Data Structures for Java

William H. Ford
William R. Topp

Chapter 10
Linked Lists B

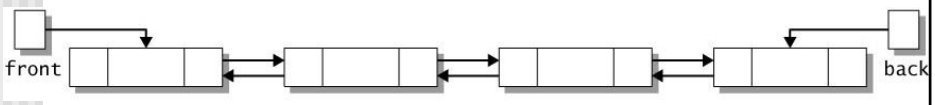
Bret Ford

Doubly Linked Lists

- Doubly-linked list nodes contain two references that point to the next and previous node.
- Such a list has a reference, **`front`**, that points to the first node in the sequence and a reference, **back**, that points at the last node in the sequence.

Doubly Linked Lists (2)

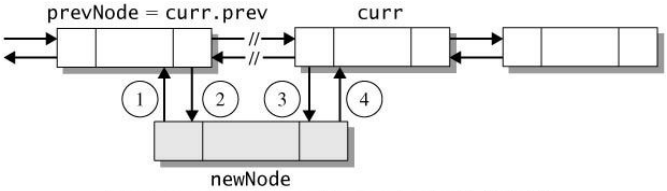
- You can scan a doubly-linked list in both directions. The forward scan starts at front and ends when the link is a reference to back. In the backward direction, reverse the process and the references.
- Like a singly-linked list, a doubly-linked list is a sequential structure.
- To move forward or backward in a doubly-linked list use the node links next and prev.
- Insert and delete operations need to have only the reference to the node in question.



Doubly Linked Lists (4)

- Inserting into a doubly linked list requires four reference assignments.

```
prevNode = curr.prev;
newNode.prev = prevNode;   // statement 1
prevNode.next = newNode;   // statement 2
curr.prev = newNode;       // statement 3
newNode.next = curr;       // statement 4
```

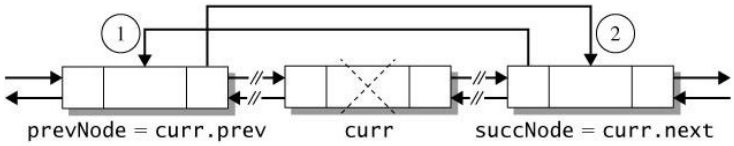


Inserting a new node at position curr in a doubly linked list.

Doubly Linked Lists (5)

- To delete a node curr, link the predecessor (curr.prev) of curr to the successor of curr (curr.next).

```
prevNode = curr.prev;
succNode = curr.next;
succNode.prev = prevNode;  // statement 1
prevNode.next = succNode;  // statement 2
```



Doubly Linked Lists (6)

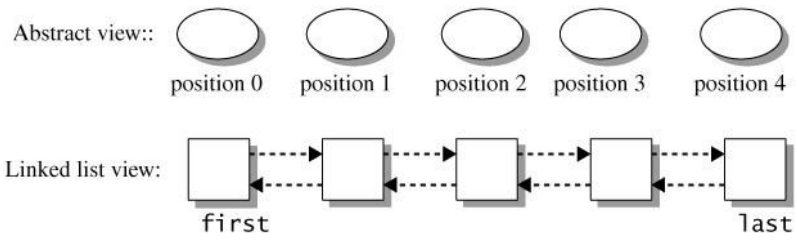
- In a singly-linked list, adding and removing a node at the front of the list are O(1) operation.
- With a doubly-linked list, you can add and remove a node at the back of the list with same runtime efficiency. Simply update the reference back.

The LinkedList Collection

- An ArrayList uses an array in contiguous memory.
- A LinkedList uses a doubly-linked list whose elements reside in noncontiguous memory locations.
- View a LinkedList collection as a disjoint sequence of nodes starting at the first element and proceeding in successive order to a final element. Motion can be forward or backward from a given node.

The LinkedList Collection (2)

- The figure provides views of a LinkedList collection as an abstract sequence and as a linked list.



The LinkedList Collection (3)

- The LinkedList class has a default constructor that creates an empty list.
- A toString() method returns a string representing the list as a comma-separated sequence of elements enclosed in brackets.

The LinkedList Collection (4)

- By implementing the List interface, the class also implements the Collection interface with its familiar general purpose collection methods isEmpty(), size(), contains(), and toArray().
- The collection add() method inserts an element. Since a LinkedList allows duplicates, add() will always insert a new element at the back of the list and return true.

The LinkedList Collection (5)

- A call to remove() with an Object reference deletes the first occurrence of the object in the list. The method returns true or false depending on whether a match occurs.

LinkedList Example

1. Use the constructor to create an empty linked list.

```
LinkedList<String> aList = new LinkedList<String>();
```

2. Assume the list contains the strings "Red", "Blue", "Green". Output its size and check whether aList contains the color "White".

```
System.out.println("Size = " +  aList.size());
System.out.println("List contains the string 'White' is "
                                + aList.contains("White");
```

```
Output:  Size = 3
         List contains the string 'White' is false
```

LinkedList Example (2)

Add the color "Black" and a second element with color "Blue". Then delete the first occurrence of "Blue".
An output statement uses `toString()` to list the elements in the sequence.

```
aList.add("Black");           // add Black at the end
aList.add("Blue");            // add Blue at the end
aList.remove("Blue");         // delete first "Blue"
System.out.println(aList);    // uses toString()
```

```
Output: [Red, Green, Black, Blue]
```

LinkedList Index Methods

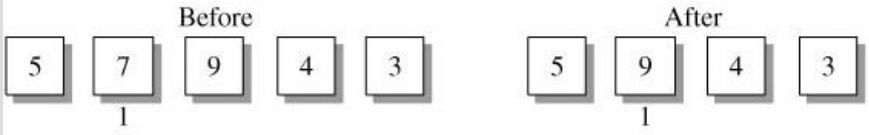
- The LinkedList class implements the indexed-based methods that characterize the List interface.
- A collection can access and update an element with the **get()** and **set()** methods and modify the list with the **add(index, element)** and **remove(index)** methods.
- The index methods have O(n) worst case running time. Use these methods only for small data sets.

LinkedList Index Methods Example

Assume the collection, list, initially contains elements with values [5, 7, 9, 4, 3].

Use `get()` to access the object at index 1 and then remove the element. The element at index 1 then has the value 9.

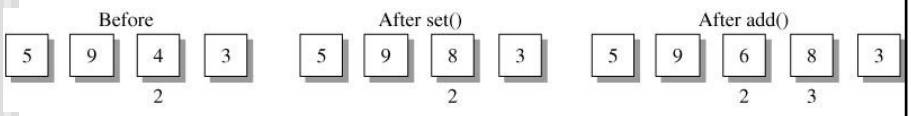
```
Integer intObj = list.get(1); // intObj has value 7
list.remove(1);
```



LinkedList Index Methods Example (concluded)

Use `set()` to update the element at index 2. Give it the value 8. Add a new element with value 6 at index 2. The new element occupies position 2, and its insertion shifts the tail of the list up one position. Thus the node at index 3 has value 8.

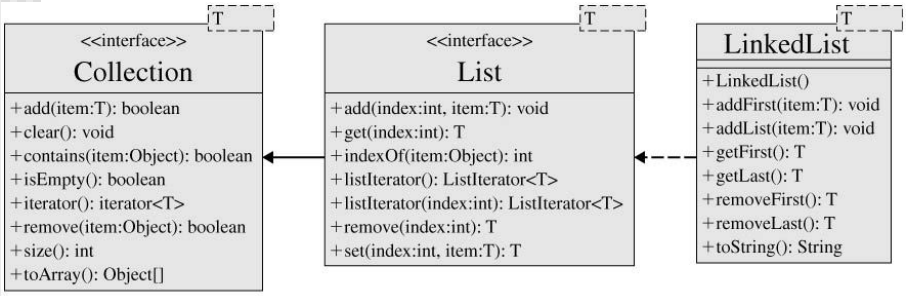
```
list.set(2, 8);
list.add(2, 6);
```



Accessing the Ends of a LinkedList

- A series of O(1) operations access and update the elements at the ends of the list.
- For the front of the list, the class defines the methods getFirst(), addFirst(), and removeFirst().
- The counterparts at the back of the list are getLast(), addLast(), and removeLast().
- A linked list is a natural storage structure for implementing a queue. The element at the front (getFirst()) is the one that exits (removeFirst()) the queue. A new element enters (addLast()) at the back of the queue.

UML for the LinkedList Class

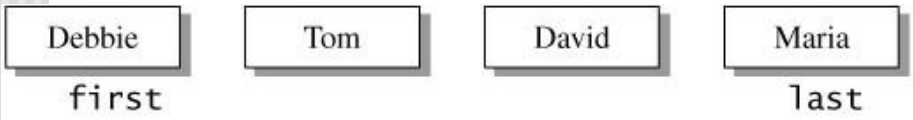


End-of-List Methods Example

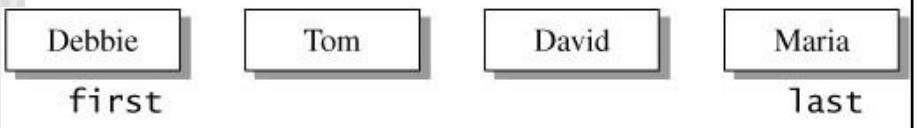
The "add" methods build the list by adding a new element. Observe that successive calls to `addFirst()` inserts elements in reverse order; successive calls to `addLast()` inserts the elements in the normal order.

```
list.addFirst("Tom");
list.addFirst("Debbie");

list.addLast("David");
ist.addLast("Maria");
```



End-of-List Methods Example (2)



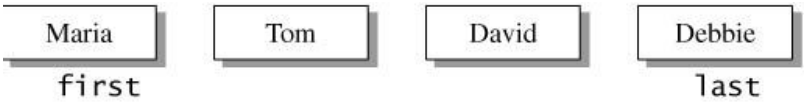

```
// identify the elements at the ends of the list
System.out.println("First element is " + list.getFirst());
System.out.println("Last element is " + list.getLast());
```

```
Output:  First element is Debbie
         Last element is Maria
```

End-of-List Methods Example (3)

Exchange the first and last elements in the list.

```
String firstElement, lastElement;
// remove the elements at the ends of the list and capture
// their values
firstElement = aList.removeFirst();
lastElement = aList.removeLast();
// add the elements back into the list with firstElement
// at the back and lastElement at the front
aList.addLast(firstElement);
aList.addFirst(lastElement);
```



End-of-List Methods Example (4)

Output the elements in the list by position. Repeatedly delete the first element and display its value until the list is empty.

```
while (!aList.isEmpty())
   System.out.print(aList.removeFirst() + "  ");
```

```
Output:
   Maria  Tom  David  Debbie
```

Program 10.2

```
import ds.util.LinkedList;
import java.util.Scanner;

public class Program10_2
{
   public static void main(String[] args)
   {
      // create an empty linked list
      LinkedList<String> draftlist =
         new LinkedList<String>();

      // variables used to update the draft list
      int fromIndex, toIndex;
      char updateAction;
      String playerName;
      String obj;
```

Program 10.2 (continued)

```
// initial names in the list and the
// keyboard input file
String[] playerArr ={"Jones", "Hardy",
   "Donovan", "Bundy"};
Scanner keyIn = new Scanner(System.in);
String inputStr;
// initialize the list
for (int i = 0; i < playerArr.length; i++)
   draftlist.add(playerArr[i]);

// give instructions on updating the list
System.out.println("Add player:    " +
   "Input 'a' <name>");
System.out.println("Shift player:  " +
   "Input 's' <from> <to>");
System.out.println("Delete player: " +
   "Input 'r' <name>" + "\n");
```

Program 10.2 (continued)

```
// initial list
System.out.println("List: " + draftlist);

// loop executes the simulation of draft updates
while (true)
{
   // input updateAction, exiting on 'q'
   System.out.print("   Update: ");
   updateAction = keyIn.next().charAt(0);

   if (updateAction == 'q')
      break;

   // execute the update
   switch(updateAction)
   {
```

Program 10.2 (continued)

```
      case 'a':
         // input the name and add to end of list
         playerName = keyIn.next();
         draftlist.add(playerName);
         break;
      case 'r':
         // input the name and remove from list
         playerName = keyIn.next();
         draftlist.remove(playerName);
         break;
      case 's':
         // input two indices to shift an
         // element from a source position
         // to a destination position;
         // remove element at source and
         // add at destination
         fromIndex = keyIn.nextInt();
```

Program 10.2 (concluded)

```
                // set to list position
                fromIndex--;
                toIndex = keyIn.nextInt();
                // set to list position
                toIndex--;
                obj = draftlist.remove(fromIndex);
                draftlist.add(toIndex, obj);
                break;
            }
            // Display status of current draft list
            System.out.println("List: " + draftlist);
        }
    }
}
```

Program 10.2 (Run)

```
Run:

Add player:    Input 'a' <name>
Shift player:  Input 's' <from> <to>
Delete player: Input 'r' <name>

List: [Jones, Hardy, Donovan, Bundy]
   Update: a Harrison
List: [Jones, Hardy, Donovan, Bundy, Harrison]
   Update: s 4 2
List: [Jones, Bundy, Hardy, Donovan, Harrison]
   Update: r Donovan
List: [Jones, Bundy, Hardy, Harrison]
   Update: a Garcia
List: [Jones, Bundy, Hardy, Harrison, Garcia]
   Update: s 5 2
List: [Jones, Garcia, Bundy, Hardy, Harrison]
   Update: s 1 4
List: [Garcia, Bundy, Hardy, Jones, Harrison]
   Update: q
```

Palindromes

- A palindrome is a sequence of values that reads the same forward and backward. "level" is a palindrome.
- The method, isPalindrome(), takes a LinkedList object as an argument and returns the boolean value true if the sequence of elements is a palindrome and false otherwise.
- The algorithm compares the elements on opposite ends of the list, using getFirst() and getLast().

isPalindrome()

In the implementation of isPalindrome(), the return Type (boolean) does not depend on a named generic type. Likewise, the parameter list does not require a named generic type. In this situation, we use a wildcard in the method signature. The syntax

```
LinkedList<?> aList
```

means that aList is a LinkedList object whose elements are of unknown type.

isPalindrome() (concluded)

```
public static boolean isPalindrome(LinkedList<?> aList)
{
   // check values at ends of list as
   // long as list size > 1
   while (aList.size() > 1)
   {
      // compare values on opposite ends; if not equal,
      // return false
      if (aList.getFirst().equals(aList.getLast())
            == false)
         return false;
      // delete the objects
      aList.removeFirst();
      aList.removeLast();
   }
   // if still have not returned, list is a palindrome
   return true;
}
```

Program 10.3

```
import ds.util.LinkedList;
import java.util.Scanner;

public class Program10_3
{
   public static void main(String[] args)
   {
      String str;
      LinkedList<Character> charList =
         new LinkedList<Character>();
      Scanner keyIn = new Scanner(System.in);
      int i;
      char ch;
      // prompt user to enter a string that may include
      // blanks and punctuation marks
System.out.print("Enter the string: ");
      str = keyIn.nextLine();
```

Program 10.3 (continued)

```
      // copy all of the letters as
      // lowercase characters to the
      // linked list charList
      for (i = 0; i < str.length(); i++)
      {
         ch = str.charAt(i);

         if (Character.isLetter(ch))
            charList.addLast(Character.toLowerCase(ch));
      }

      // call isPalindrome() and use return
      // value to designate whether the string
      // is or is not a palindrome
      if (isPalindrome(charList))
         System.out.println("'" + str +
                            "' is a palindrome");
```

Program 10.3 (concluded)

```
      else
         System.out.println("'" + str +
                            "' is not a palindrome");
   }
   < Code for method isPalindrome() >
}
```

```
      Run 1:
      Enter the string: A man, a plan, a canal, Panama
      'A man, a plan, a canal, Panama' is a palindrome

      Run 2:
      Enter the string: Go hang a salami, I'm a lasagna hog
      'Go hang a salami, I'm a lasagna hog' is a palindrome

      Run 3:
      Enter the string: palindrome
      'palindrome' is not a palindrome
```