Northeastern University

Lecture 13: A Bag Implementation that Links Data - 2

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5 © 2019 Pearson Education, Inc.

Removing an Item from a Linked Chain

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 the address of the second desk in chain.)
  - » Return first desk to hallway

Removing an Item from a Linked Chain

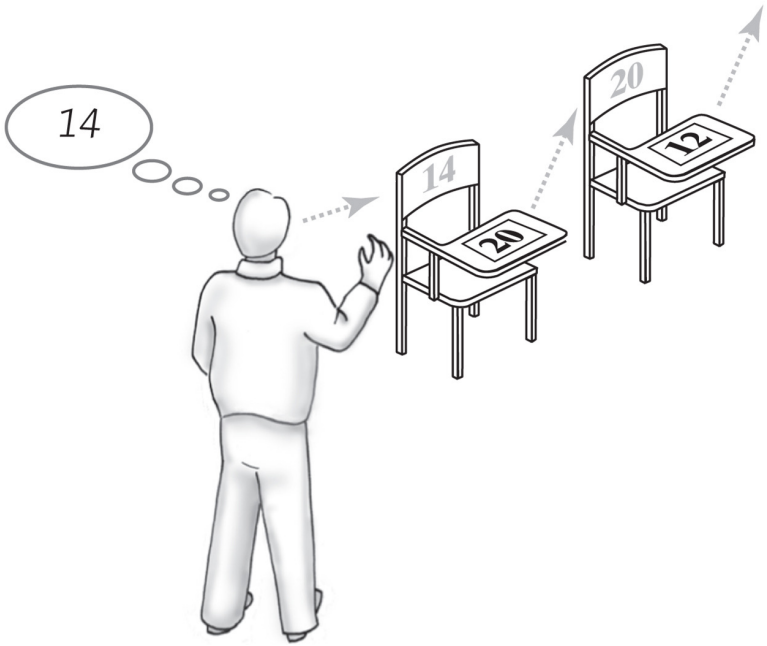

Figure 3-8: A chain of desks just prior to removing its first desk

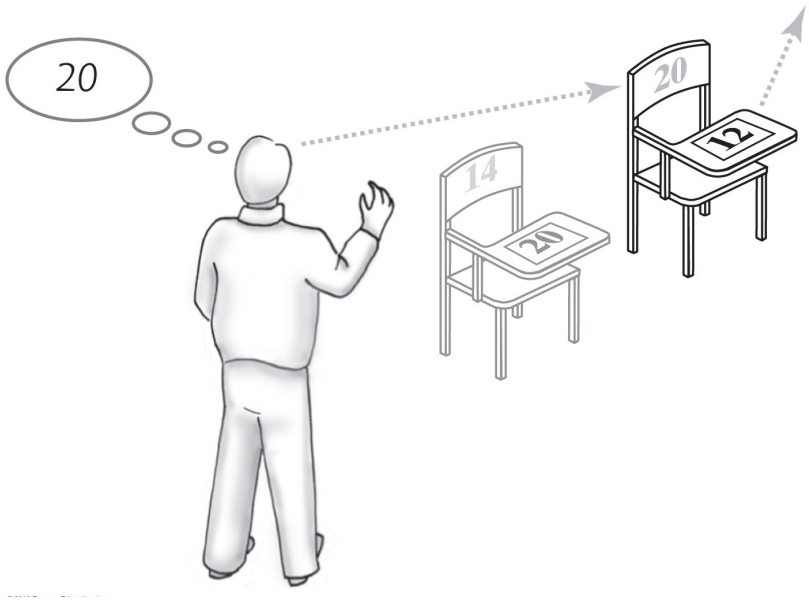

Figure 3-9: A chain of desks just after removing its first desk

Removing an Item from a Linked Chain

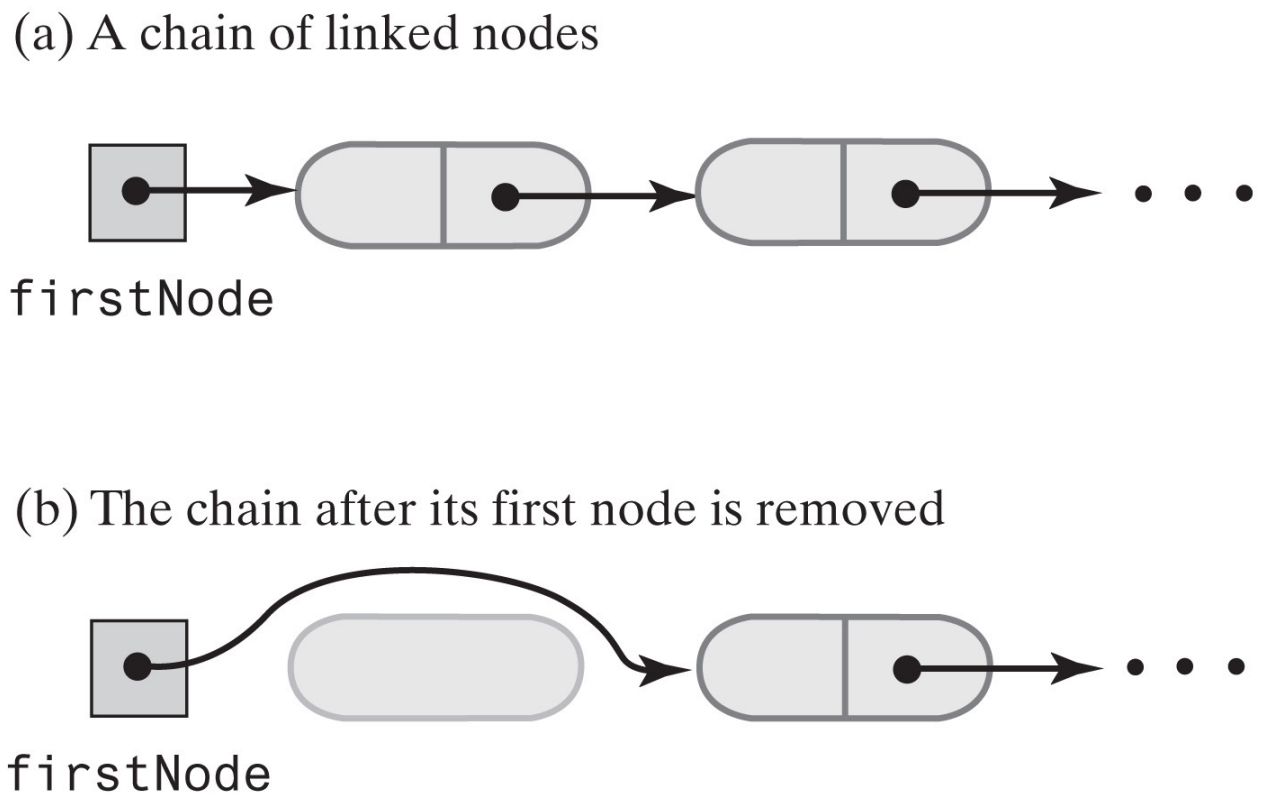

Figure 3-10: A chain of nodes (a) just prior to removing the first node; (b) just after removing the first node

Removing an Item from a Linked Chain

- Case 2:
  - » Move the student (data) in the first desk to your former desk (the desk you want to remove in the chain)
  - » Remove the first desk using the steps described for Case 1

Method `remove`

```
/** Removes one unspecified entry from this bag, if possible.
    @return  Either the removed entry, if the removal was successful,
             or null */
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 `clear`

```
/** Removes all entries from this bag.  */
  public void clear()
  {
      while (!isEmpty())
         remove();
  } // end clear
```

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

A Class `Node` That Has Set and Get Methods

As an inner Class

```
private class Node
{
   private T data;    // Entry in bag
   private Node next; // Link to next node

   private Node(T dataPortion)
   {
      this(dataPortion, null);
   } // end constructor

   private Node(T dataPortion, Node nextNode)
   {
      data = dataPortion;
      next = nextNode;
   } // end constructor

   private T getData()
   {
      return data;
   } // end getData

   private void setData(T newData)
   {
      data = newData;
   } // end setData

   private Node getNextNode()
   {
      return next;
   } // end getNextNode

   private void setNextNode(Node nextNode)
   {
      next = nextNode;
   } // end setNextNode
} // end Node
```

Listing 3-4: The inner class **`Node`** with **`set`** and **`get`** methods

As a Class within a Package

```
package BagPackage;

class Node<T>
{
   private T       data;
   private Node<T> next;

   Node(T dataPortion)
   {
      this(dataPortion, null);
   } // end constructor

   Node(T dataPortion, Node<T> nextNode)
   {
      data = dataPortion;
      next = nextNode;
   } // end constructor

   T getData()
   {
      return data;
   } // end getData

   void setData(T newData)
   {
      data = newData;
   } // end setData

   Node<T> getNextNode()
   {
      return next;
   } // end getNextNode

   void setNextNode(Node<T> nextNode)
   {
      next = nextNode;
   } // end setNextNode
} // 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
```

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.
- Removing and recycling nodes that are no longer needed.
- No need to resize the storage.
- Adding a new entry to the end of array or to the beginning of chain both relatively simple.

Cons of Using a Chain

- Removing specific entry requires search of array or chain.
- The chain requires more memory than an array of the same length.

Question

- Compare the efforts made by the `contains` method in the classes `LinkedBag` and `ResizableArrayBag`.
- Does one take more time to perform its task?

Answer

- 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.

Exercise

- Define a method `removeEvery` for the class `LinkedBag` that removes all occurrences of a given entry from a bag.

```
public void removeEvery(T anEntry)
```

Answer

```
/** Removes every occurrence of a given entry from this bag.
    @param anEntry The entry to be removed. */

public void removeEvery(T anEntry) {
    if (!isEmpty()){
        Node searcher = firstNode;
        while (searcher != null)
        {
            T nextEntry = searcher.data;
            if (nextEntry.equals(anEntry)) {
                // Replace located entry with entry in first node
                searcher.data = firstNode.data;

                firstNode = firstNode.next; // Remove first node
                numberOfEntries--;
            } // end if
            searcher = searcher.next; // Continue searching
        } // end while
    } // end if
} // end removeEvery
```