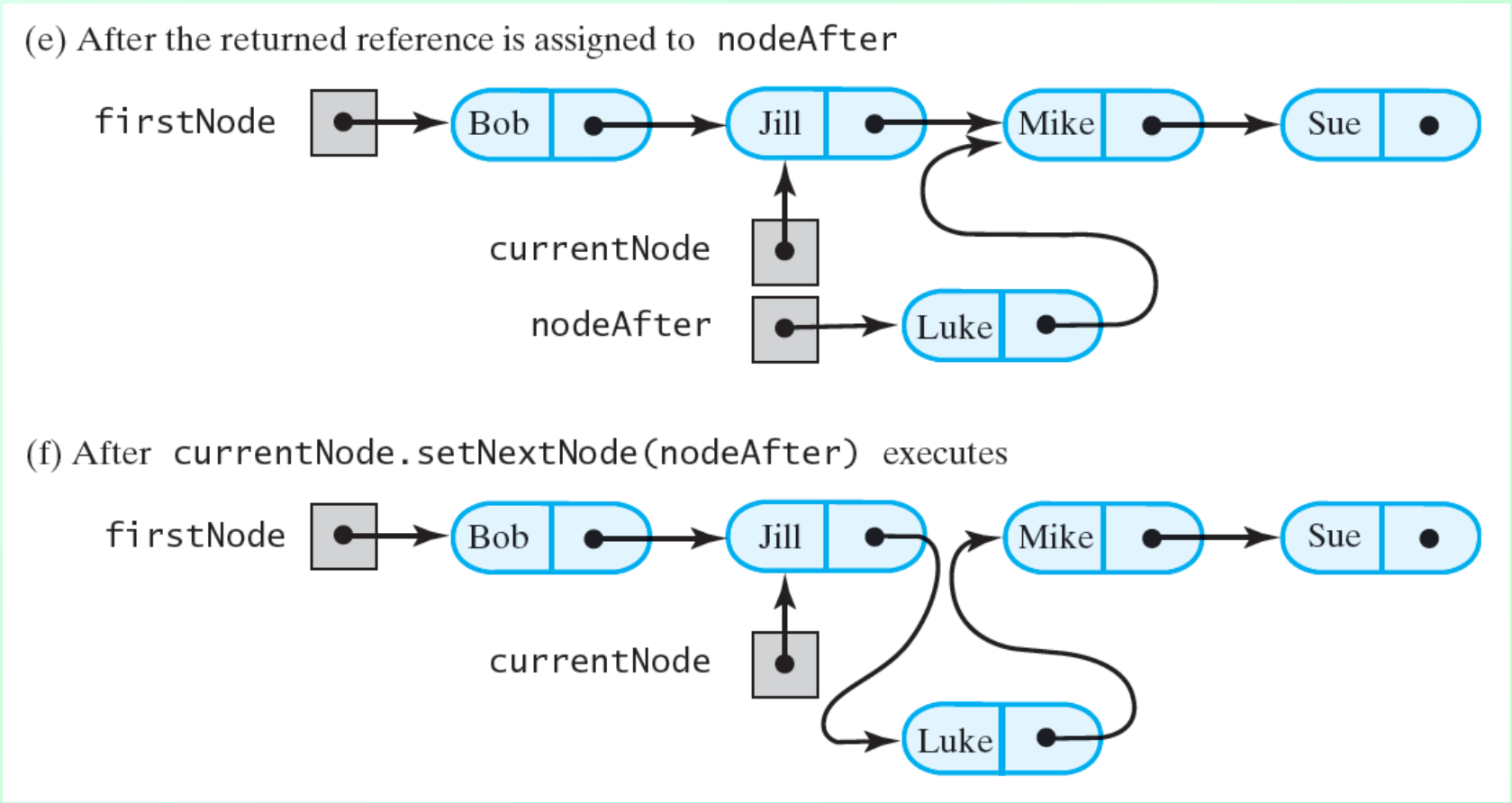


Figure 16-4 Recursively adding a node between existing nodes in a chain

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 8 The linked implementation of the ADT sorted list, as given in this chapter, does not maintain a tail reference. Why is a tail reference more significant for a linked implementation of the ADT list than it is for a sorted list?

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 8 The linked implementation of the ADT sorted list, as given in this chapter, does not maintain a tail reference. Why is a tail reference more significant for a linked implementation of the ADT list than it is for a sorted list?

The method add(newEntry) for a list adds a new entry at the end of the list. A tail reference makes this method O(1) instead of O(n). For a sorted list, add(newEntry) must traverse the chain to locate the point of insertion. If the insertion is at the end of the chain, the traversal will give you a reference to the last node. A separate tail reference is not needed.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Efficiency of the Linked Implementation

ADT Sorted List Operation	Array	Linked
`add(newEntry)`	O(n)	O(n)
`remove(anEntry)`	O(n)	O(n)
`getPosition(anEntry)`	O(n)	O(n)
`getEntry(givenPosition)`	O(1)	O(n)
`contains(anEntry)`	O(n)	O(n)
`remove(givenPosition)`	O(n)	O(n)
`toArray()`	O(n)	O(n)
`clear(), getLength(), isEmpty()`	O(1)	O(1)

FIGURE 16-5 The worst-case efficiencies of the operations on the ADT sorted list for two implementations

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Implementation That Uses the ADT List

- Sorted list a natural application for ADT list
- Possible ways
  - Use list as data field within class that implements sorted list
  - Use inheritance to derive sorted list from list
- Our class **`SortedList`** will implement the interface **`SortedListInterface`**
- View source code, Listing 16-A

Copyright ©2012 by Pearson Education, Inc. All rights reserved

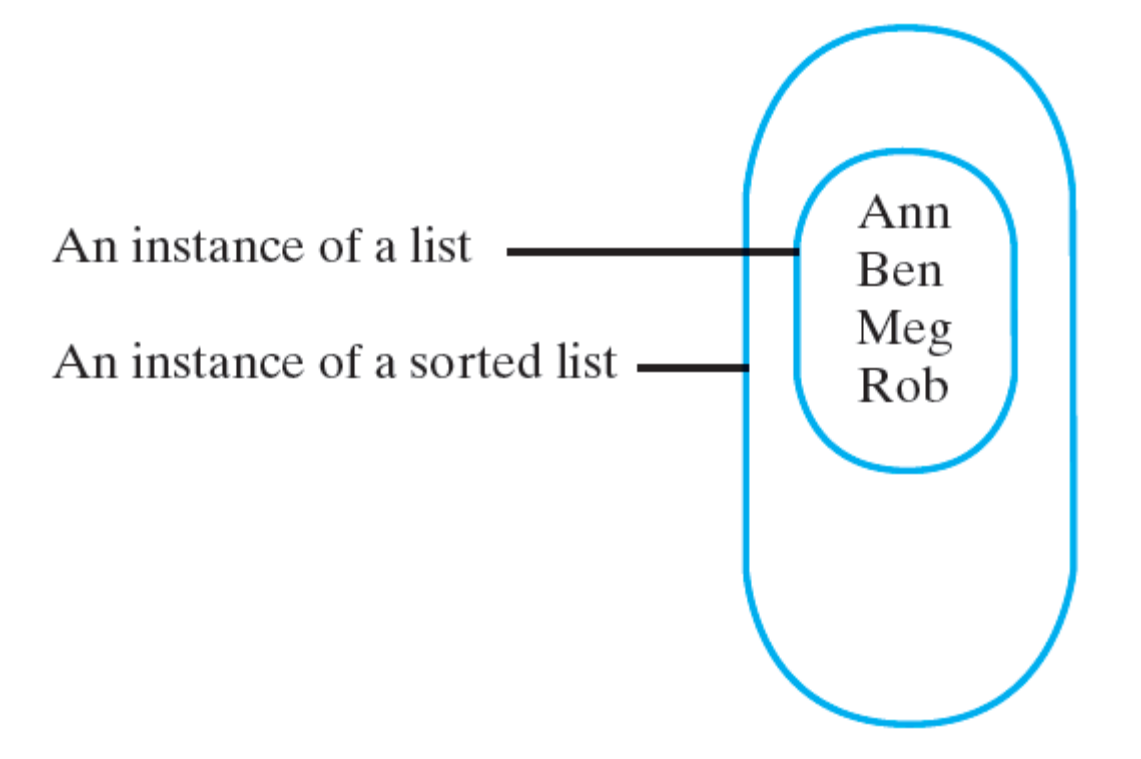


Figure 16-6 An instance of a sorted list that contains a list of its entries

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 11 If a sorted list contains five duplicate objects and you use the below method remove to remove one of them, what will be removed from the list: the first occurrence of the object, the last occurrence of the object, or all occurrences of the object?

```
public boolean remove(T anEntry)
{   boolean result = false;
    int  position = getPosition(anEntry);
    if  (position > 0)
    {   list.remove(position);
        result = true;
    }
    return  result;
}
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 11 If a sorted list contains five duplicate objects and you use the below method remove to remove one of them, what will be removed from the list: the first occurrence of the object, the last occurrence of the object, or all occurrences of the object?

```
public boolean remove(T anEntry)
{   boolean result = false;
    int  position = getPosition(anEntry);
    if  (position > 0)
    {   list.remove(position);
        result = true;
    }
    return  result;
}
```

The first occurrence of the object. Note that getPosition returns the position of the first occurrence of the entry within the list.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

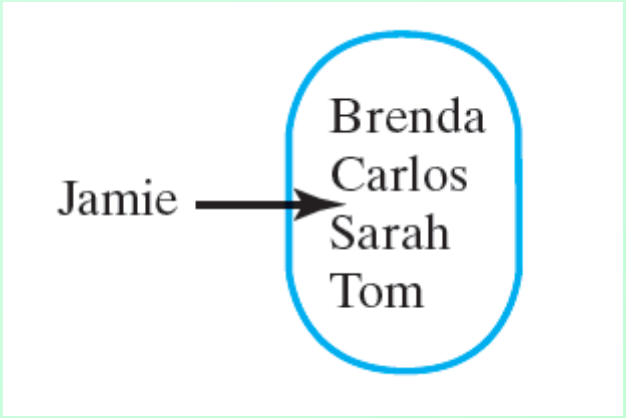


Figure 16-7 A sorted list in which *Jamie* belongs after *Carlos* but before *Sarah*

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Efficiency Issues

ADT List Operation	Array	Linked
`getEntry(givenPosition)`	O(1)	O(*n*)
`add(newPosition, newEntry)`	O(*n*)	O(*n*)
`remove(givenPosition)`	O(*n*)	O(*n*)
`contains(anEntry)`	O(*n*)	O(*n*)
`toArray()`	O(*n*)	O(*n*)
`clear()`, `getLength()`, `isEmpty()`	O(1)	O(1)

Figure 16-8 The worst-case efficiencies of selected ADT list operations for array-based and linked implementations

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 15 Give an advantage and a disadvantage of using composition in the implementation of the class SortedList.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 15 Give an advantage and a disadvantage of using composition in the implementation of the class SortedList.

Advantage: The implementation is easy to write.

Disadvantage: The implementation is not efficient when the implementation of the underlying list is linked.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

ADT Sorted List Operation	List Implementation	
	Array	Linked
`add(new Entry)`	$O(n)$	$O(n^2)$
`remove(anEntry)`	$O(n)$	$O(n^2)$
`getPosition(anEntry)`	$O(n)$	$O(n^2)$
`getEntry(givenPosition)`	$O(1)$	$O(n)$
`contains(anEntry)`	$O(n)$	$O(n)$
`remove(givenPosition)`	$O(n)$	$O(n)$
`toArray()`	$O(n)$	$O(n)$
`clear(), getLength(), isEmpty()`	$O(1)$	$O(1)$

Figure 16-9 The worst-case efficiencies of the ADT sorted list operations when implemented using an instance of the ADT list

Copyright ©2012 by Pearson Education, Inc. All rights reserved

End

Chapter 16

Copyright ©2012 by Pearson Education, Inc. All rights reserved