A Bag Implementation that Links Data

Chapter 3

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

Contents

- Linked Data
  - Forming a Chain by Adding to Its Beginning
- A Linked Implementation of the ADT Bag
  - The Private Class **`Node`**
  - An Outline of the Class **`LinkedBag`**
  - Defining Some Core Methods
  - Testing the Core Methods
  - The Method **`getFrequencyOf`**
  - The Method **`contains`**

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

Contents

- Removing an Item from a Linked Chain
  - The Methods **`remove`** and **`clear`**
- A Class **`Node`** That Has Set and Get Methods
- The Pros and Cons of Using a Chain to Implement the ADT Bag

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

Objectives

- Describe linked organization of data
- Describe how to add new node to beginning of chain of linked nodes
- Describe how to remove first node in chain of linked nodes

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

Objectives

- Describe how to locate particular piece of data within chain of linked nodes
- Implement ADT bag by using chain of linked nodes
- Describe differences between array-based and linked implementations of ADT bag

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

Linked Data

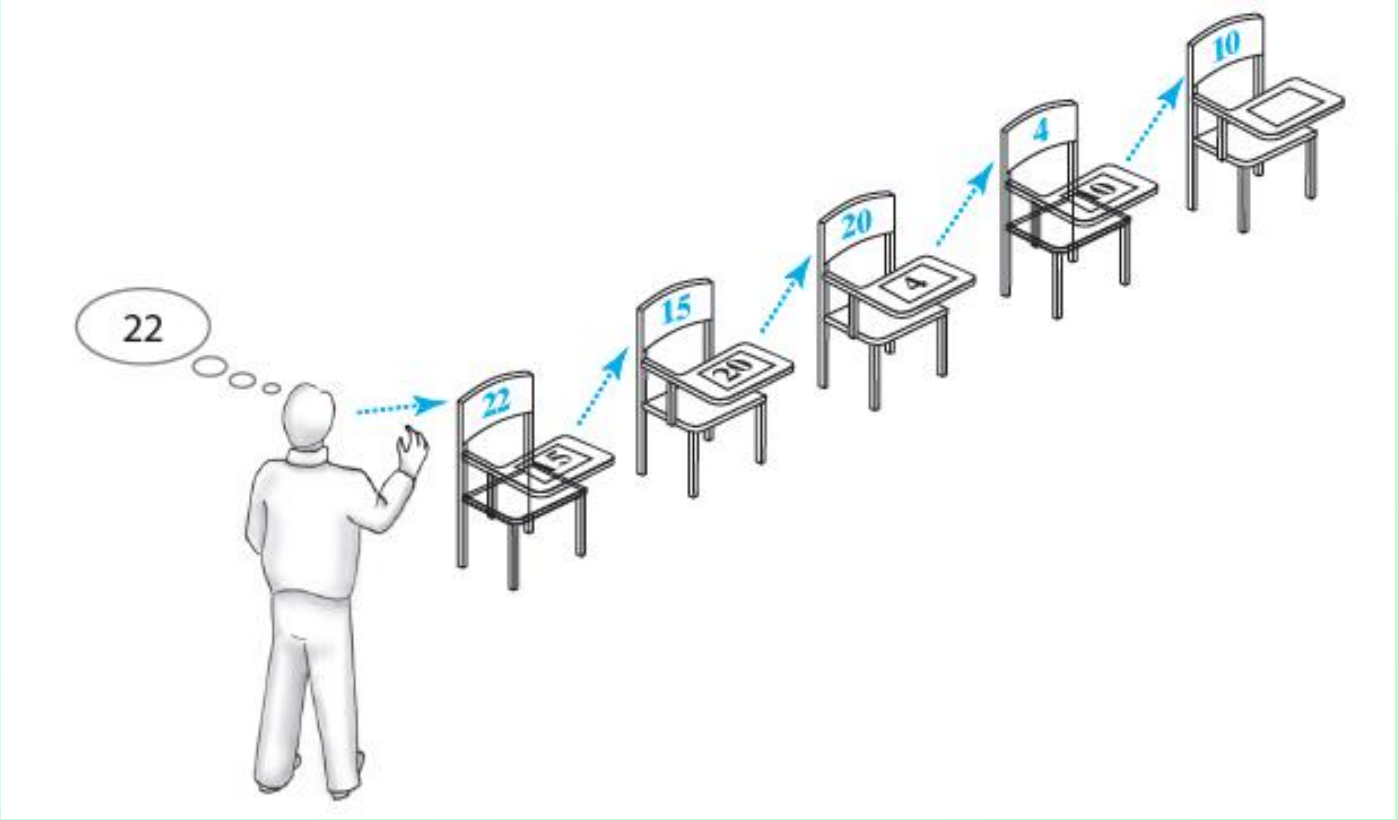


Figure 3-1 A chain of five desks
As they enter classroom, students bring in a desk from hall

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

Forming a Chain by Adding to Its Beginning

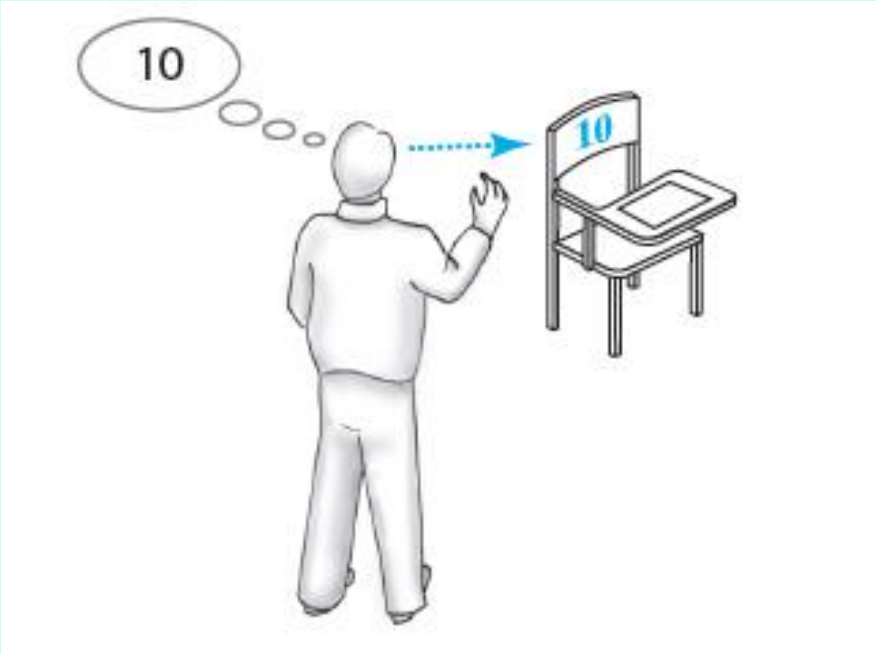


Figure 3-2 One desk in the room

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

Forming a Chain by Adding to Its Beginning

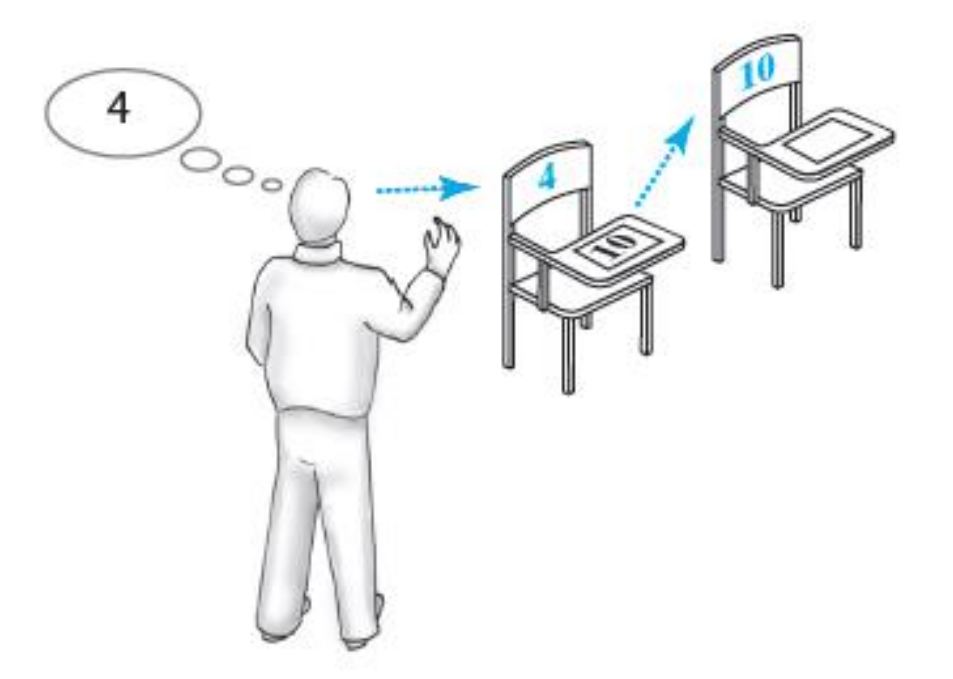


Figure 3-3 Two linked desks, with the newest desk first

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

Forming a Chain by Adding to Its Beginning

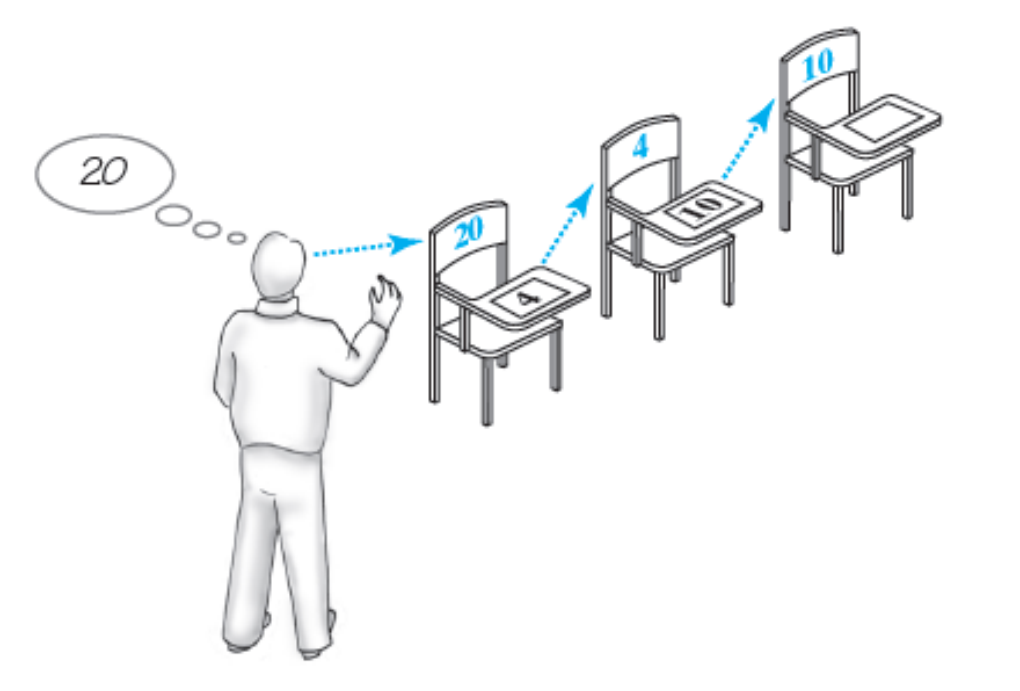


Figure 3-4 Three linked desks, with the newest desk first

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

Question 1 The instructor knows the address of only one desk.

a. Where in the chain is that desk: first, last, or somewhere else?

b. Who is sitting at that desk: the student who arrived first, the student who arrived last, or someone else?

Question 2 Where in the chain of desks is a new desk added: at the beginning, at the end, or somewhere else?

1. **a.** First.
   b. The student who arrived last (most recently).

2. At the beginning.

A Linked Implementation of the ADT Bag

- The private class **`Node`**
  - (this is the Desk in our analogy)

```
class Node
{
    private T    data; // entry in bag
    private Node next; // link to next node

    < Constructors >
    . . .
    < Accessor and mutator methods: getData, setData, getNextNode, setNextNode >
    . . .
} // end Node
```

  - the Student sitting at the desk would be "data"
  - the sign referring to the next desk would be "next"

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

The Private Class **Node**

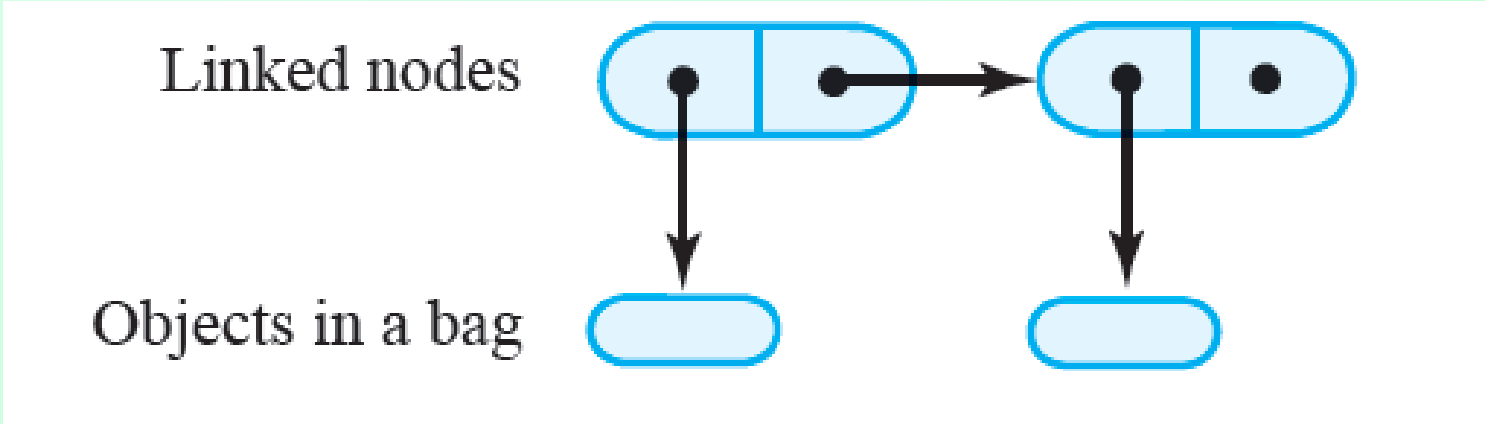


Figure 3-5 Two linked nodes that each reference object data

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

Class **`LinkedBag`**

- View source code of **`Node`** class, Listing 3-1
- Note outline of class **`LinkedBag`** (Listing 3-2) which uses class **`Node`**

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

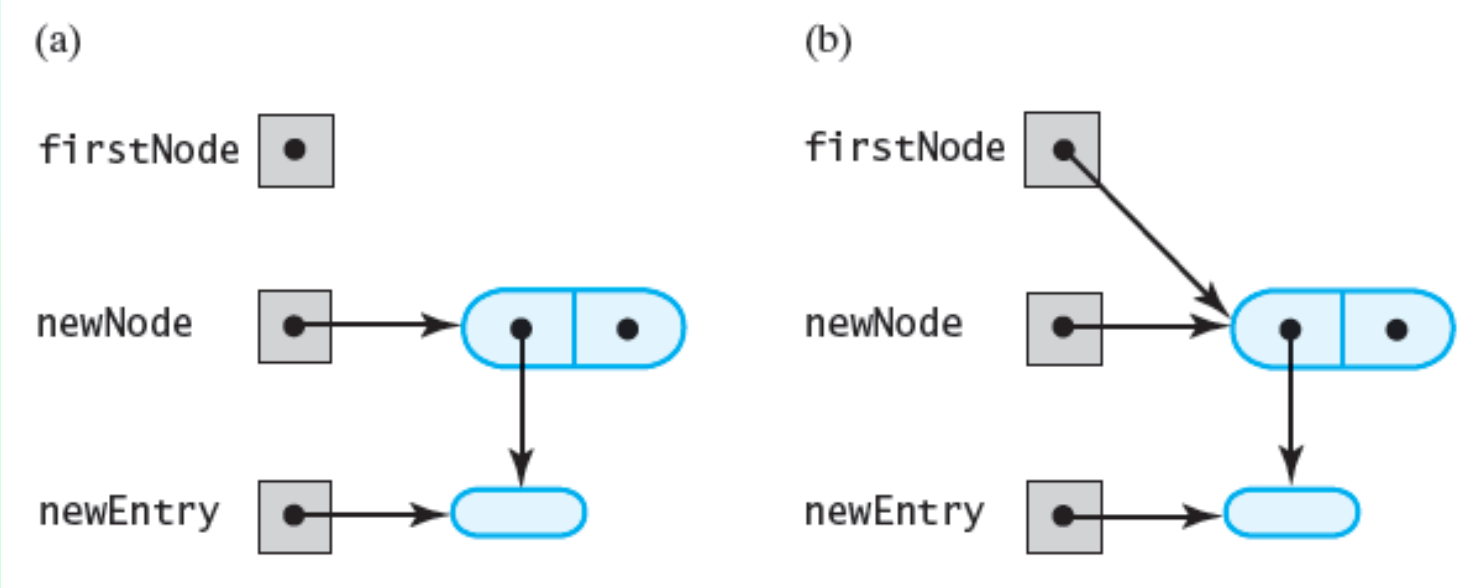


Figure 3-6 (a) An empty chain and a new node; (b) after adding a new node to a chain that was empty

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

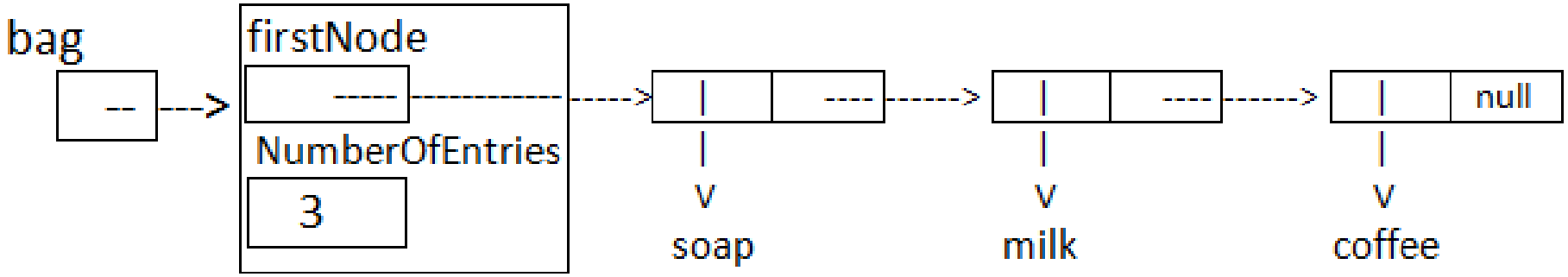
Figure 3-7 A chain of nodes (a) just prior to adding a node at the beginning; (b) just after adding a node at the beginning

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

2a) Draw the Bag as each item is added

```
BagInterface<String> bag = new LinkedBag<String>();
bag.add("coffee");
bag.add("milk");
bag.add("soap");
```

2a Answer



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

```java
/** 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**

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

Question 3 The code that we developed in Segment 3.10 to add a node to an empty chain is

```
Node newNode = new Node(newEntry);
firstNode = newNode;
```

The code that we just developed to add to the beginning of a chain is

```
Node newNode = new Node(newEntry);
newNode.next = firstNode;
firstNode = newNode;
```

Why do these three statements also work correctly when the chain is empty?

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

3. When the chain is empty, `firstNode` is `null`. Setting `newNode.next` to `firstNode` sets it to `null`. Since `newNode.next` already is `null`, no harm is done by the additional assignment.

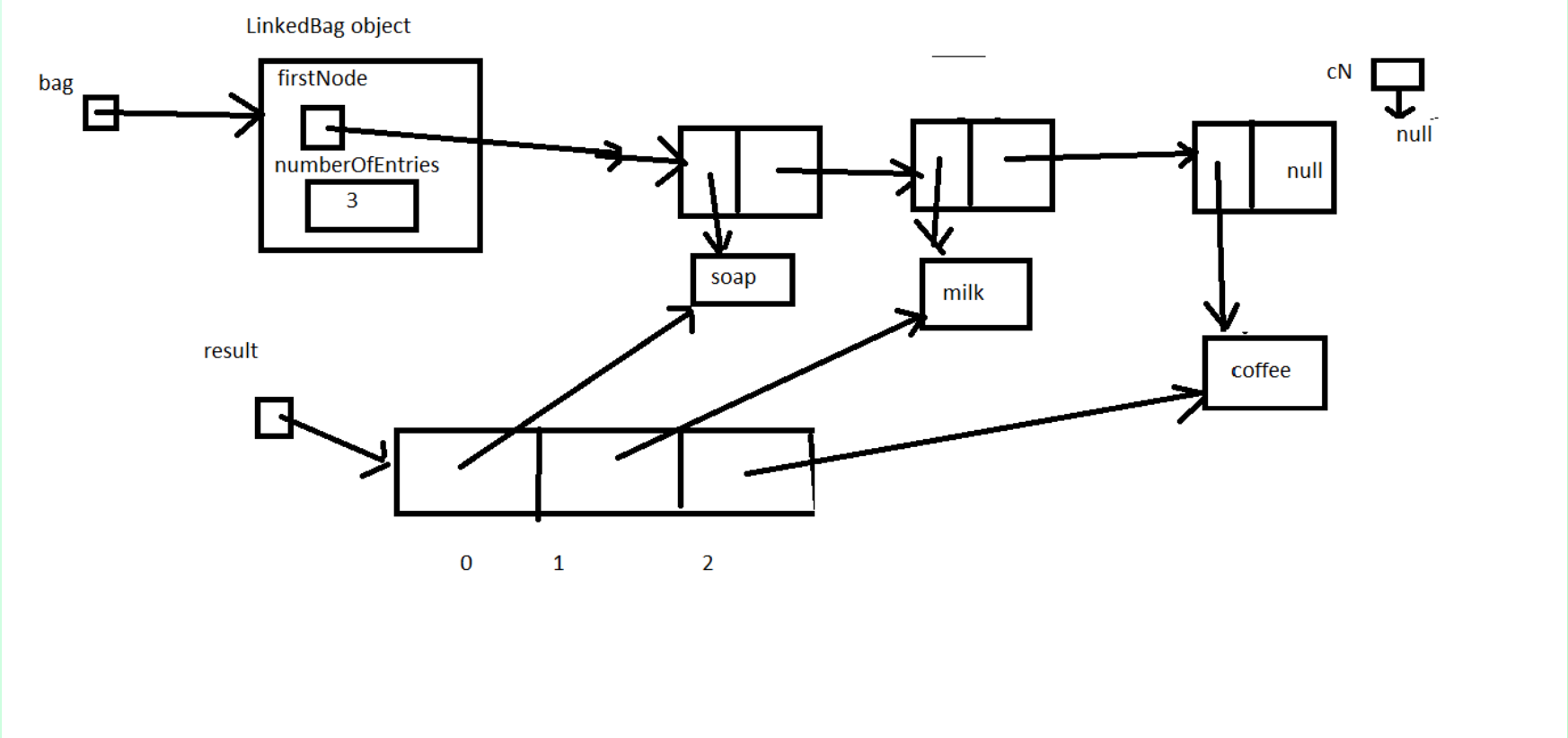
Copyright ©2012 by Pearson Education, Inc. All rights reserved

```java
/** 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`**

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

Memory map after the toArray method runs



Question 4 In the previous definition of `toArray`, the `while` statement uses the boolean expression `(index < numberOfEntries) && (currentNode != null)` to control the loop. Is it necessary to test the values of both `index` and `currentNode`? Explain your answer.

4. Testing the values of both `index` and `currentNode` is not necessary. Although testing either one of these values is sufficient, testing both values provides a check against mistakes in your code.

Testing Core Methods

- Initial test methods
  - **`add`**
  - **`toArray`**
- Stub methods (for now)
  - **`getCurrentSize`**
  - **`isFull`**
  - **`isEmpty`**
- View source code, Listing 3-3

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

```java
/** 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 counter = 0;
   Node currentNode = firstNode;
   while ((counter < numberOfEntries) && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         frequency++;

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

   return frequency;
} // end getFrequencyOf
```

Method **`getFrequencyOf`**

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

```
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
```

Method **`contains`**

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

Question 5 If `currentNode` in the previous method `contains` becomes `null`, what value does the method return when the bag is not empty?

Question 6 Trace the execution of the method `contains` when the bag is empty. What is the result?

5. The method returns false. If `currentNode` becomes `null`, the entire chain has been searched without success.

6. Since the bag is empty, `firstNode`—and hence `currentNode`—is `null`. The `while` loop ends immediately and the method returns false.

Removing a Student from the Classroom (Removing an Item from a Linked Chain)

- Case 1
  - The departing student is sitting in first desk.

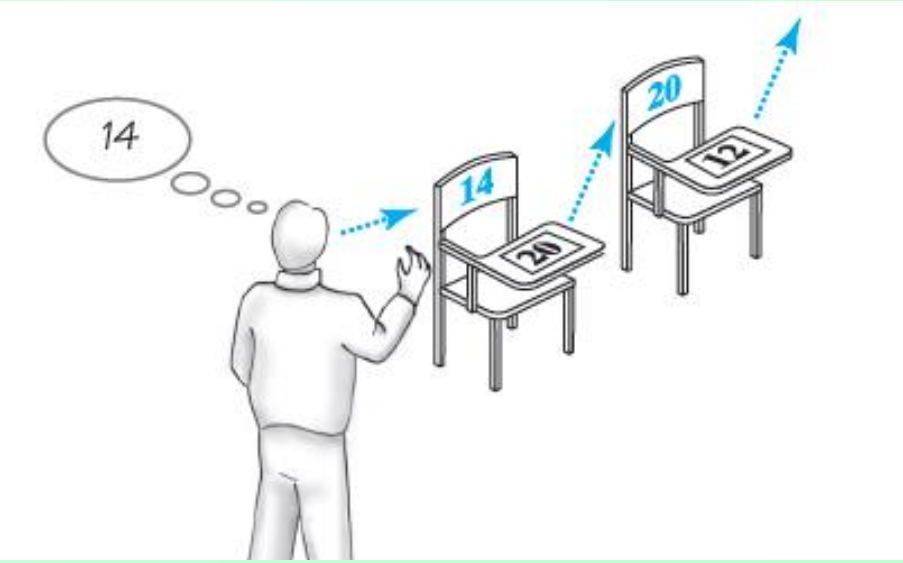


Figure 3-8 A chain of desks just prior to removing the first student/desk

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

Steps Required for Case 1

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

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

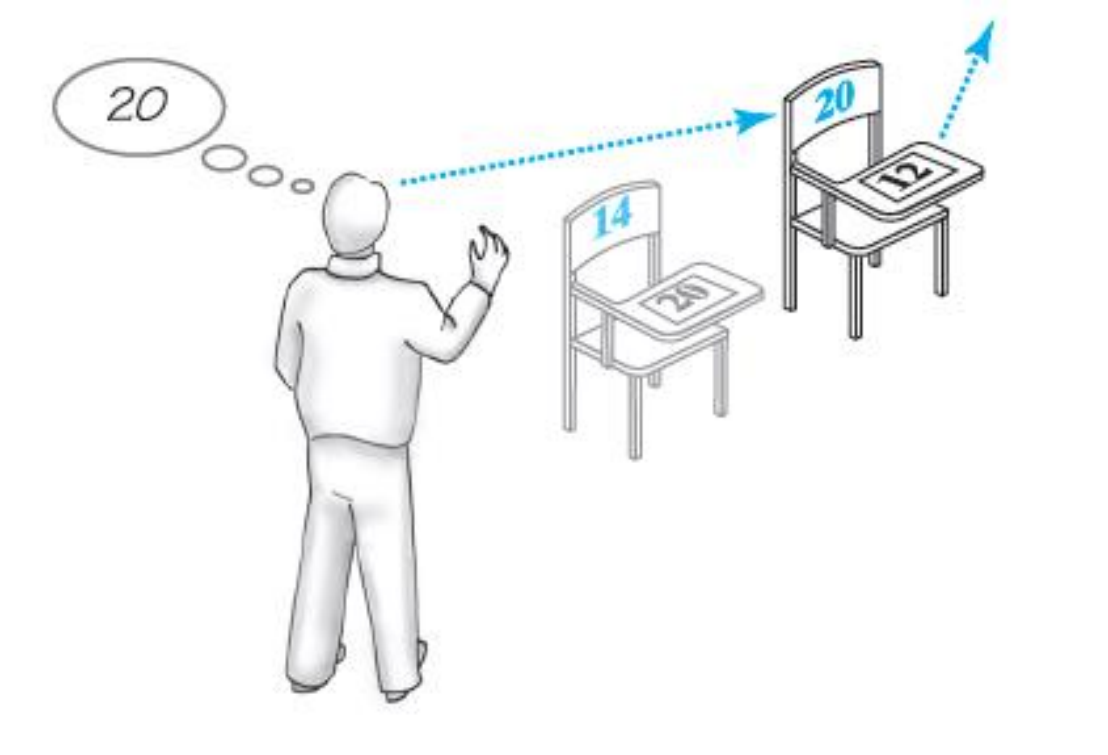


Figure 3-9 A chain of desks just after removing the first student/desk (after first two steps)

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

Removing a Student from the Classroom (Removing an Item from a Linked Chain)

- Case 2
  - Departing Student is *not* sitting in the first desk.
- Steps required for Case 2
  1. Move student in the first desk to the desk about to be vacated by departing student.
  2. Remove first desk using steps described for Case 1.

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

Question 7 What steps are necessary to remove the first desk in a chain of five desks?

Question 8 What steps are necessary to remove the third desk in a chain of five desks?

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

7.
- Locate the first desk by asking the instructor for its address.
- Give the address that is written on the first desk's paper to the instructor. This is the address of the second desk in the chain.
- Return the first desk to the hallway.

8.
- The student in the first desk moves to the third desk.
- Remove the first desk using the three steps given as the answer to the previous question.

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

OK now instead of Desks think Nodes

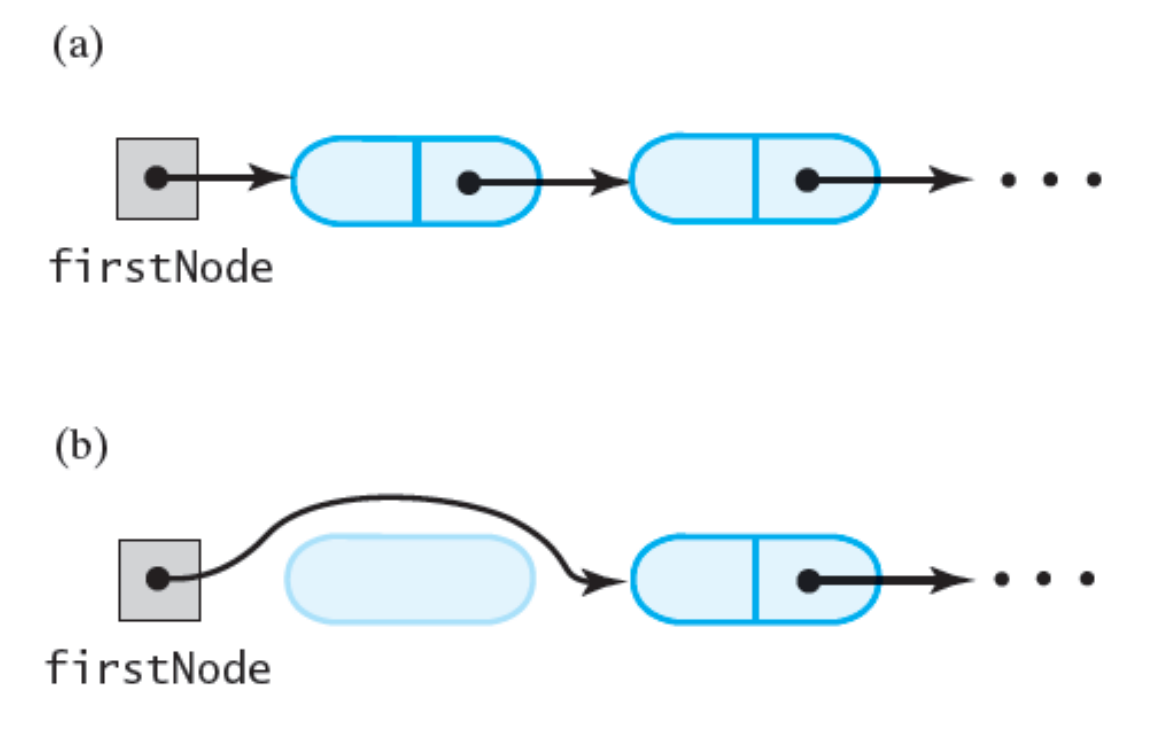


Figure 3-10 A chain of nodes (a) just prior to removing the first node; (b) just after removing the first node

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

This is case 1: remove first node

```
public T remove()
{
   T result = null;
   if (firstNode != null)
   {
      result = firstNode.data;
      firstNode = firstNode.next; // remove first node from chain
      numberOfEntries--;
   } // end if

   return result;
} // end remove
```

Method **`remove`**

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

Removing a Given Entry

- Required steps

  1. Replace entry in node *N* with entry in first node.
  2. Remove first node from chain.

- Must find reference to specified value

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

```
// 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
```

Method **`getReferenceTo`**

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

Method `remove` for Given Value

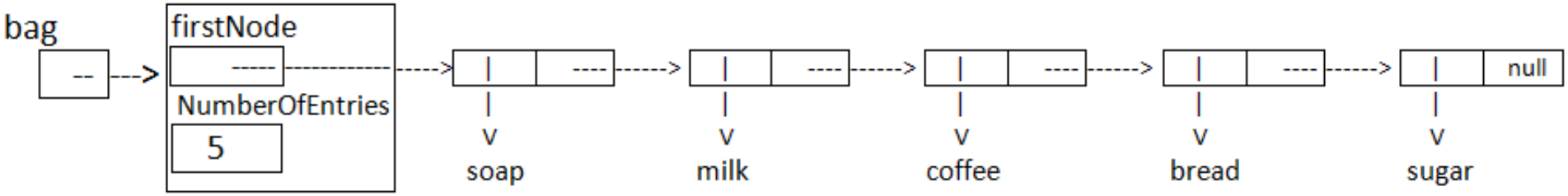
```
public boolean remove(T anEntry)
{
   boolean result = false;
   Node nodeN = getReferenceTo(anEntry);

   if (nodeN != null)
   {
      nodeN.data = firstNode.data; // replace located entry with entry
                                   // in first node
      remove();                    // remove first node
      result = true;
   } // end if

   return result;
} // end remove
```

- Method `clear`

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

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

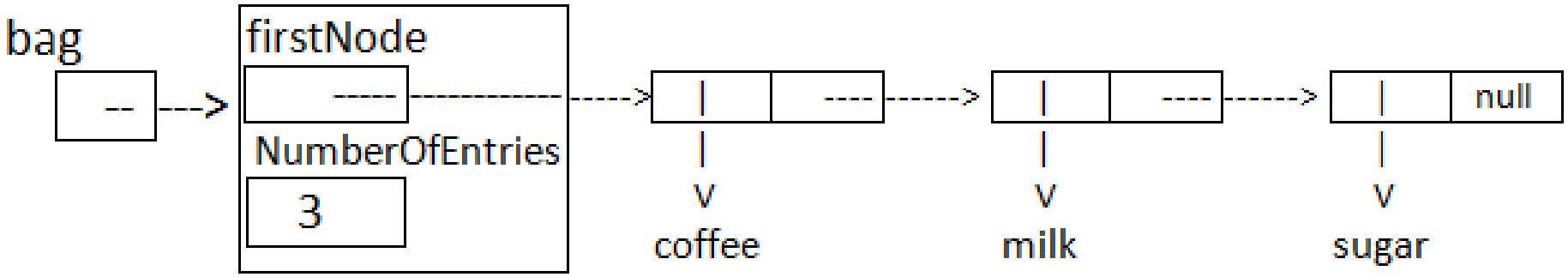
9) Draw the bag after removals




```
bag.remove();
bag.remove("bread");
```

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

9 Answer



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

Question 10 Revise the definition of the method `contains` so that it calls the private method `getReferenceTo`.

Question 11 Revise the definition of the method `getReferenceTo` so that it controls its loop by using a counter and `numberOfEntries` instead of `currentNode`.

Question 12 What is an advantage of the definition of `getReferenceTo`, as given in the previous segment, over the one that the previous question describes?

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

10.
```
public boolean contains(T anEntry)
{
   return getReferenceTo(anEntry) != null;
} // end contains
```

11.
```
private Node getReferenceTo(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;
   int counter = 0;

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

   return currentNode;
} // end getReferenceTo
```

12. **The original definition of** `getReferenceTo` **ensures that** `currentNode` **is not** `null`, **and thus avoids a** `NullPointerException`.

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

Putting It Together
3 design approaches

- private inner class **`Node`** that has **`Set`** and **`Get`** methods, uses LinkedBags generic T
  - View source code, Listing 3-4
- Class **`Node`** with package access and own generic specifier T
  - Source code, Listing 3-5
- Class **`LinkedBag`** with **`Node`** in same package, needs different generic specifier S
  - Note source code, Listing 3-6

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

Pros and Cons of Using a Chain

- Bag can grow and shrink in size as necessary
- Possible to remove and recycle nodes
- Copying values for array enlargement not needed
- Removing specific value entry requires search of array or chain
- Chain requires more memory than array of same length

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

Question 13 Compare the efforts made by the `contains` methods in the classes `LinkedBag` in this chapter and `ResizableArrayBag` in Chapter 2. Does one take more time to perform its task? Explain.

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

13. The effort expended by each of these two methods is about the same. Each method calls a private method that searches for the desired entry. In `LinkedBag`, `contains` calls `getReferenceTo`, which searches at most `numberOfEntries` nodes for the desired entry. In `ResizableArrayBag`, `contains` calls `getIndexOf`, which searches at most `numberOfEntries` array elements for the desired entry. The next chapter will discuss these methods and analyze their time requirements in more detail.

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

End

Chapter 3

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