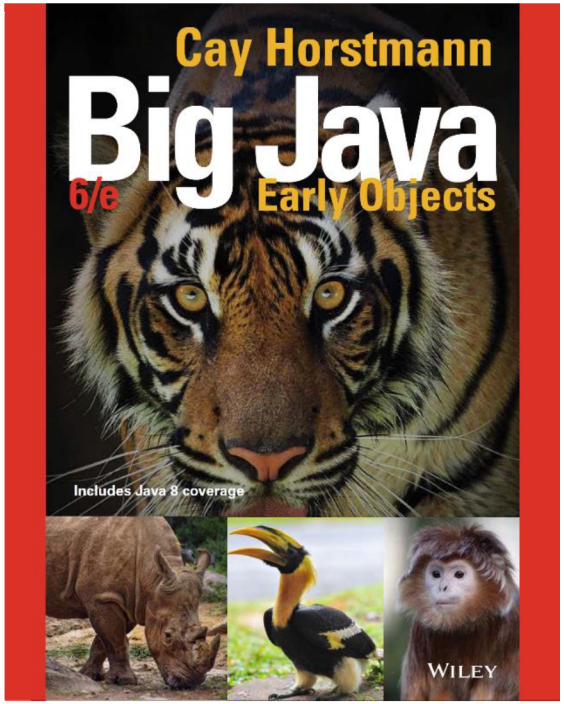

Big Java Early Objects by Cay Horstmann

Chapter 15 – An Introduction to Data Structures:

Part I: LinkedLsits

Chapter Goals

- To learn how to use the linked lists provided in the standard library
- To be able to use iterators to traverse linked lists

Using Linked Lists

- A linked list consists of a number of nodes, each of which has a reference to the next node
- Adding and removing elements in the middle of a linked list is efficient
- Visiting the elements of a linked list in sequential order is efficient
- Random access is not efficient

Inserting an Element into a Linked List

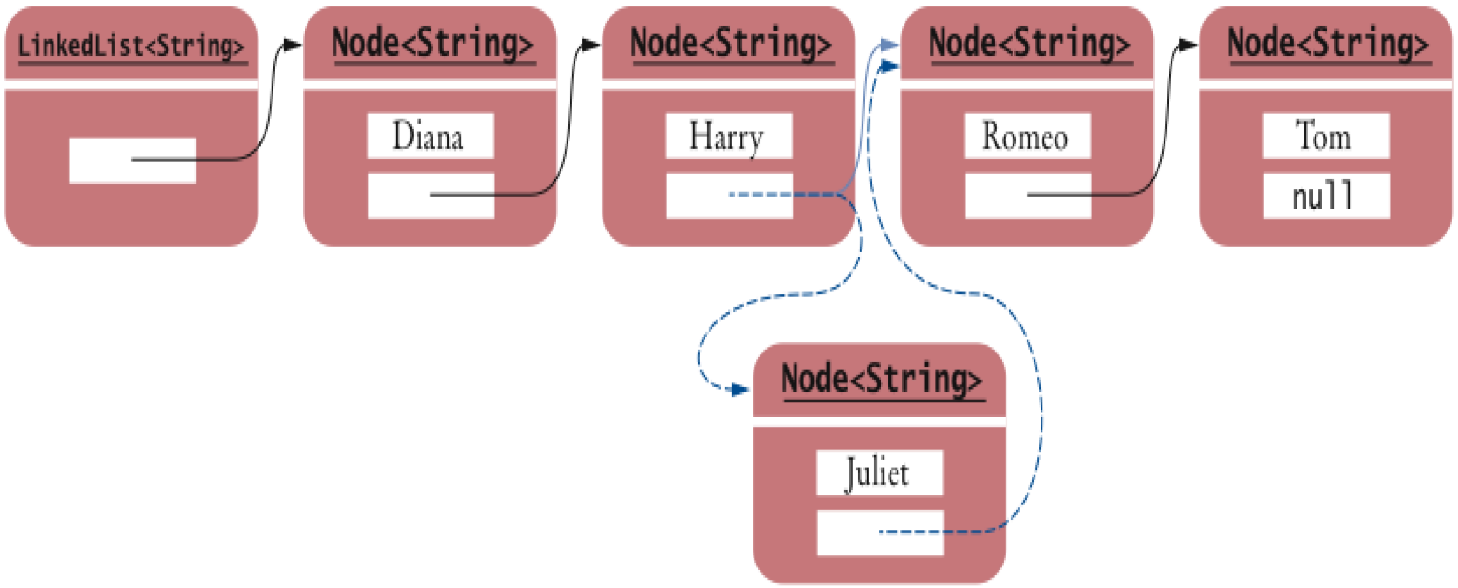

Figure 1 Inserting an Element into a Linked List

Java's `LinkedList` class

- Generic class
  - *Specify type of elements in angle brackets:* `LinkedList<Product>`
- Package: `java.util`

Table 1 LinkedList Methods

`LinkedList<String> lst = new LinkedList<String>();`	An empty list.
`lst.addLast("Harry")`	Adds an element to the end of the list. Same as `add`.
`lst.addFirst("Sally")`	Adds an element to the beginning of the list. `lst` is now `[Sally, Harry]`.
`lst.getFirst()`	Gets the element stored at the beginning of the list; here `"Sally"`.
`lst.getLast()`	Gets the element stored at the end of the list; here `"Harry"`.
`String removed = lst.removeFirst();`	Removes the first element of the list and returns it. `removed` is `"Sally"` and `lst` is `[Harry]`. Use `removeLast` to remove the last element.
`ListIterator<String> iter = lst.listIterator()`	Provides an iterator for visiting all list elements (see Table 2 on page 634).

List Iterator

- `ListIterator` type
- Gives access to elements inside a linked list
- Encapsulates a position anywhere inside the linked list
- Protects the linked list while giving access

A List Iterator

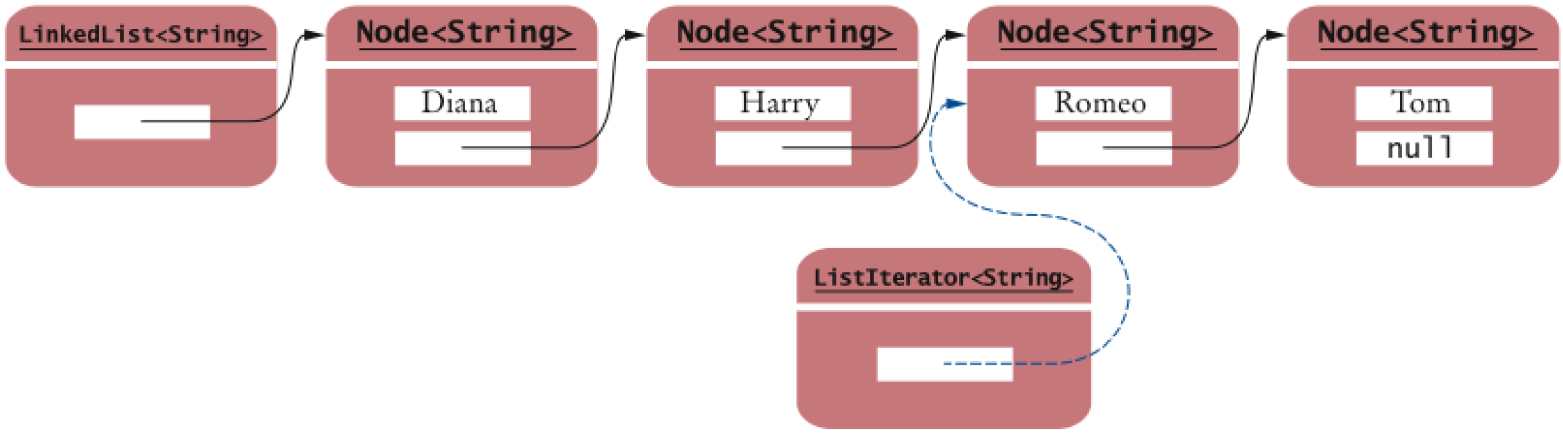

Figure 2 A List Iterator

A Conceptual View of the List Iterator

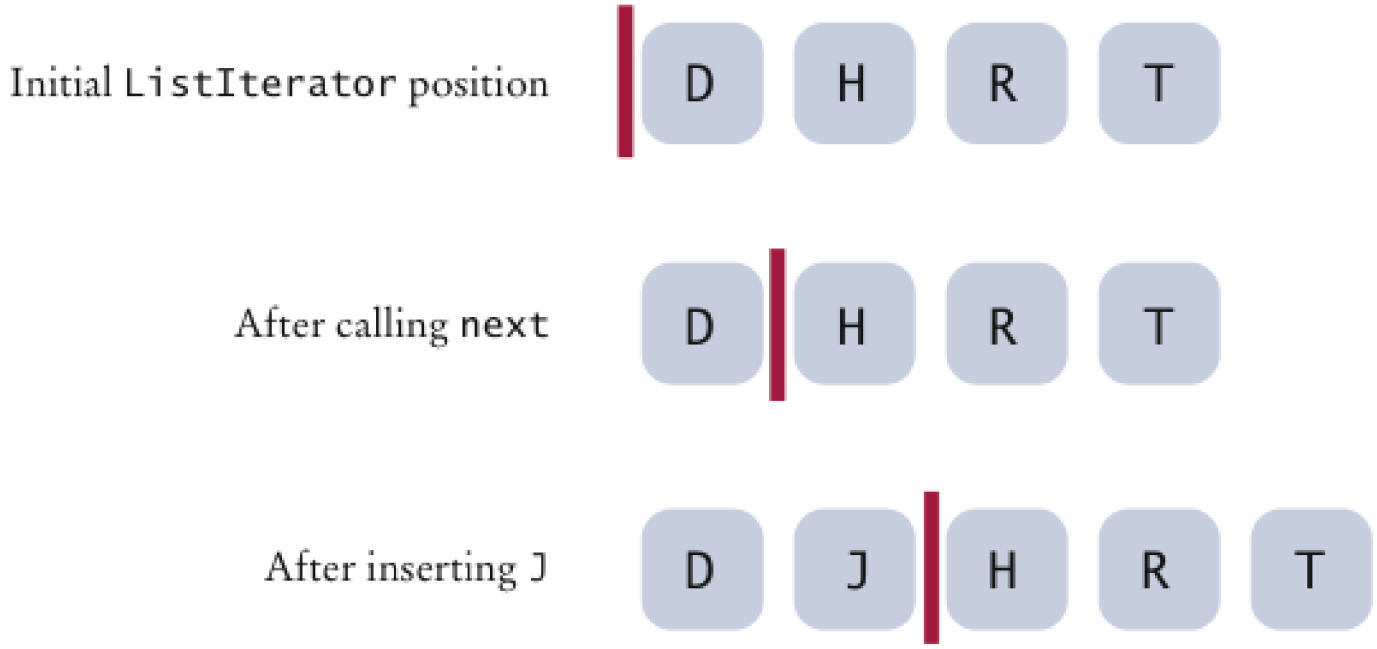

Figure 3 A Conceptual View of the List Iterator

List Iterator

- Think of an iterator as pointing between two elements
  - *Analogy: Like the cursor in a word processor points between two characters*
- The `listIterator` method of the `LinkedList` class gets a list iterator

```
LinkedList<String> employeeNames = ...;
ListIterator<String> iterator =
   employeeNames.listIterator();
```

List Iterator

- Initially, the iterator points before the first element
- The `next` method moves the iterator:

```
iterator.next();
```

- `next` throws a `NoSuchElementException` if you are already past the end of the list
- `hasNext` returns true if there is a next element:

```
if (iterator.hasNext())
   iterator.next();
```

List Iterator

- The `next` method returns the element that the iterator is passing:

```
while iterator.hasNext()
{
   String name = iterator.next();
   Do something with name
}
```

- Shorthand:

```
for (String name : employeeNames)
{
   Do something with name
}
```

Behind the scenes, the `for` loop uses an iterator to visit all list elements

List Iterator

- `LinkedList` is a *doubly linked list*
  - *Class stores two links:*
    - *One to the next element, and*
    - *One to the previous element*
- To move the list position backwards, use:
  - *`hasPrevious`*
  - *`previous`*

Adding and Removing from a `LinkedList`

- The `add` method:
  - *Adds an object after the iterator*
  - *Moves the iterator position past the new element:*

```
iterator.add("Juliet");
```

Adding and Removing from a `LinkedList`

- The `remove` method
  - *Removes and*
  - *Returns the object that was returned by the last call to `next` or `previous`*

```
//Remove all names that fulfill a certain condition
while (iterator.hasNext())
{
    String name = iterator.next();
    if (name fulfills condition)
       iterator.remove();

}
```

Adding and Removing from a `LinkedList`

- Be careful when calling `remove`:
  - *It can be called only once after calling `next` or `previous`:*

```
iterator.next();
iterator.next();
iterator.remove();
iterator.remove();
// Error: You cannot call remove twice.
```

  - *You cannot call it immediately after a call to `add`:*

```
iter.add("Fred");
iter.remove(); // Error: Can only call remove after
               // calling next or previous
```

  - *If you call it improperly, it throws an `IllegalStateException`*

Methods of the `ListIterator` Interface

Table 2 Methods of the `ListIterator` Interface

`String s = iter.next();`	Assume that `iter` points to the beginning of the list `[Sally]` before calling `next`. After the call, `s` is `"Sally"` and the iterator points to the end.
`iter.hasNext()`	Returns `false` because the iterator is at the end of the collection.
`if (iter.hasPrevious()) { s = iter.previous(); }`	`hasPrevious` returns `true` because the iterator is not at the beginning of the list.
`iter.add("Diana");`	Adds an element before the iterator position. The list is now `[Diana, Sally]`.
`iter.next(); iter.remove();`	`remove` removes the last element returned by `next` or `previous`. The list is again `[Diana]`.

Sample Program

- `ListTester` is a sample program that
  - *Inserts strings into a list*
  - *Iterates through the list, adding and removing elements*
  - *Prints the list*

ch15/uselist/ListTester.java

```
 1  import java.util.LinkedList;
 2  import java.util.ListIterator;
 3
 4  /**
 5     A program that tests the LinkedList class
 6  */
 7  public class ListTester
 8  {
 9     public static void main(String[] args)
10     {
11        LinkedList<String> staff = new LinkedList<String>();
12        staff.addLast("Diana");
13        staff.addLast("Harry");
14        staff.addLast("Romeo");
15        staff.addLast("Tom");
16
17        // | in the comments indicates the iterator position
18
19        ListIterator<String> iterator = staff.listIterator(); // |DHRT
20        iterator.next(); // D|HRT
21        iterator.next(); // DH|RT
22
```

Continued

ch15/uselist/ListTester.java (cont.)

```
23          // Add more elements after second element
24
25          iterator.add("Juliet"); // DHJ|RT
26          iterator.add("Nina"); // DHJN|RT
27
28          iterator.next(); // DHJNR|T
29
30          // Remove last traversed element
31
32          iterator.remove(); // DHJN|T
33
34          // Print all elements
35
36          for (String name : staff)
37             System.out.print(name + " ");
38          System.out.println();
39          System.out.println("Expected: Diana Harry Juliet Nina Tom");
40       }
41    }
```

Continued

ch15/uselist/ListTester.java (cont.)

Program Run:

```
Diana Harry Juliet Nina Tom
Expected: Diana Harry Juliet Nina Tom
```

Self Check 15.1

Do linked lists take more storage space than arrays of the same size?

Answer: Yes, for two reasons. You need to store the node references, and each node is a separate object. (There is a fixed overhead to store each object in the virtual machine.)

Self Check 15.2

Why don't we need iterators with arrays?

Answer: An integer index can be used to access any array location.