Northeastern University

Lecture 12: A Bag Implementation that Links Data - 1

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.

Problems with Array Implementation

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

Alternative Approach?!

- Introduces a new data organization that uses memory only as needed for a new entry.
- Returns the unneeded memory to the system after an entry is removed.

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

Analogy

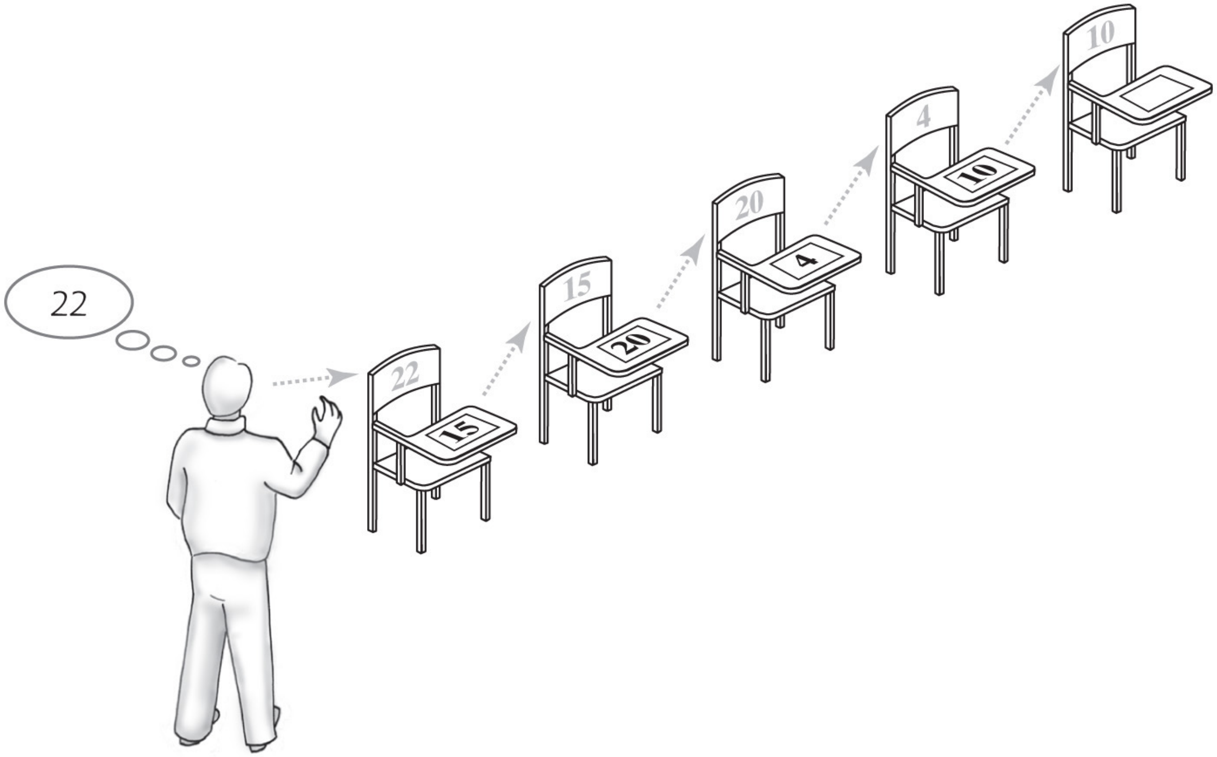


Figure 3-1: A chain of five desks

Forming a Chain by Adding to Its Beginning

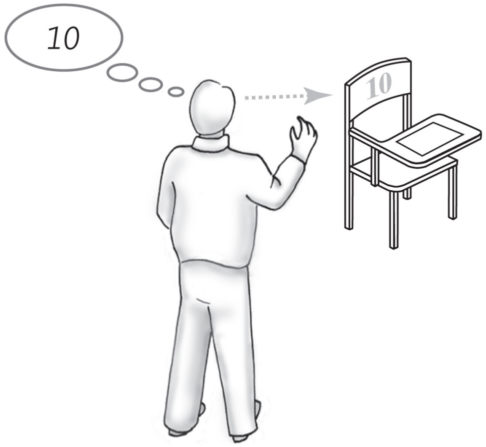


Figure 3-2: One desk in the room

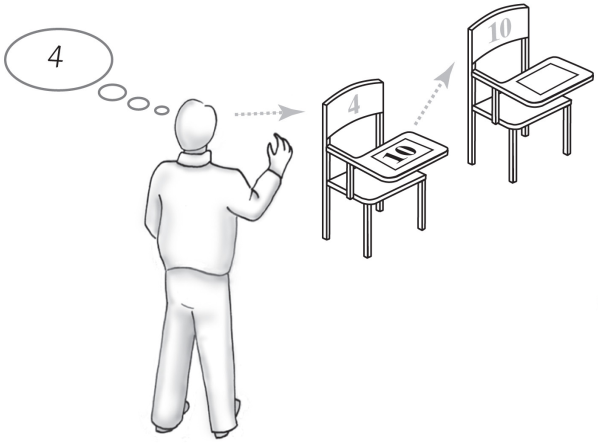


Figure 3-3: Two linked desks, with the newest desk first

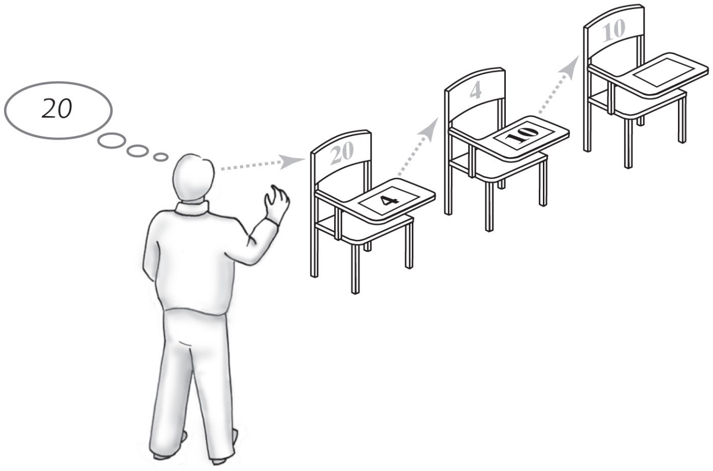


Figure 3-4: Three linked desks, with the newest desk first.

Question

- The instructor knows the address of only one desk.
  - A. Where in the chain is that desk?
    - First
    - Last
    - Somewhere else
  - B. Who is sitting at that desk?
    - the student who arrived first
    - the student who arrived last
    - Someone else

Answer

- The instructor knows the address of only one desk.
  - A. Where in the chain is that desk?
    - First
    - Last
    - Somewhere else
  - B. Who is sitting at that desk?
    - the student who arrived first
    - the student who arrived last
    - Someone else

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**

```
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

 } // end Node
```

Listing 3-1: The private inner class **Node**

The Private Class `Node`

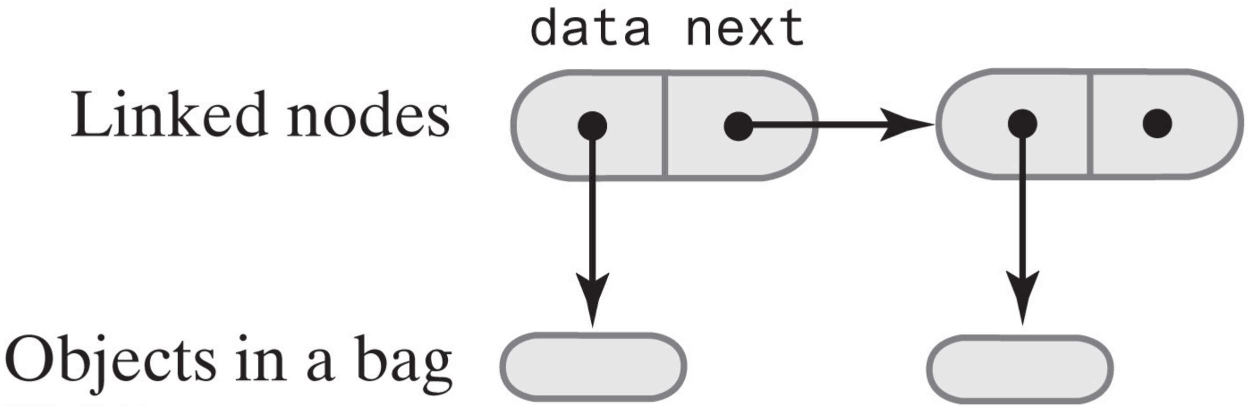


Figure 3-5: Two linked nodes that each reference object data

An Outline of the Class LinkedBag - Part 1

```
/** A class of bags whose entries are stored in a chain
    of linked nodes. The bag is never full. */

public 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

   // . . .
```

Listing 3-2: An outline of the class LinkedBag

An Outline of the Class **LinkedBag** - Part 2

```
   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
   } // end Node

} // end LinkedBag
```

Listing 3-2: An outline of the class **LinkedBag**

Beginning a Chain of Nodes

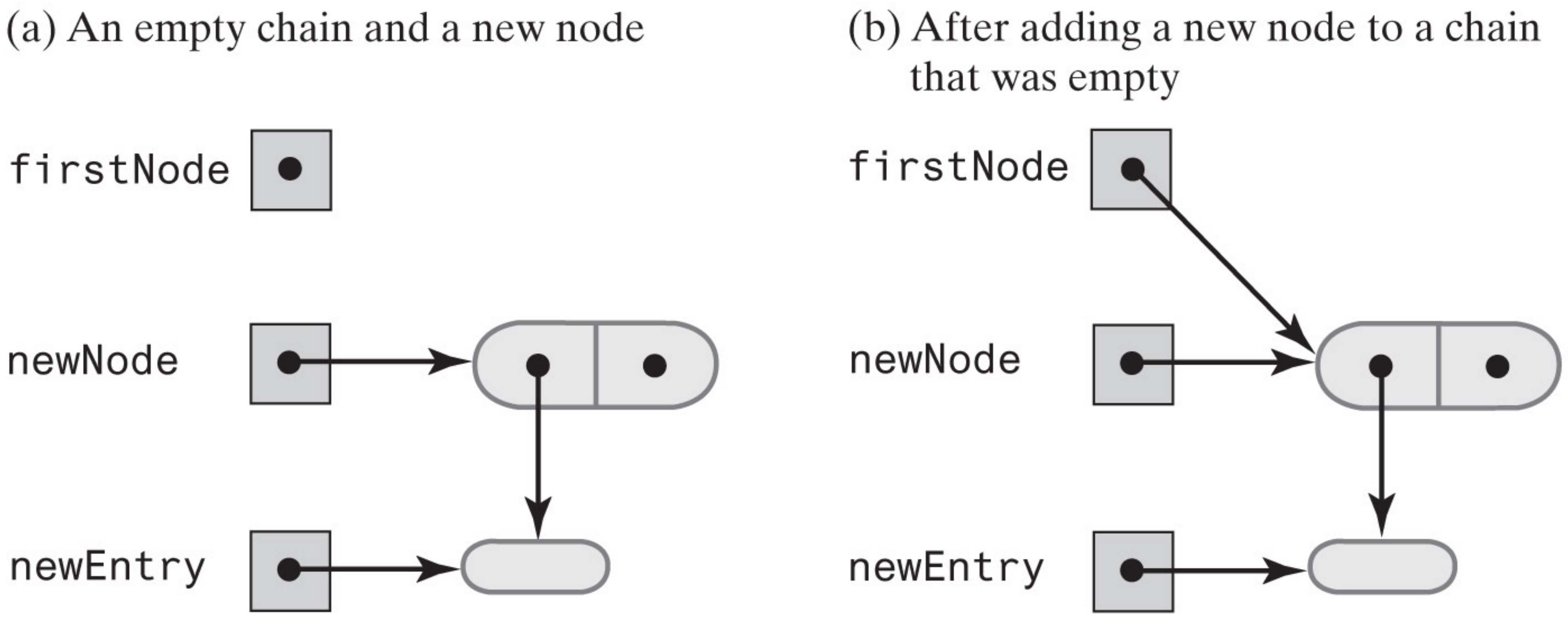

(a) An empty chain and a new node
(b) After adding a new node to a chain that was empty
firstNode
newNode
newEntry
firstNode
newNode
newEntry

Figure 3-6: Adding a new node to an empty chain. (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

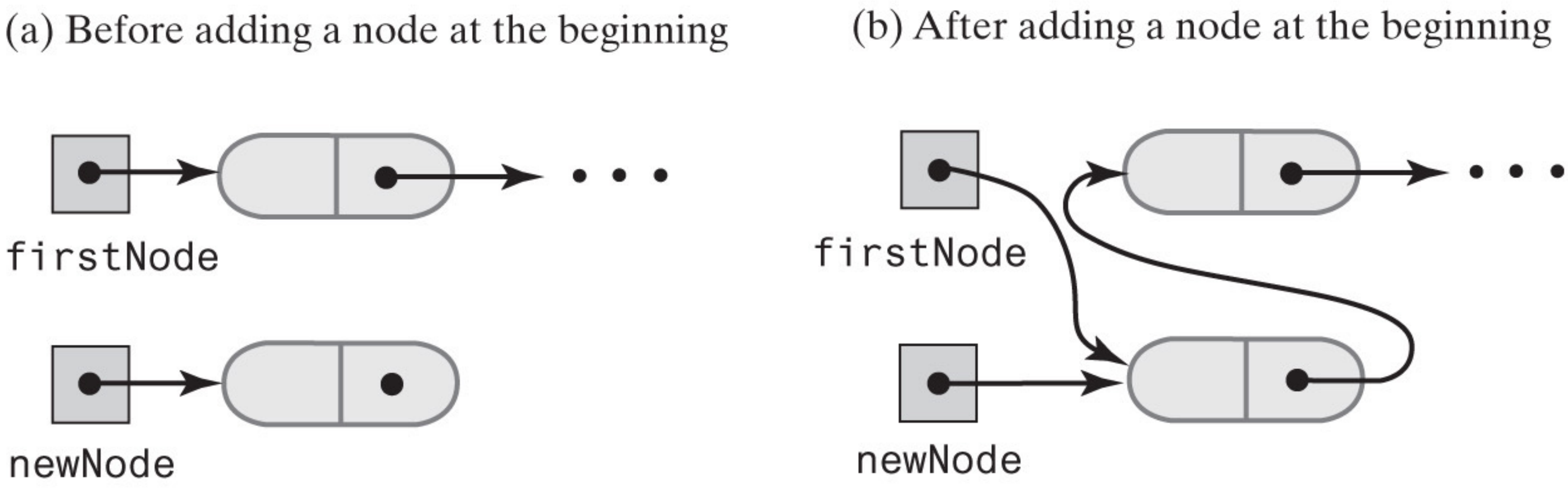

(a) Before adding a node at the beginning
firstNode
newNode
(b) After adding a node at the beginning
firstNode
newNode

Figure 3-7: A chain of nodes just before and just after adding a node at the beginning. (a) Before adding a node at the beginning; (b) 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`

```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` returns an array of the entries currently in a bag

Question

- In the previous definition of `toArray`, the `while()` statement uses following Boolean expression to control the loop.

```
(index < numberOfEntries) && (currentNode != null)
```

- Is it necessary to involve the values of both `index` and `currentNode`? Why?

Answer

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

Testing the Core Methods

Test Program - Part 1

```
/** A test of the methods add, toArray, isEmpty, and getCurrentSize,
    as defined in the first draft of the class LinkedBag. */
public class LinkedBagDemo1
{
   public static void main(String[] args)
   {
      System.out.println("Creating an empty bag.");
      BagInterface<String> aBag = new LinkedBag1<>();
      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 - Part 2

```
// Tests the method isEmpty.
// Precondition: If 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())
      System.out.println("empty: OK.");
   else if (empty)
      System.out.println("not empty, but it is: ERROR.");
   else if (!empty && bag.isEmpty())
      System.out.println("empty, but it is not empty: ERROR.");
   else
      System.out.println("not empty: OK.");
} // end testIsEmpty
```

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

Test Program - Part 3

```java
// Tests the method add.
private static void testAdd(BagInterface<String> aBag, String[] content)
{
   System.out.print("Adding the following strings to the bag: ");
   for (int index = 0; index < content.length; index++)
   {
      if (aBag.add(content[index]))
         System.out.print(content[index] + " ");
      else
         System.out.print("\nUnable to add " + content[index] +
                          " to the bag.");
   } // end for
   System.out.println();

   displayBag(aBag);
} // end testAdd
```

Listing 3-3: A sample program that tests some methods in the class `LinkedBag`

Test Program - Part 4

```
  // Tests the method toArray while displaying the bag.
   private static void displayBag(BagInterface<String> aBag)
   {
      System.out.println("The bag contains the following string(s):");
      Object[] bagArray = aBag.toArray();
      for (int index = 0; index < bagArray.length; index++)
      {
         System.out.print(bagArray[index] + " ");
      } // end for

      System.out.println();
   } // end displayBag

} // end LinkedBagDemo1
```

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

Test Program - Result

Program Output

```
Creating an empty bag.

Testing isEmpty with an empty bag:
isEmpty finds the bag empty: OK.
The bag contains the following string(s):

Adding the following strings to the bag: A D B A C A D
The bag contains the following string(s):
D A C A B D A

Testing isEmpty with a bag that is not empty:
isEmpty finds the bag not empty: OK.
```

Method getFrequencyOf

```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 loopCounter = 0;
   Node currentNode = firstNode;

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

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

   return frequency;
} // end getFrequencyOf
```

Counts the number of times a given entry appears

Method `contains`

```java
/** Tests whether this bag contains a given entry.
    @param anEntry  The entry to locate.
    @return  True if the bag contains anEntry, or false otherwise */

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

Question

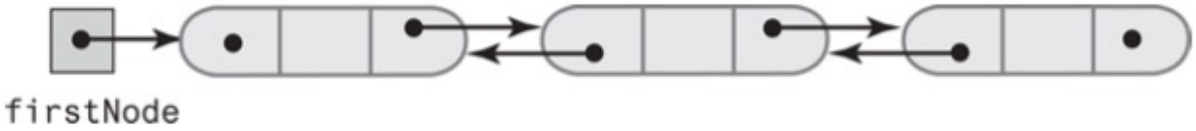
- If `currentNode` in the previous method `contains()` becomes `null`, what value does the method return when the bag is not empty?

Answer

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

Exercise

- In a doubly linked chain, each node can reference the previous node as well as the next node. Following figure shows a doubly linked chain and its head reference.



- Define a class to represent a node in a doubly linked chain. Write the class as an inner class of a class that implements the ADT bag. You can omit set and get methods.

Answer

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

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

    private DoublyLinkedNode(T dataPortion, DoublyLinkedNode nextNode,
                            DoublyLinkedNode previousNode)
    {
        data = dataPortion;
        next = nextNode;
        previous = previousNode;
    } // end constructor
} // end DoublyLinkedNode
```