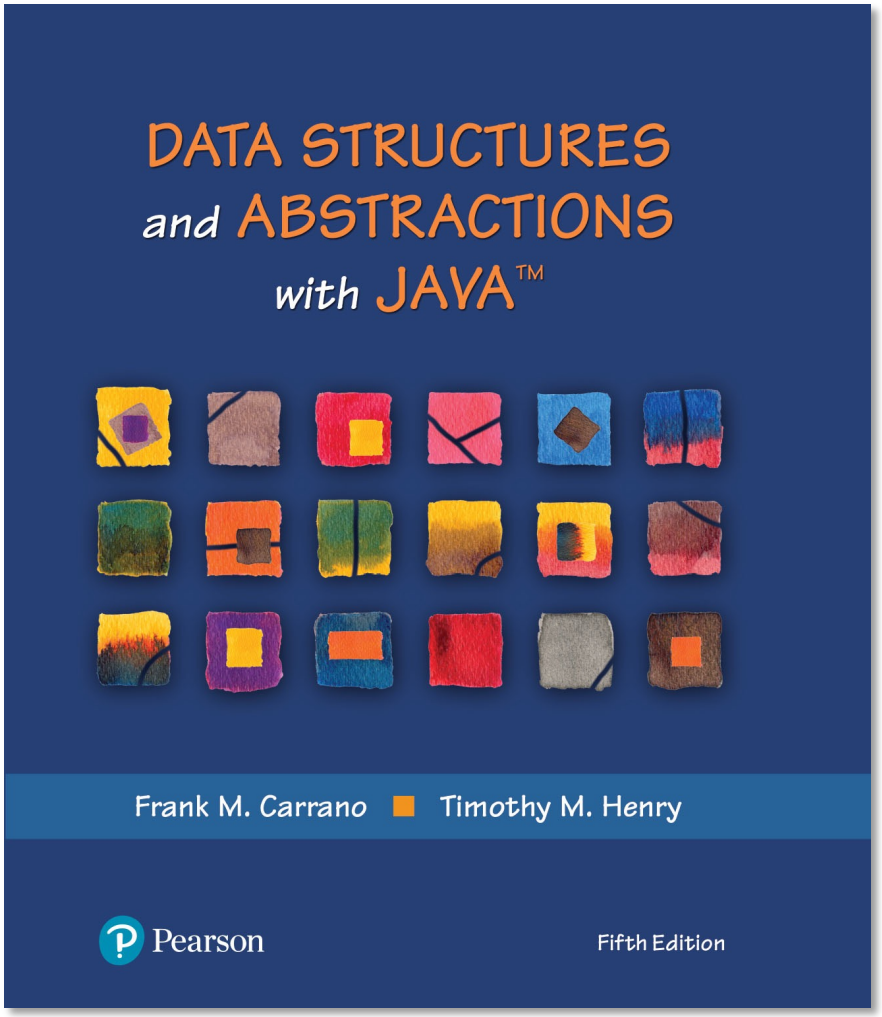
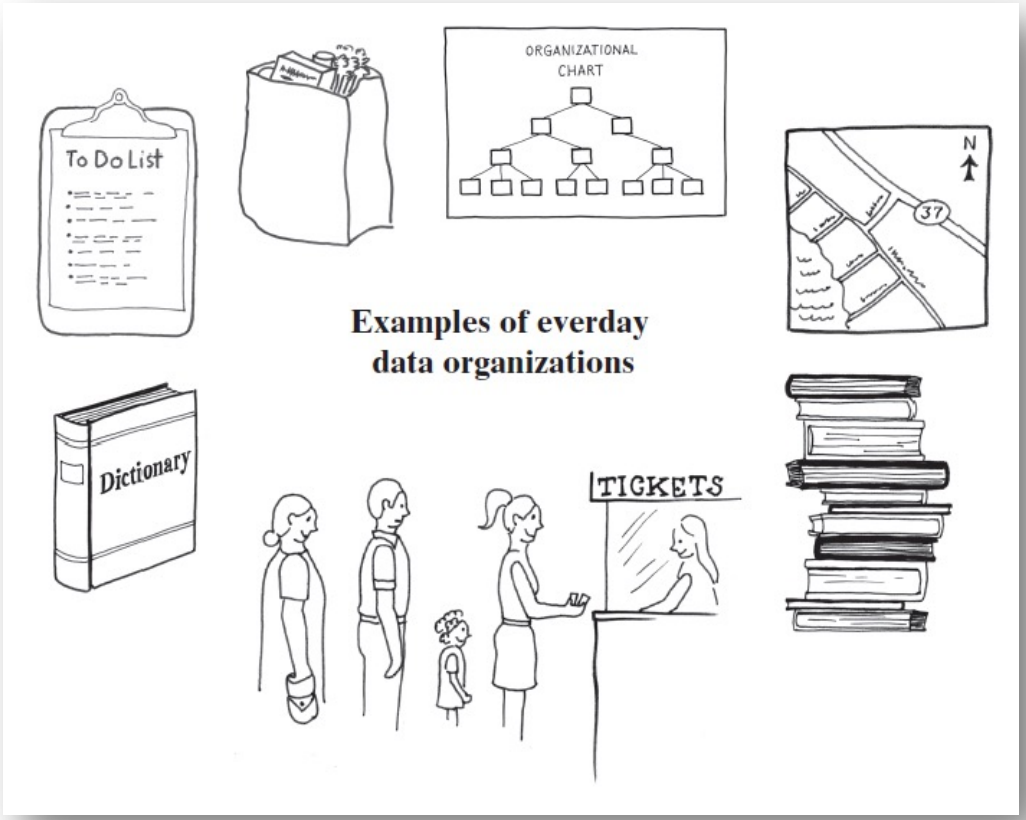
Data Structures



Introduction

Data Organization in Life

- Standing in a line
- Stack of books
- To-Do list
- Dictionary
- Folders, directories on your computer
- Road map



Computer Data Organization

- Abstract Data Type: ADT
- Data Structure
- Collection
- Examples of containers
  - Bag
  - List
  - Stack
  - Queue
  - Dictionary
  - Tree
  - Graph

Problems with Array Implementation

- Array has fixed size
- May become full
- Alternatively may have wasted space
- Resizing is possible but requires overhead of time

Analogy

- Empty classroom
- Numbered desks stored in hallway
  - Number on back of desk is the “address”
- Number on desktop references another desk in chain of desks
- Desks are linked by the numbers

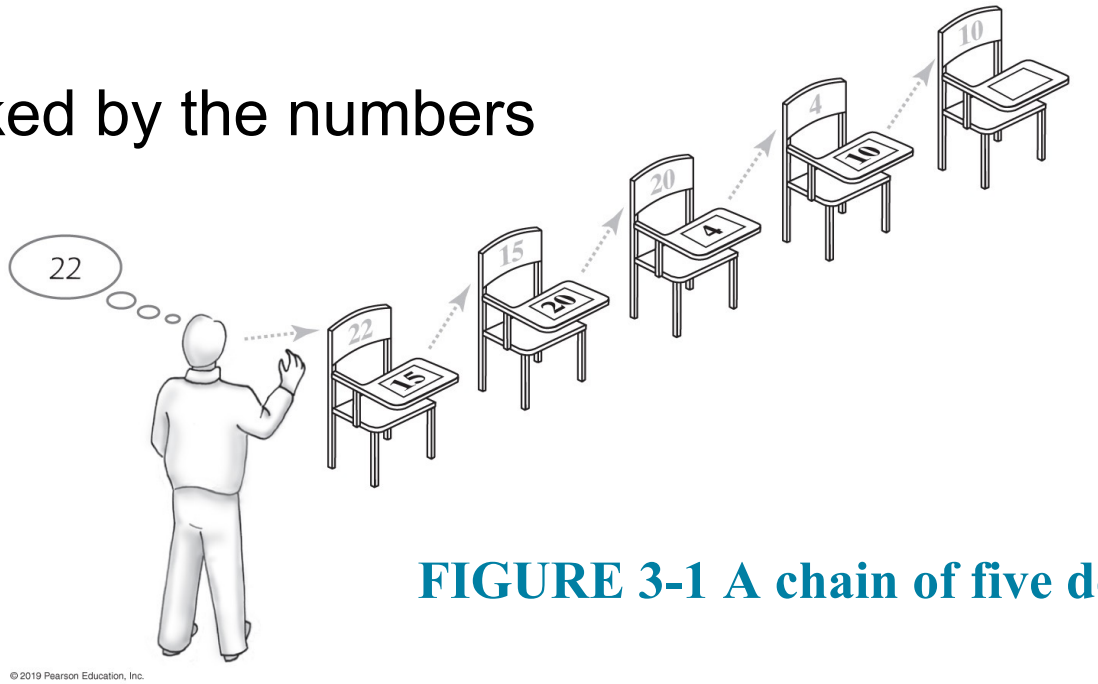


FIGURE 3-1 A chain of five desks

Forming a Chain by Adding to Its Beginning

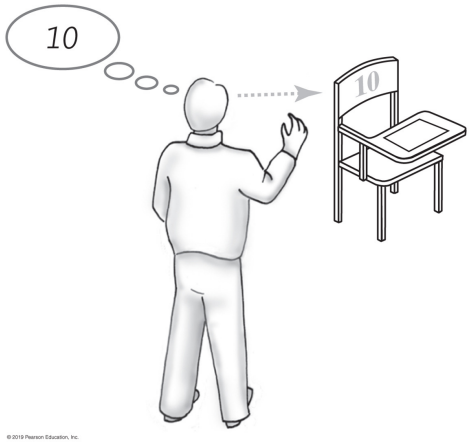


FIGURE 3-2
One desk in the room

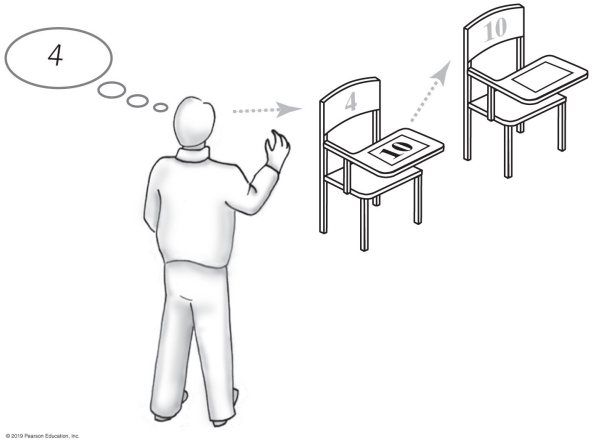


FIGURE 3-3
Two linked desks, with the newest desk first

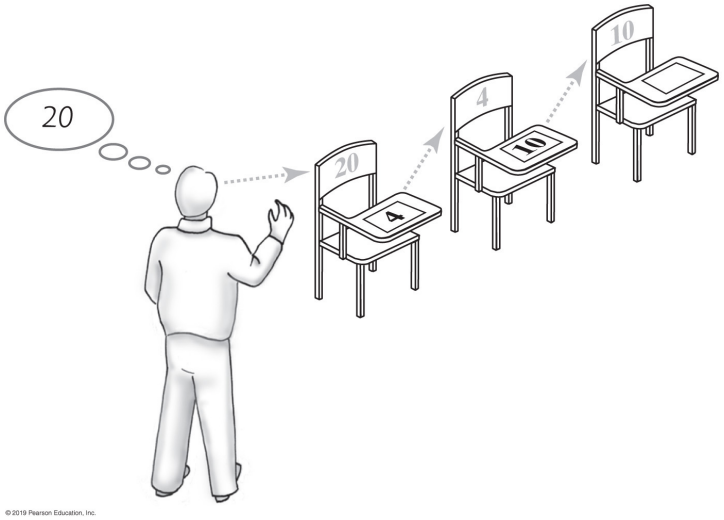


FIGURE 3-4
Three linked desks, with the newest desk first

What Is an Iterator?

- An object that traverses a collection of data
- During iteration, each data item is considered once
  - Possible to modify item as accessed
- Should implement as a distinct class that interacts with the ADT

Beginning a Chain of Nodes

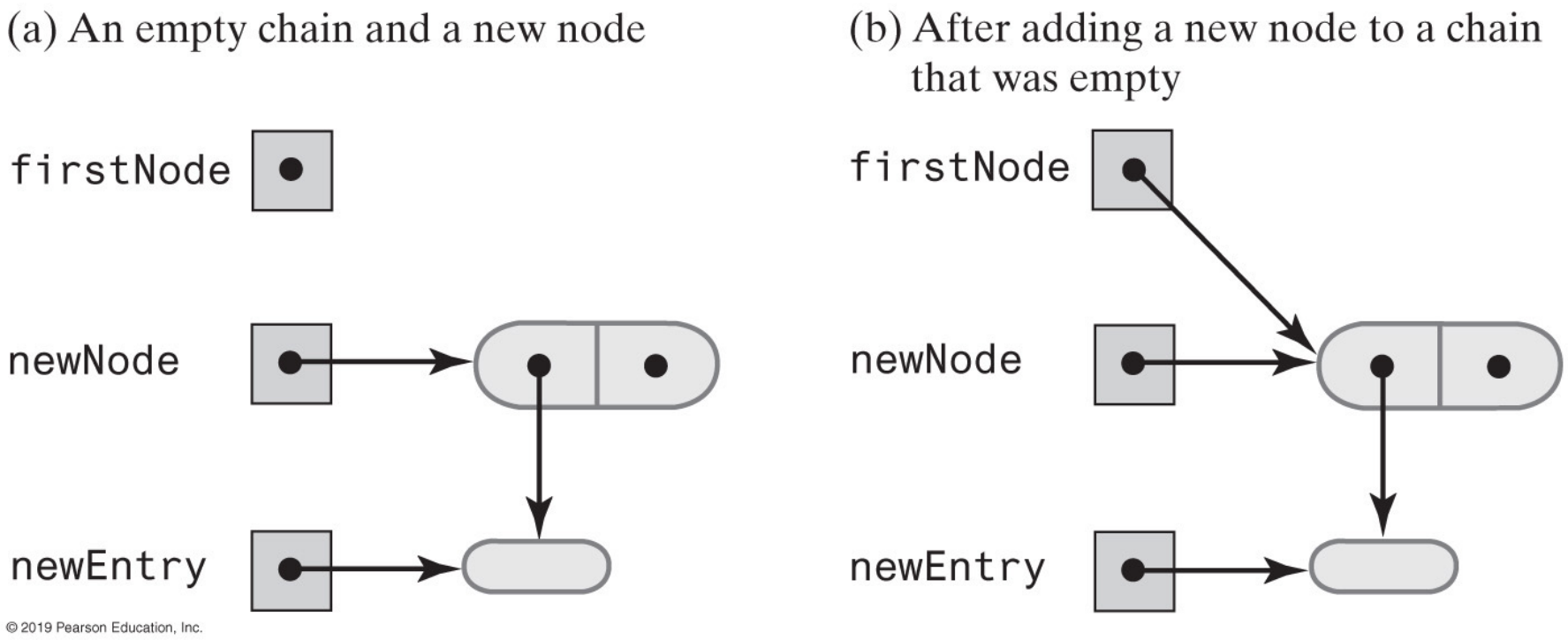


FIGURE 3-6 Adding a new node to an empty chain

Beginning a Chain of Nodes

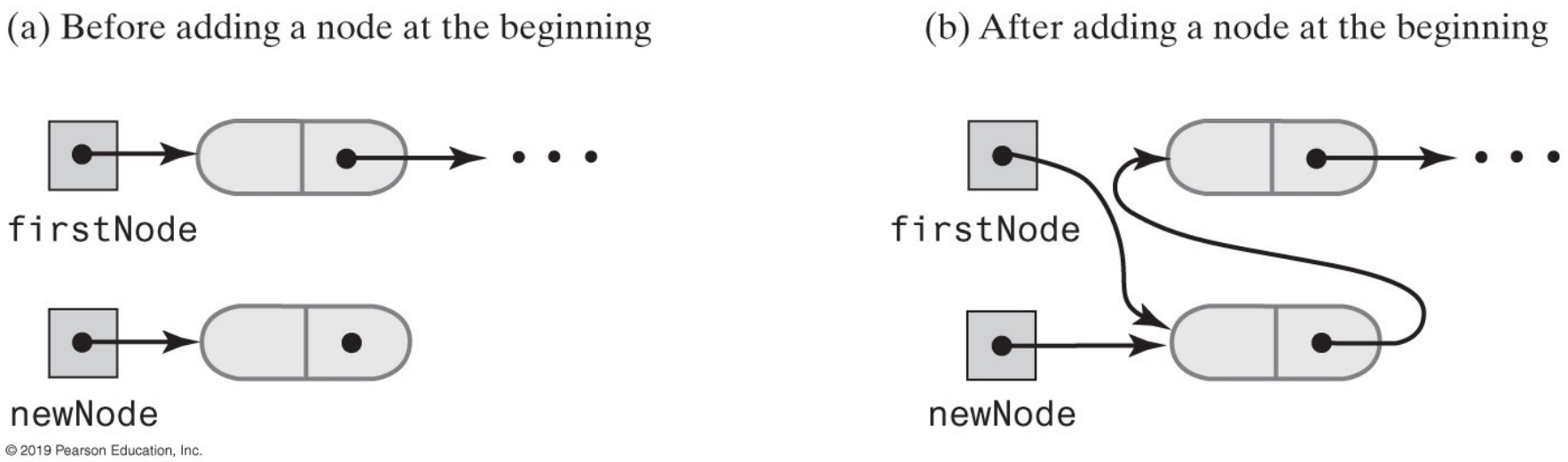


FIGURE 3-7 A chain of nodes just before and just after adding a node at the beginning

Removing an Item from a Linked Chain

- **Case 1:**
  - Your desk is first in the chain of desks.
- **Case 2:**
  - Your desk is not first in the chain of desks.

Removing an Item from a Linked Chain

- **Case 1**
  - Locate first desk by asking instructor for its address.
  - Give address written on the first desk to instructor.
    - This is address of second desk in chain.
  - Return first desk to hallway.

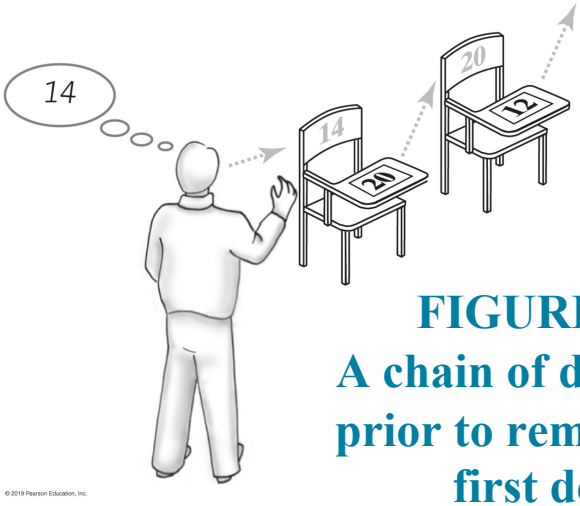


FIGURE 3-8
A chain of desks just prior to removing its first desk

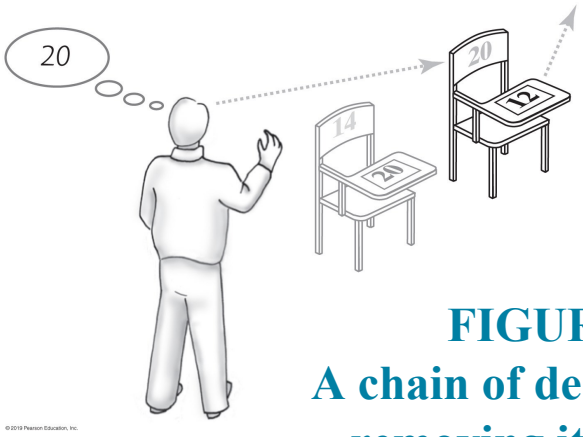


FIGURE 3-9
A chain of desks just after removing its first desk

Removing an Item from a Linked Chain

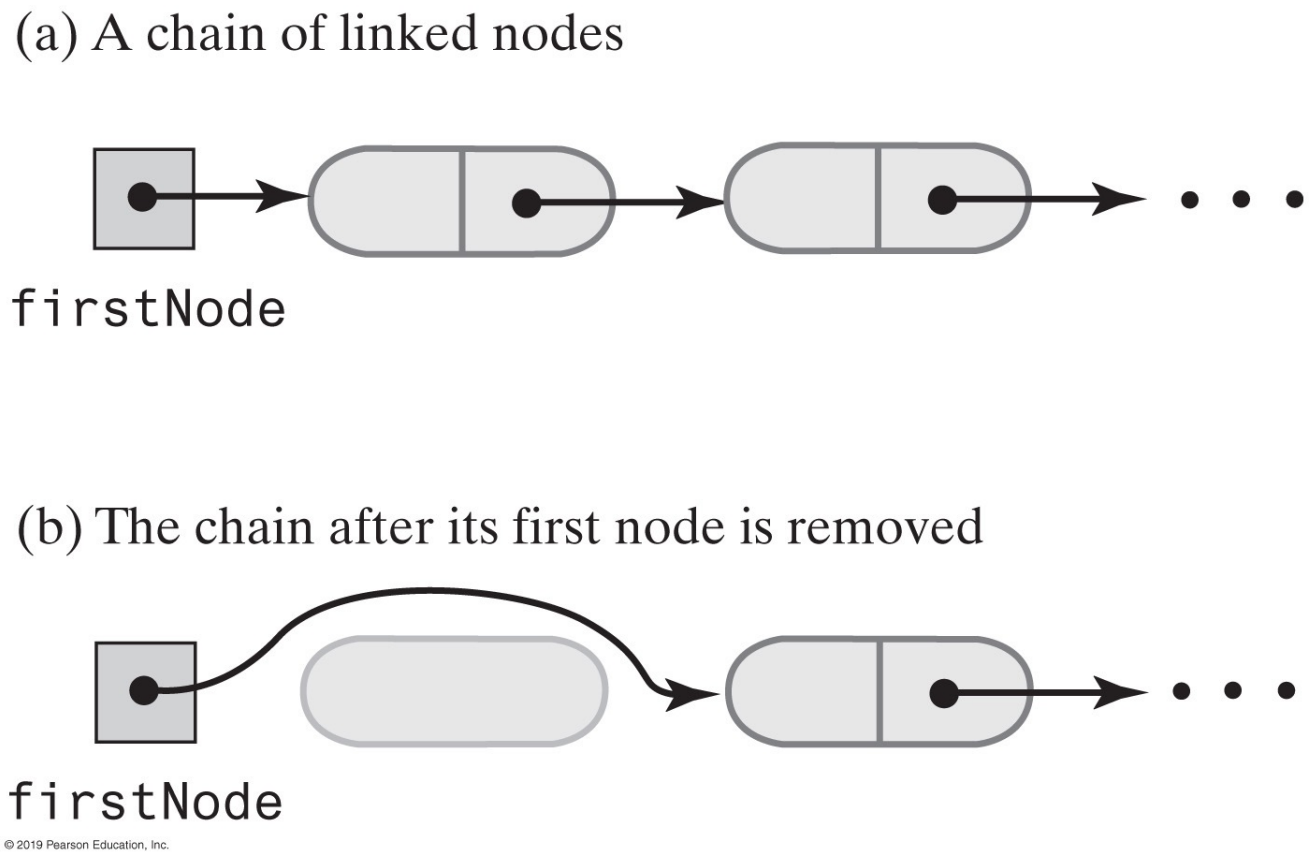


FIGURE 3-10 A chain of nodes just before and just after its first node is removed

Removing an Item from a Linked Chain

- **Case 2**
  - Move the student in the first desk to your former desk.
  - Remove the first desk using the steps described for Case 1.

Pros of Using a Chain

- Bag can grow and shrink in size as necessary.
- Remove and recycle nodes that are no longer needed
- Adding new entry to end of array or to beginning of chain both relatively simple
- Similar for removal

Cons of Using a Chain

- Removing specific entry requires search of array or chain
- Chain requires more memory than array of same length

Advantages of Linked Implementation

- Uses memory only as needed
- When entry removed, unneeded memory returned to system
- Avoids moving data when adding or removing entries

Adding a Node at Various Positions

- Possible cases:
  - Chain is empty
  - Adding node at chain's beginning
  - Adding node between adjacent nodes
  - Adding node to chain's end

Adding a Node

This pseudocode establishes a new node for the given data

newNode *references a new instance of* Node

Place newEntry *in* newNode

firstNode = *address of* newNode

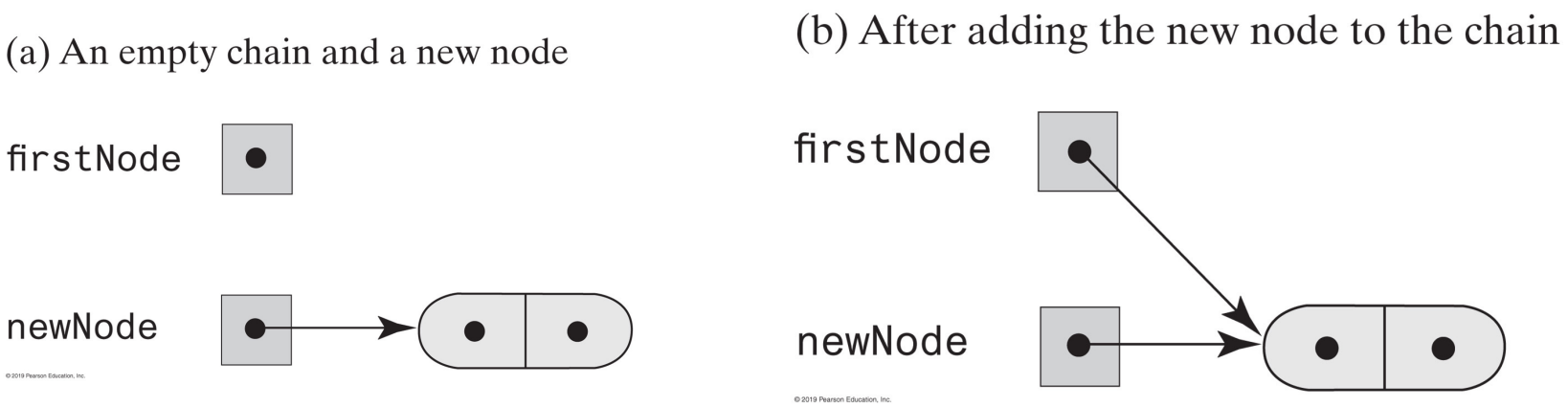


FIGURE 12-1 Adding a node to an empty chain

Adding a Node

This pseudocode describes the steps needed to add a node to the beginning of a chain.

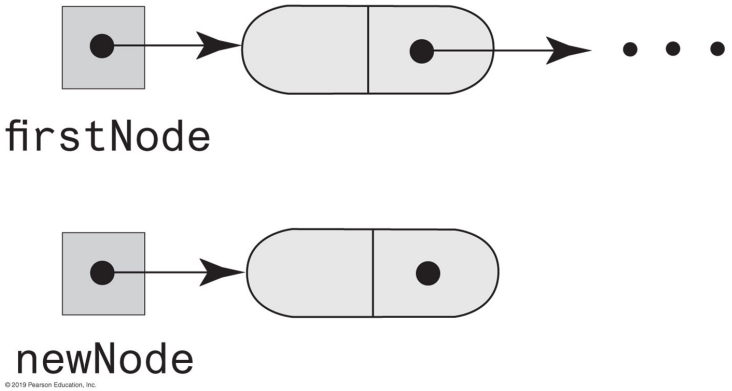
newNode *references a new instance of* Node

Place newEntry *in* newNode

Set newNode*'s link to* firstNode

Set firstNode *to* newNode

(a) A chain of nodes and a new node



(b) After adding the new node to the beginning of the chain

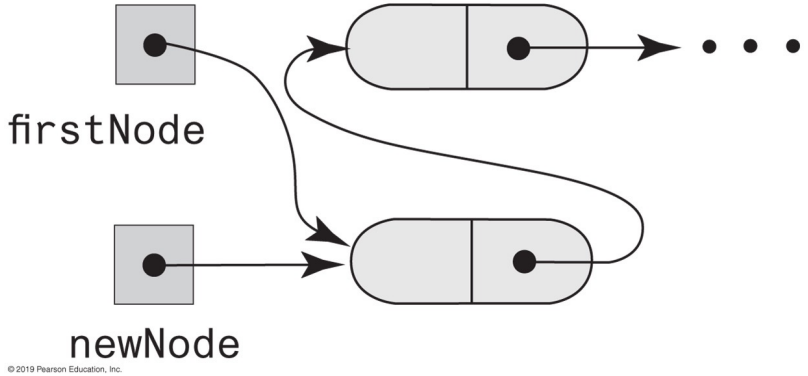


FIGURE 12-2 Adding a node to the beginning of a chain

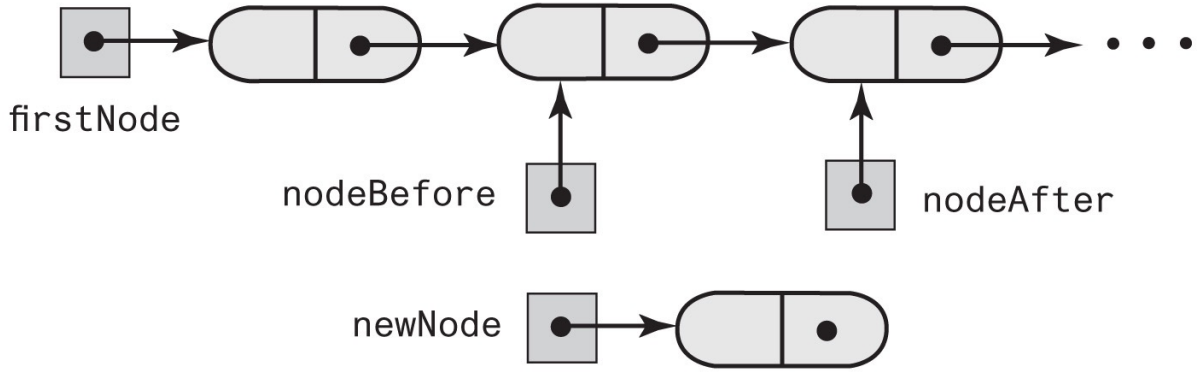
Adding a Node

Pseudocode to add a node to a chain between two existing, consecutive nodes

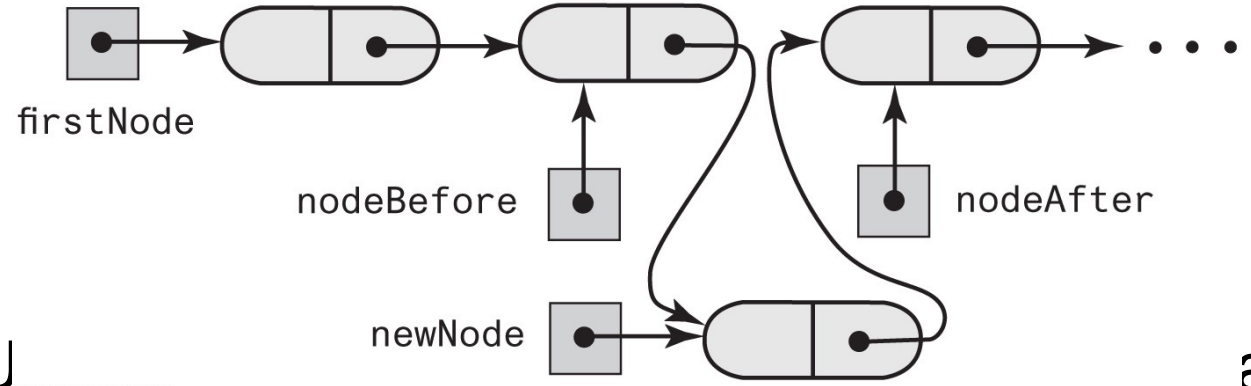
newNode *references the new node*
Place newEntry *in* newNode
Let nodeBefore *reference the node that will be before the new node*
Set nodeAfter *to* nodeBefore*'s link*
Set newNode*'s link to* nodeAfter
Set nodeBefore*'s link to* newNode

Adding a Node

(a) A chain of nodes and a new node



(b) After adding the new node between adjacent nodes



FIGU acent nodes

Adding a Node

Steps to add a node at the end of a chain.

newNode *references a new instance of* Node

Place newEntry *in* newNode

Locate the last node in the chain

Place the address of newNode *in this last node*

Adding a Node

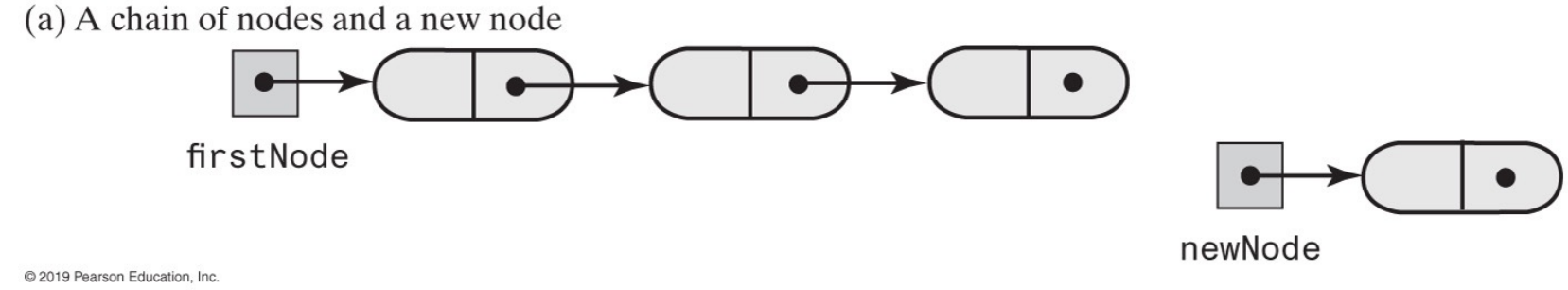


FIGURE 12-4 Adding a node to the end of a chain

Removing a Node

- Possible cases
  - Removing the first node
  - Removing a node other than first one

Removing a Node

Steps for removing the first node.

Set firstNode *to the link in the first node;* firstNode *now either references the second node or is* null *if the chain had only one node.*

Since all references to the first node no longer exist, the system automatically recycles the first node's memory.

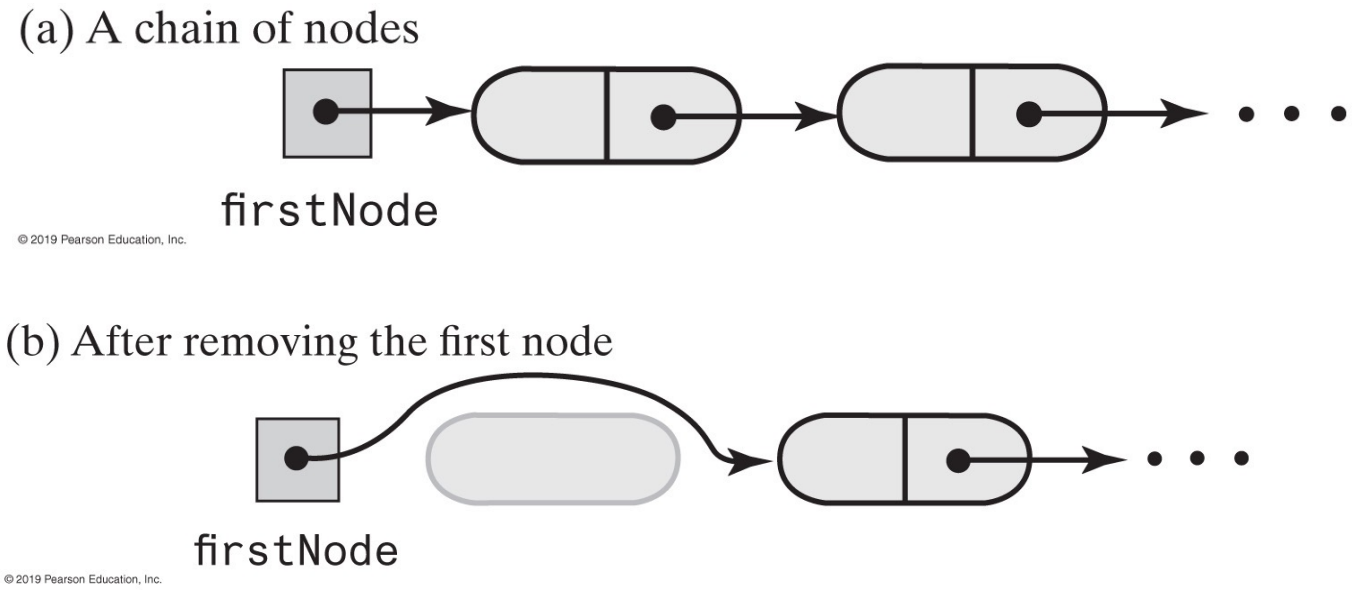


FIGURE 12-5 Removing the first node from a chain

Removing a Node

Removing a node other than the first one.

Let nodeBefore *reference the node before the one to be removed.*

Set nodeToRemove *to* nodeBefore*'s link;* nodeToRemove *now references the node to be removed.*

Set nodeAfter *to* nodeToRemove*'s link;* nodeAfter *now either references the node after the one to be removed or is* null.

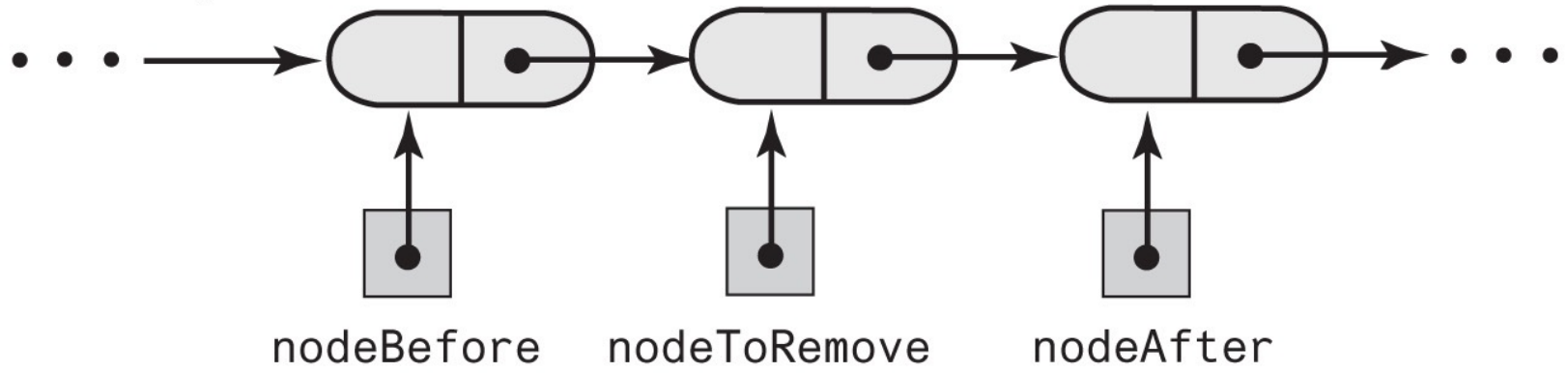
Set nodeBefore*'s link to* nodeAfter. *(*nodeToRemove *is now disconnected from the chain.)*

Set nodeToRemove *to* null.

Since all references to the disconnected node no longer exist, the system automatically recycles the node's memory.

Removing a Node

(a) After locating the node to remove



(b) After removing the node

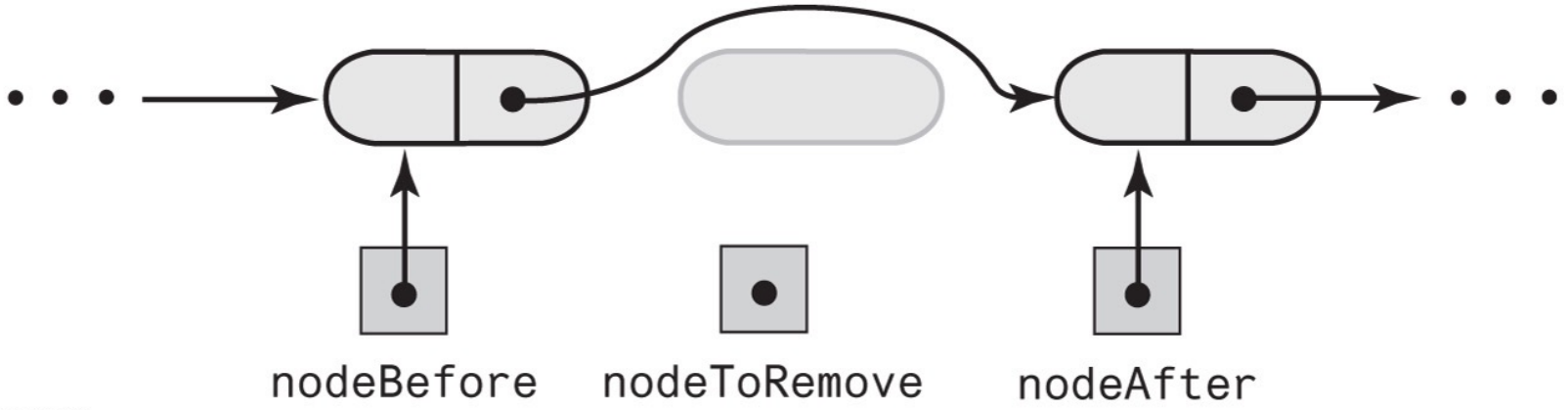
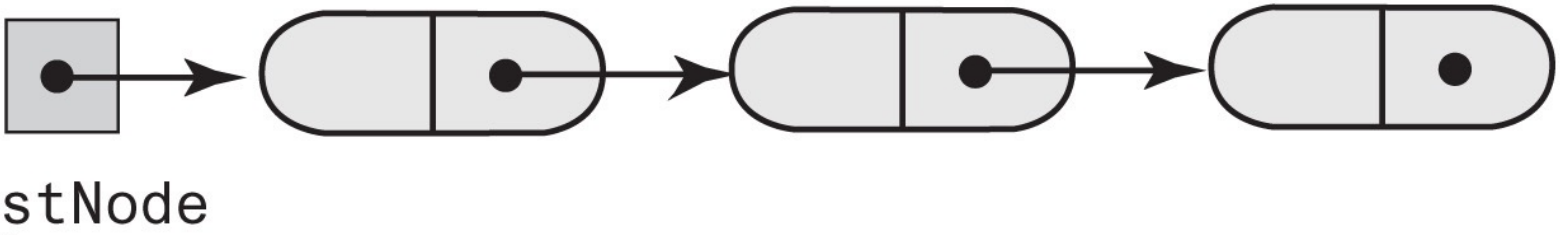


FIGURE 12-6 Removing an interior node from a chain

Using a Tail Reference

(a) With only a head reference



(b) With both a head reference and a tail reference

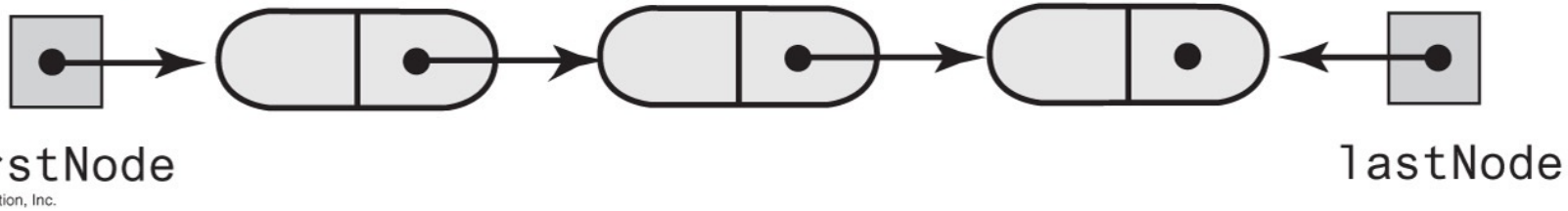


FIGURE 12-7 Two linked chains

A Refined Linked Implementation

```
private Node firstNode;        // Head reference to first node
private Node lastNode;         // Tail reference to last node
private int  numberOfEntries; // Number of entries in list
```

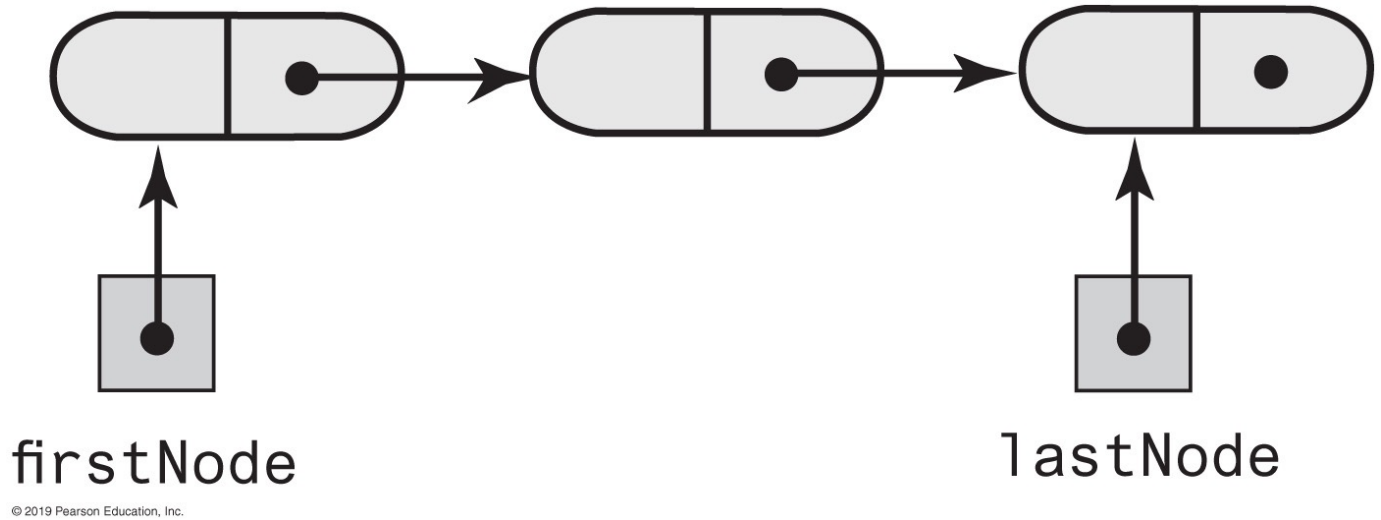
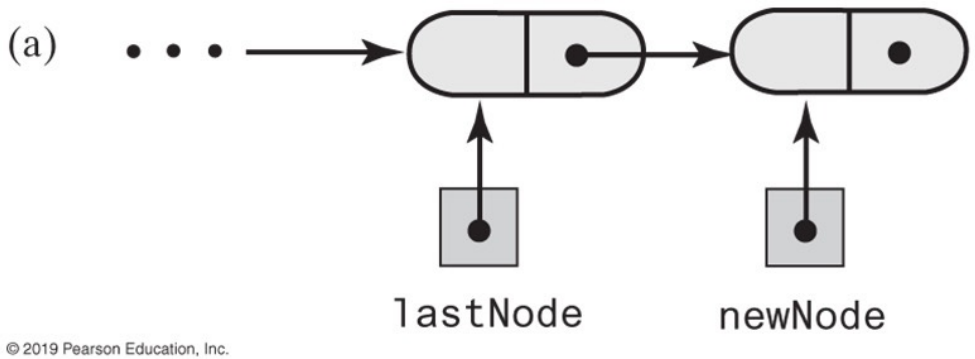
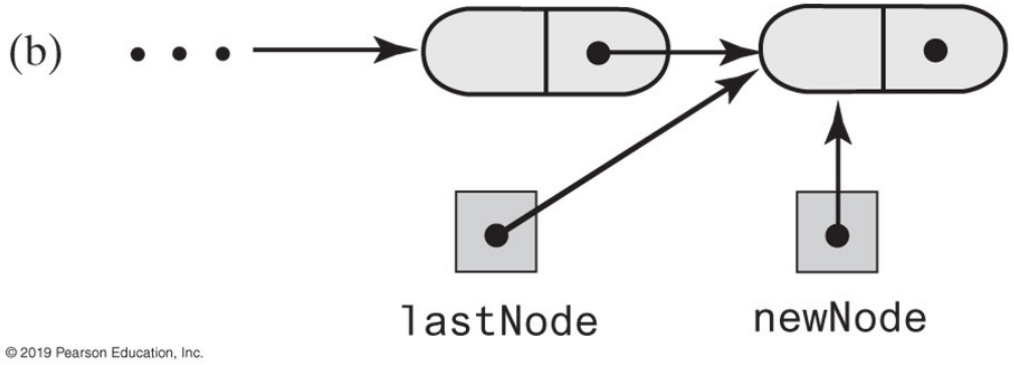


FIGURE 12-8 A linked chain with both a head reference and a tail reference

A Refined Linked Implementation



After executing
`lastNode.setNextNode(newNode);`



After executing
`lastNode = newNode;`

FIGURE 12-9 Adding a node to the end of a nonempty chain that has a tail reference

A Refined Linked Implementation

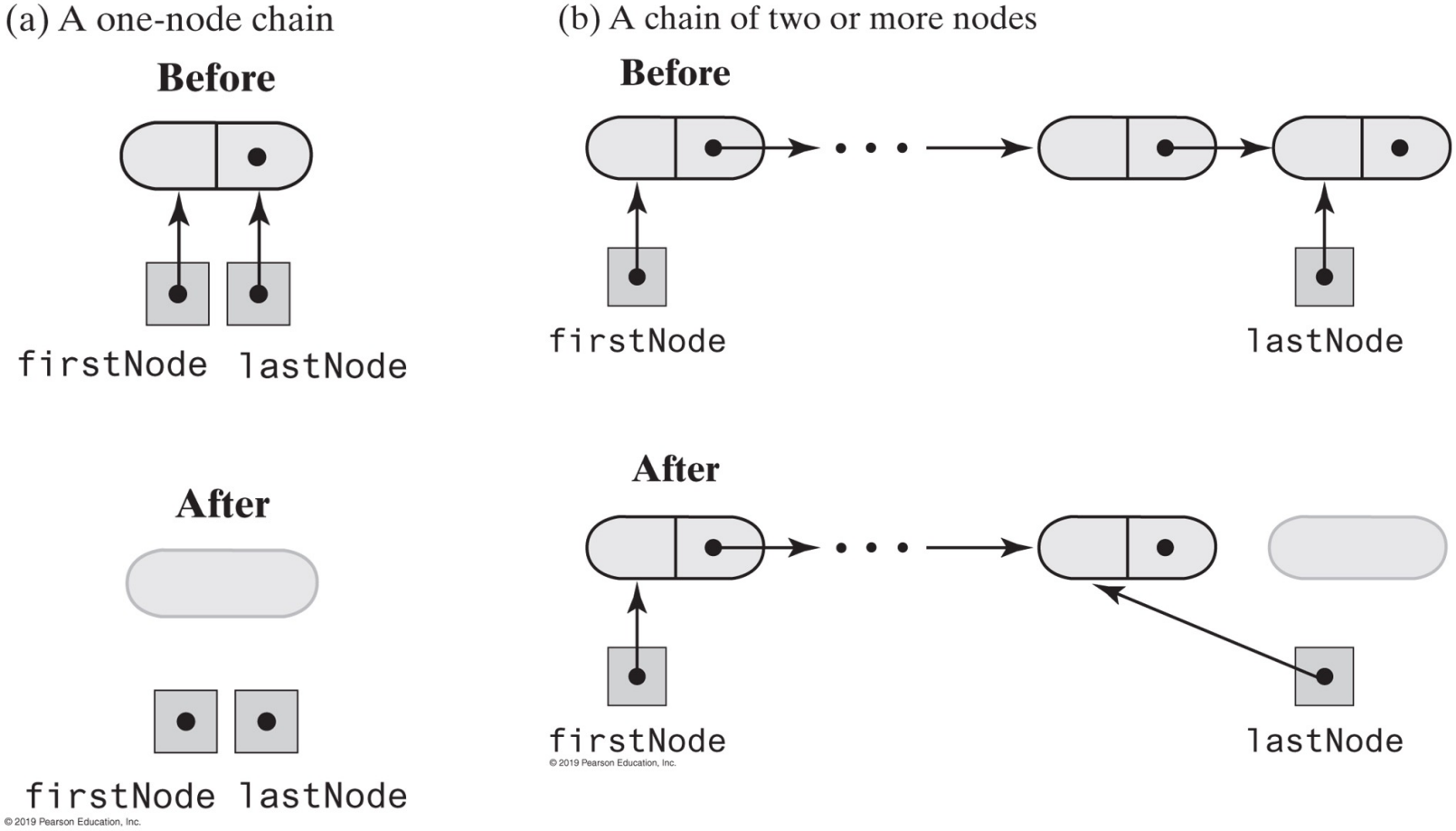


FIGURE 12-10 Before and after removing the last node from a chain that has both head and tail references and contains one or more nodes

Testing Core Methods

Program Output

```
Create an empty list.
List should be empty; isEmpty returns true.

Testing add to end:
List should contain 15 25 35 45.
The list contains 4 entries, as follows:
15 25 35 45
List should not be empty; isEmpty() returns false.

Testing clear():
List should be empty; isEmpty returns true.
```

LISTING 12-2 A main method that tests part of the implementation of the ADT list