A Bag Implementation that Links Data

Chapter 3

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

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

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 **top** of desk references another desk in chain of desks
- Desks are linked by the numbers

Analogy

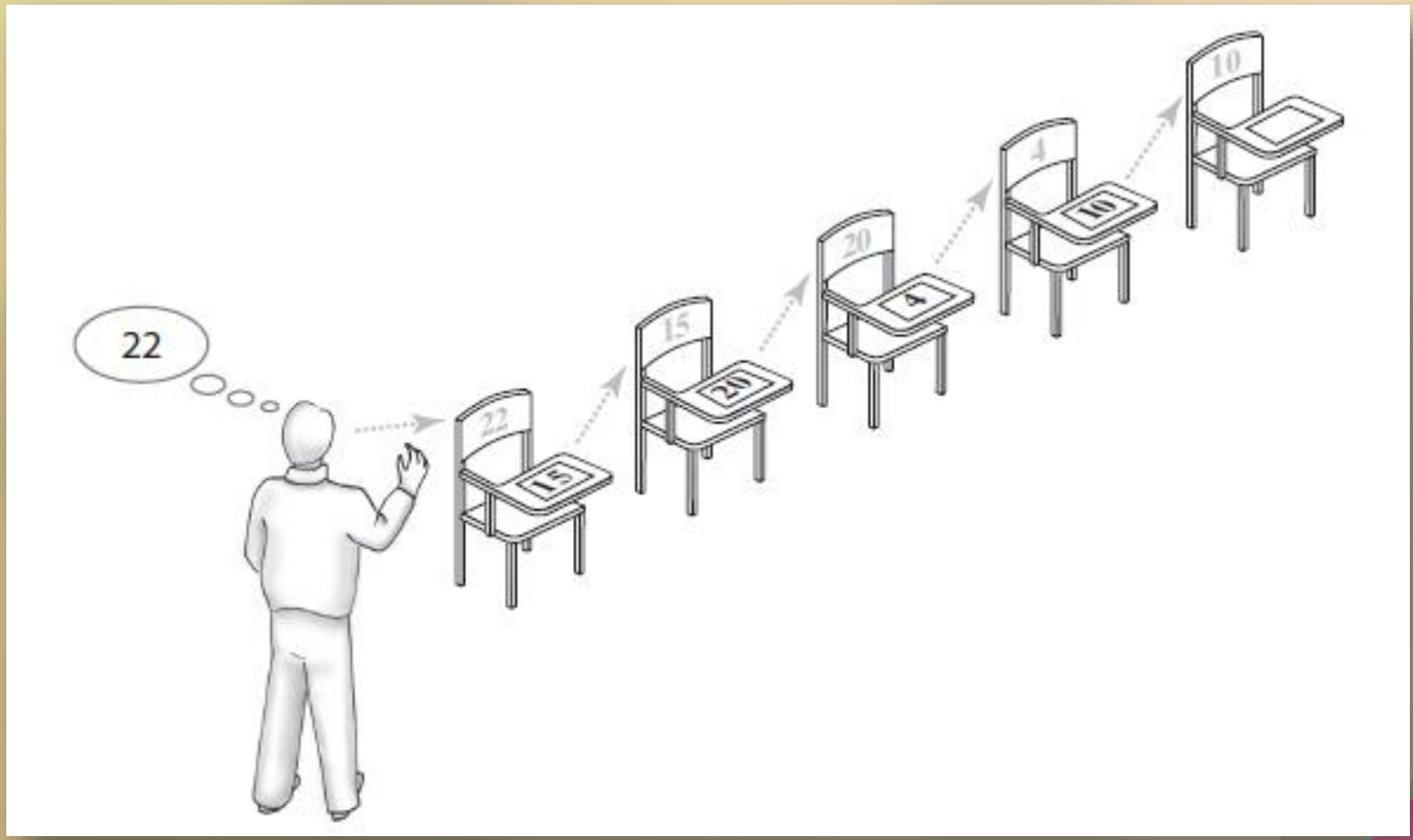

FIGURE 3-1 A chain of five desks

Forming a Chain by Adding to Its Beginning

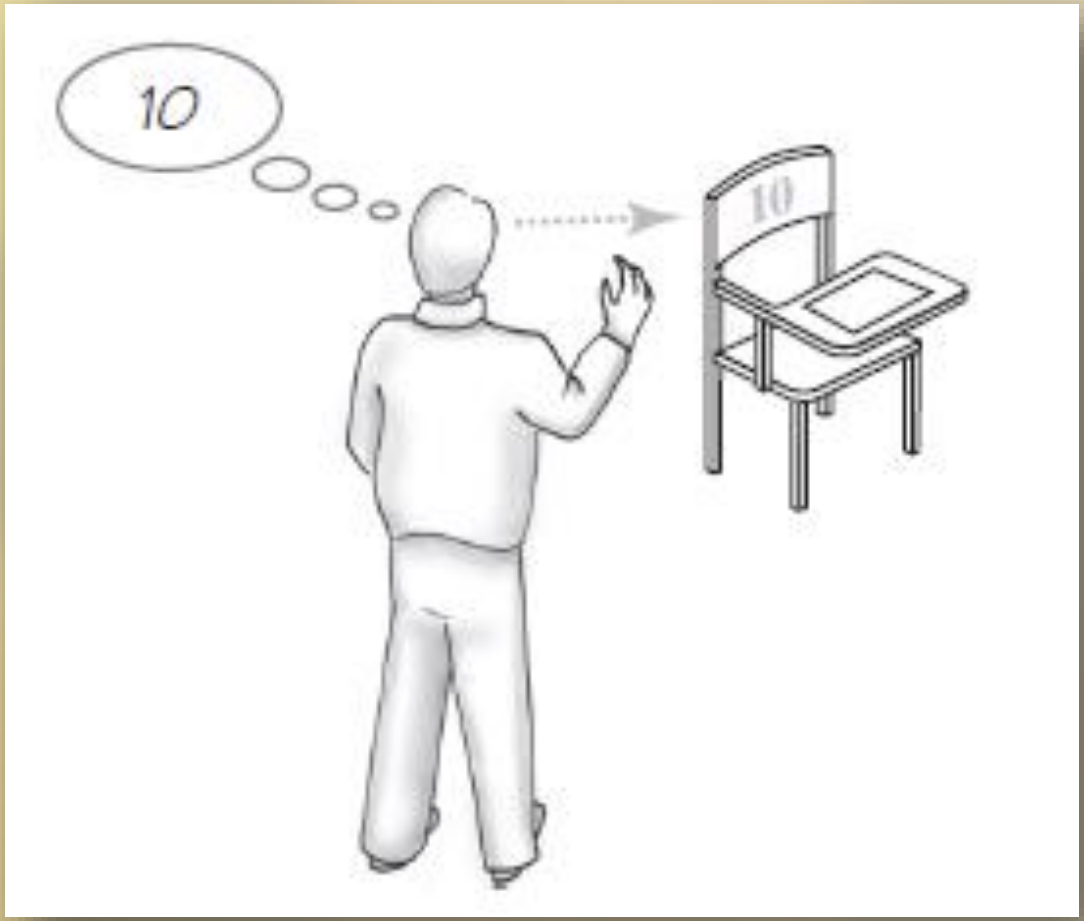

FIGURE 3-2 One desk in the room

Forming a Chain by Adding to Its Beginning

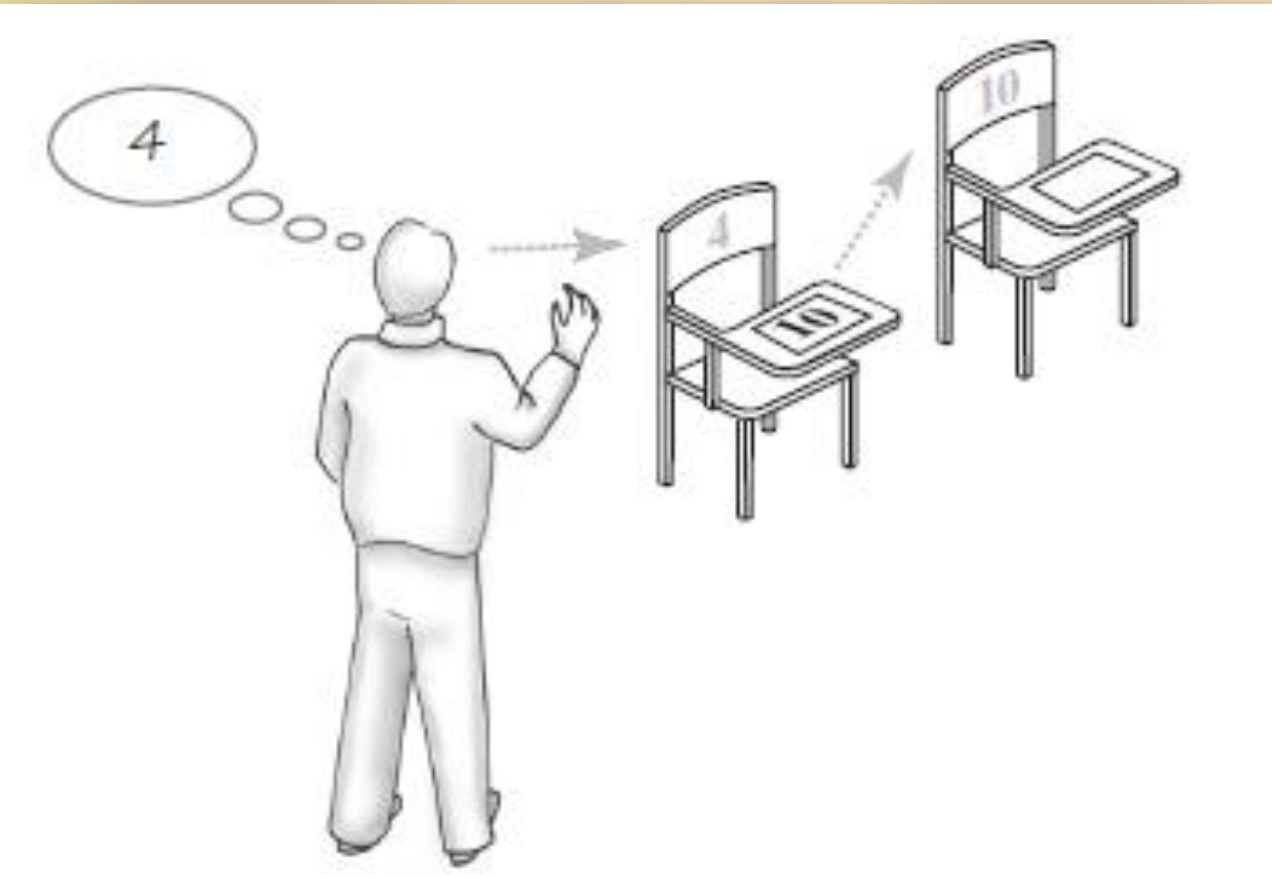

FIGURE 3-3 Two linked desks, with the newest desk first

Forming a Chain by Adding to Its Beginning

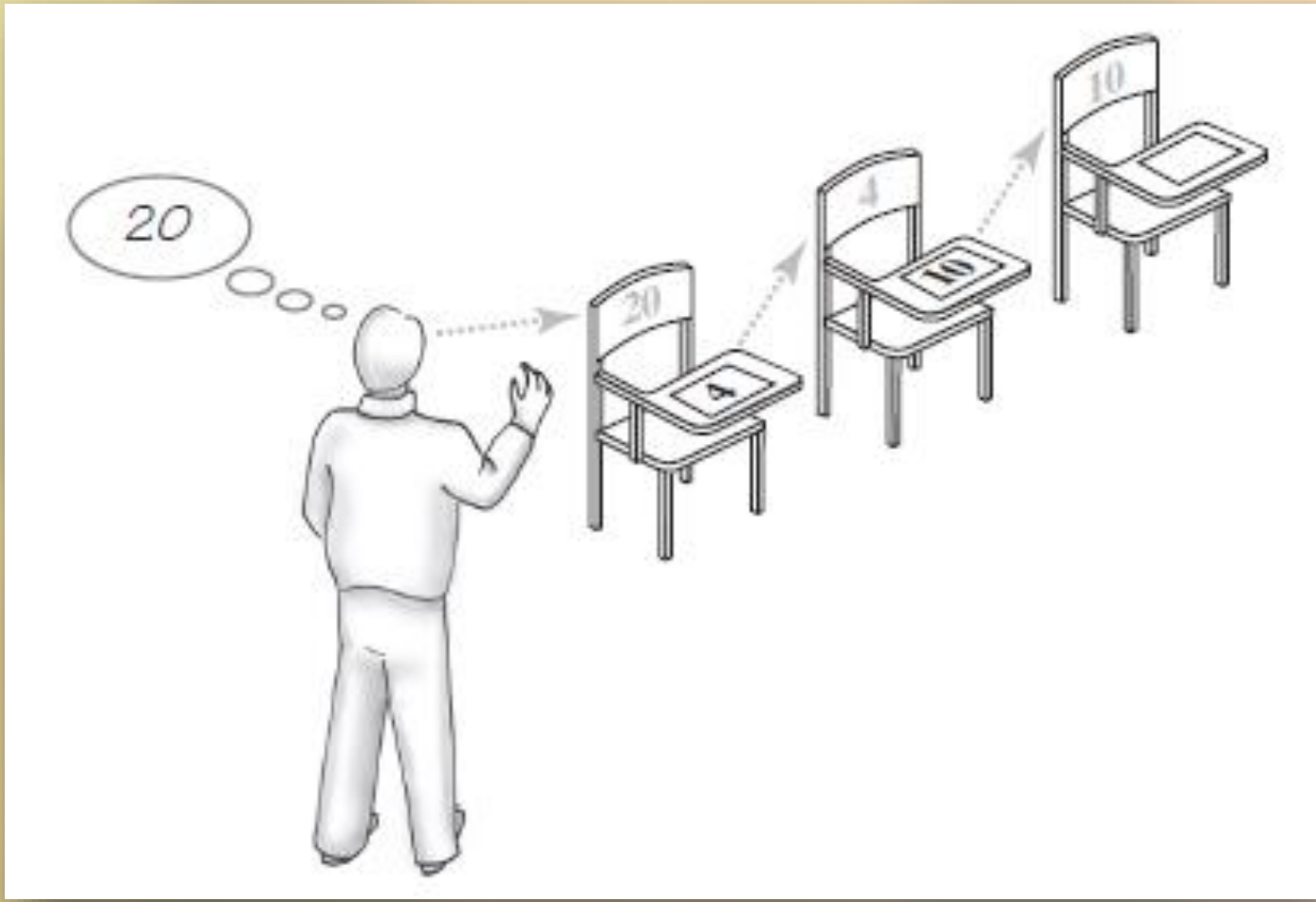

FIGURE 3-4 Three linked desks, with the newest desk first.

Forming a Chain by Adding to Its Beginning

```
// Process the first student
newDesk represents the new student's desk
New student sits at newDesk
Instructor memorizes the address of newDesk
// Process the remaining students
while (students arrive)
{
    newDesk represents the new student's desk
    New student sits at newDesk
    Write the instructor's memorized address on newDesk
    Instructor memorizes the address of newDesk
}
```


Pseudocode details the steps taken to form a chain of desks

The Private Class Node

```
1  private class Node
2  {
3     private T    data; // Entry in bag
4     private Node next; // Link to next node
5
6     private Node(T dataPortion)
7     {
8        this(dataPortion, null);
9     } // end constructor
10
11    private Node(T dataPortion, Node nextNode)
12    {
13       data = dataPortion;
14       next = nextNode;
15    } // end constructor
16 } // end Node
```


LISTING 3-1 The private inner class Node

The Private Class `Node`

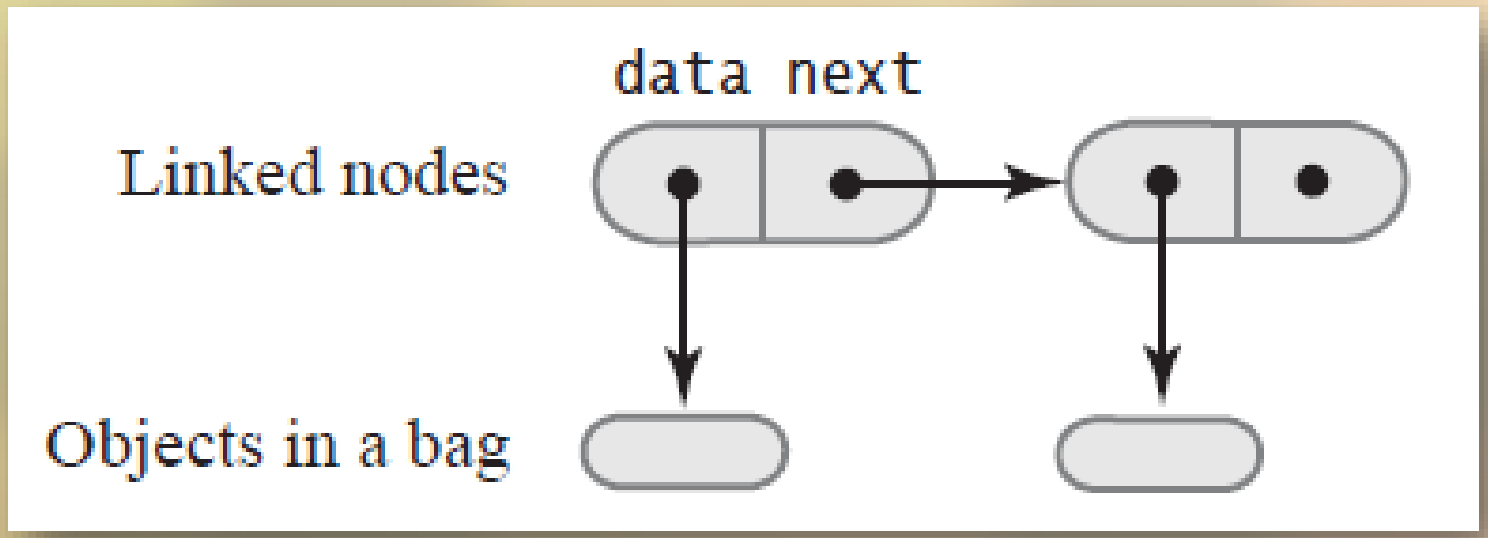

FIGURE 3-5 Two linked nodes that each reference object data

An Outline of the Class `LinkedBag`

```
/**
    A class of bags whose entries are stored in a chain of linked nodes.
    The bag is never full.
    @author Frank M. Carrano
*/
public final class LinkedBag<T> implements BagInterface<T>
{
   private Node firstNode;       // Reference to first node
   private int  numberOfEntries;

   public LinkedBag()
   {
      firstNode = null;
      numberOfEntries = 0;
   } // end default constructor

   // Implementations of the public methods declared in BagInterface go here...
```


LISTING 3-2 An outline of the class `LinkedBag`

An Outline of the Class `LinkedBag`

```
        numberOfEntries = 0;
    } // end default constructor

    < Implementations of the public methods declared in BagInterface go here. >

    . . .

    private class Node // Private inner class
    {
        < See Listing 3-1. >
    } // end Node
} // end LinkedBag
```

LISTING 3-2 An outline of the class `LinkedBag`

Beginning a Chain of Nodes

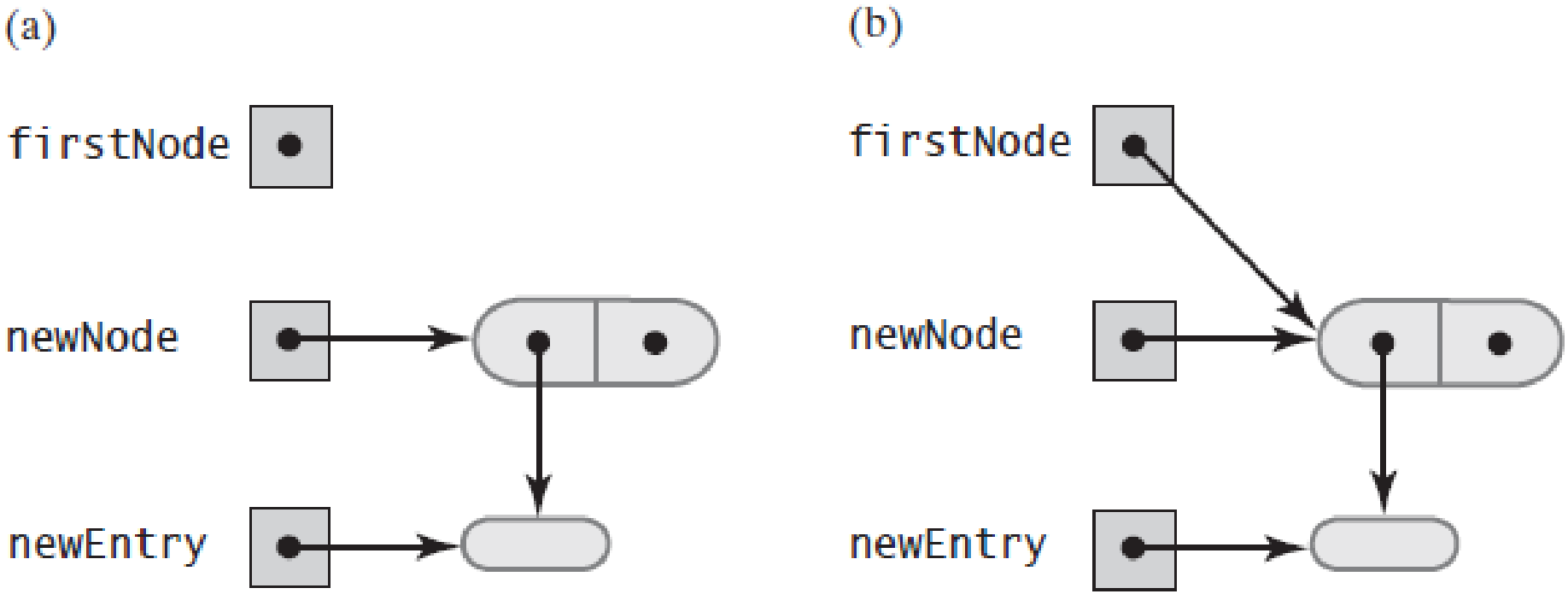

FIGURE 3-6 (a) An empty chain and a new node; (b) after adding a new node to a chain that was empty

Beginning a Chain of Nodes

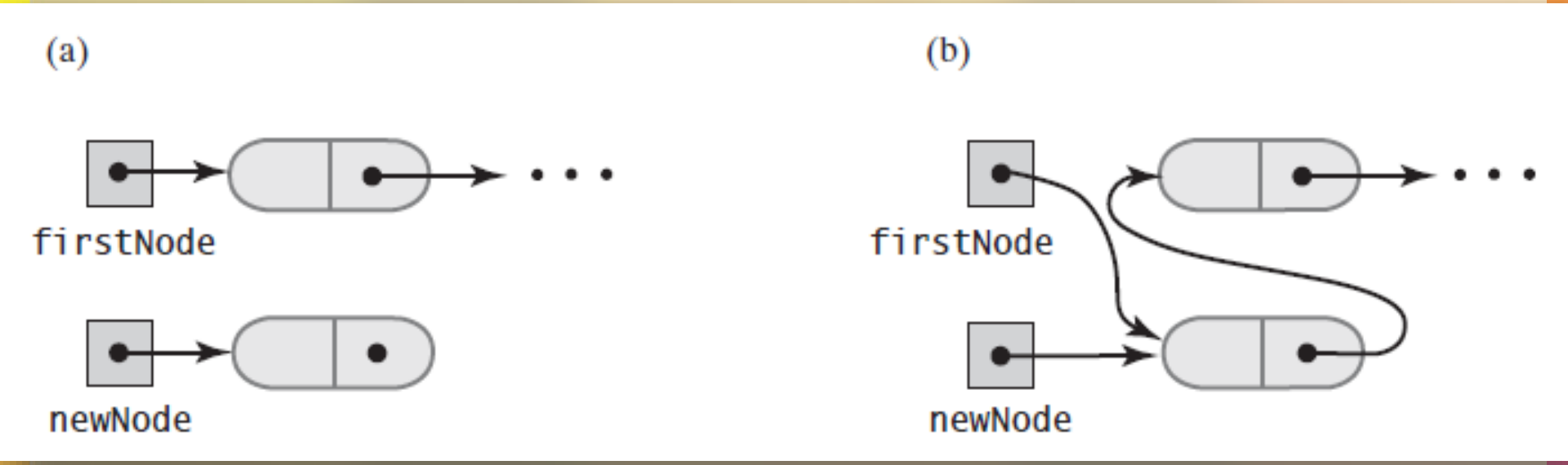

FIGURE 3-7 A chain of nodes (a) just prior to adding a node at the beginning; (b) just after adding a node at the beginning

Beginning a Chain of Nodes

```
/** Adds a new entry to this bag.
    @param newEntry  The object to be added as a new entry.
    @return  True. */
public boolean add(T newEntry) // OutOfMemoryError possible
{
   // Add to beginning of chain:
   Node newNode = new Node(newEntry);
   newNode.next = firstNode;    // Make new node reference rest of chain
                                // (firstNode is null if chain is empty)
   firstNode = newNode;    // New node is at beginning of chain
   numberOfEntries++;

   return true;
} // end add
```

The method **add**

Method `toArray`

```
/** Retrieves all entries that are in this bag.
    @return  A newly allocated array of all the entries in the bag. */
public T[] toArray()
{
   // The cast is safe because the new array contains null entries
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries]; // Unchecked cast

   int index = 0;
   Node currentNode = firstNode;
   while ((index < numberOfEntries) && (currentNode != null))
   {
      result[index] = currentNode.data;
      index++;
      currentNode = currentNode.next;
   } // end while

   return result;
} // end toArray
```

The method `toArray` returns an array of the entries currently in a bag

Test Program

```
/** A test of the methods add, toArray, isEmpty, and getCurrentSize,
    as defined in the first draft of the class LinkedBag.
    @author Frank M. Carrano
*/
public class LinkedBagDemo1
{
   public static void main(String[] args)
   {
      System.out.println("Creating an empty bag.");
      BagInterface<String> aBag = new LinkedBag<>();
      testIsEmpty(aBag, true);
      displayBag(aBag);

      String[] contentsOfBag = {"A", "D", "B", "A", "C", "A", "D"};
      testAdd(aBag, contentsOfBag);
      testIsEmpty(aBag, false);
   } // end main
```

LISTING 3-3 A sample program that tests some methods in the class **LinkedBag**

Test Program

```
} // end main

// Tests the method isEmpty.
// Precondition: If the bag is empty, the parameter empty should be true;
// otherwise, it should be false.
private static void testIsEmpty(BagInterface<String> bag, boolean empty)
{
System.out.print("\nTesting isEmpty with ");
if (empty)
   System.out.println("an empty bag:");
else
   System.out.println("a bag that is not empty:");

System.out.print("isEmpty finds the bag ");
if (empty && bag.isEmpty())
```

LISTING 3-3 A sample program that tests some methods in the class **LinkedBag**

Test Program

```
30      System.out.print("isEmpty finds the bag ");
31      if (empty && bag.isEmpty())
32         System.out.println("empty: OK.");
33      else if (empty)
34         System.out.println("not empty, but it is: ERROR.");
35      else if (!empty && bag.isEmpty())
36         System.out.println("empty, but it is not empty: ERROR.");
37      else
38         System.out.println("not empty: OK.");
39      } // end testIsEmpty
40      < The static methods testAdd and displayBag from Listing 2-2 are here. >
41  } // end LinkedBagDemo1
```

LISTING 3-3 A sample program that tests some methods in the class **`LinkedBag`**

Method `getFrequencyOf`

```
/** Counts the number of times a given entry appears in this bag.
    @param anEntry  The entry to be counted.
    @return  The number of times anEntry appears in the bag. */
public int getFrequencyOf(T anEntry)
{
   int frequency = 0;
   int loopCounter = 0;
   Node currentNode = firstNode;

   while ((loopCounter < numberOfEntries) && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         frequency++;

      loopCounter++;
      currentNode = currentNode.next;
   } // end while

   return frequency;
} // end getFrequencyOf
```


Counts the number of times a given entry appears

Method `contains`

```
public boolean contains(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         found = true;
      else
         currentNode = currentNode.next;
   } // end while

   return found;
} // end contains
```

Determine whether a bag contains a given entry

Removing an Item from a Linked Chain

- Case 1: Desk to be removed is first in the chain of desks.
- Case 2: Desk to be removed is not first in the chain of desks.

Removing an Item from a Linked Chain

Case 1

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

Removing an Item from a Linked Chain

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

Removing an Item from a Linked Chain

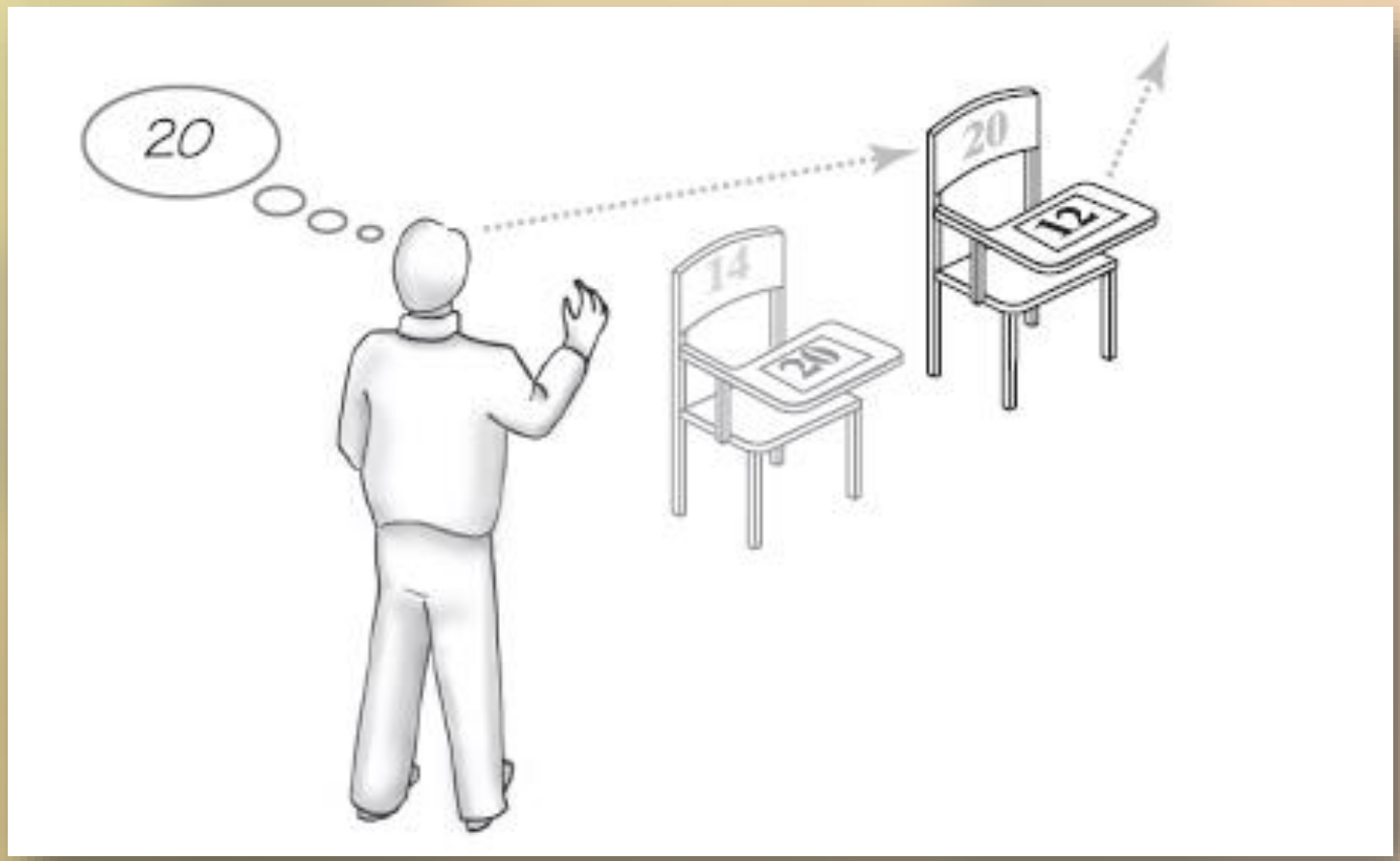

FIGURE 3-9 A chain of desks just after removing its first desk

Removing an Item from a Linked Chain

Case 2

1. Move the student in the first desk to the desk to be removed.
2. Remove the first desk using the steps described for Case 1.

Removing an Item from a Linked Chain

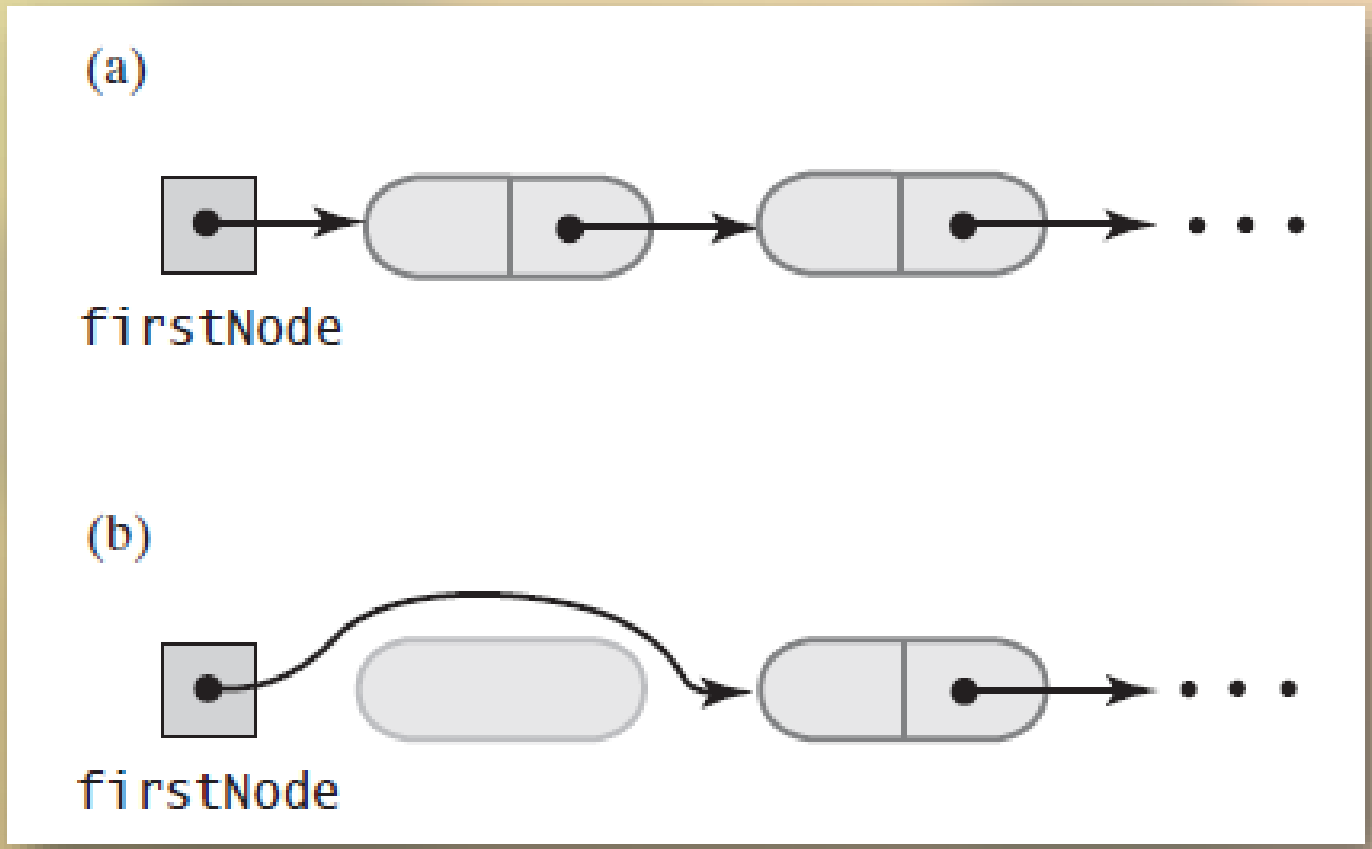

FIGURE 3-10 A chain of nodes (a) just prior to removing the first node; (b) just after removing the first node

Method `remove`

```
// Locates a given entry within this bag.
// Returns a reference to the node containing the entry, if located,
// or null otherwise.
private Node getReferenceTo(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         found = true;
      else
         currentNode = currentNode.next;
   } // end while

   return currentNode;
} // end getReferenceTo
```


Note need for private method `getReferenceTo`

Method `remove`

```
public boolean remove(T anEntry)
{
   boolean result = false;
   Node nodeN = getReferenceTo(anEntry);

   if (nodeN != null)
   {
      nodeN.data = firstNode.data; // Replace located entry with entry
                                   // in first node
      firstNode = firstNode.next;  // Remove first node
      numberOfEntries--;
      result = true;
   } // end if

   return result;
} // end remove
```


Note use of method `getReferenceTo`

Method `clear`

```
public void clear()
{
    while (!isEmpty())
        remove();
} // end clear
```

As in previous implementation, uses **`isEmpty`** and **`remove`**

Class Node That Has Set and Get Methods

```
1  private class Node
2  {
3     private T    data; // Entry in bag
4     private Node next; // Link to next node
5
6     private Node(T dataPortion)
7     {
8        this(dataPortion, null);
9     } // end constructor
10
11    private Node(T dataPortion, Node nextNode)
12    {
13       data = dataPortion;
14       next = nextNode;
15    } // end constructors
16
17    private T getData()
```


LISTING 3-4 The inner class Node with set and get methods

Class Node That Has Set and Get Methods

```
16
17      private T getData()
18      {
19          return data;
21      } // end getData
22
23      private void setData(T newData)
24      {
25          data = newData;
26      } // end setData
27
28      private Node getNextNode()
29      {
30          return next;
31      } // end getNextNode
32
```


LISTING 3-4 The inner class Node with set and get methods

Class `Node` That Has `Set` and `Get` Methods

```
27
28      private Node getNextNode()
29      {
30          return next;
31      } // end getNextNode
32
33      private void setNextNode(Node nextNode)
34      {
35          next = nextNode;
36      } // end setNextNode
37  } // end Node
```


LISTING 3-4 The inner class `Node` with `set` and `get` methods

A Class within A Package

```
package BagPackage;
class Node<T>
{
   private T       data;
   private Node<T> next;

   Node(T dataPortion) // The constructor's name is Node, not Node<T>
   {
      this(dataPortion, null);
   } // end constructor
```

LISTING 3-5 The class **Node** with package access

A Class within A Package

```
12      Node(T dataPortion, Node<T> nextNode)
13      {
14          data = dataPortion;
15          next = nextNode;
16      } // end constructor
17
18      T getData()
19      {
20          return data;
21      } // end getData
22
23      void setData(T newData)
24      {
25          data = newData;
26      } // end setData
27
```


LISTING 3-5 The class Node with package access

A Class within A Package

```
23      void setData(T newData)
24      {
25          data = newData;
26      } // end setData
27
28      Node<T> getNextNode()
29      {
30          return next;
31      } // end getNextNode
32
33      void setNextNode(Node<T> nextNode)
34      {
35          next = nextNode;
36      } // end setNextNode
37  } // end Node
```

LISTING 3-5 The class **Node** with package access

When Node Is in Same Package

```
package BagPackage;
public class LinkedBag<T> implements BagInterface<T>
{
   private Node<T> firstNode;
   . . .

   public boolean add(T newEntry)
   {
      Node<T> newNode = new Node<T>(newEntry);
      newNode.setNextNode(firstNode);
      firstNode = newNode;
      numberOfEntries++;

      return true;
   } // end add
   . . .
} // end LinkedBag
```

This occurrence of T is optional

LISTING 3-6 The class **LinkedBag** when **Node** is in the same package

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

End

Chapter 3