The LinkedList Class

- Reading: Savitch, Chapter 10

Objectives

- To learn to use the StudentNode and linked list class to write the program.
- To learn how to create and manipulate a linked list by using the LinkedList class defined in *java.util*.

An Example

```
//The program builds up a linked list and operates on it
//StudentNode.java
class StudentNode {
    String name;          // Note: package access is used only
    int mark;             //       for teaching purposes, would
    StudentNode next; //       be private
    public StudentNode(String _n, int _m) {
        name = _n;  mark = _m;
        next = null;
    }
}
```

```
//StLinkedList.java
class StLinkedList {
    private StudentNode head = null;

    public void insertInOrder(String _name, int _mark){
     StudentNode temp = new StudentNode(_name, _mark);

     if (head == null)
         head = temp;
     else
       // NOTE: compareTo is a method of the String class.  It returns
       // a positive int if the String is lexicographically greater than
       // the parameter, a negative int if it's smaller and 0 if they are equal
       if ((head.name).compareTo(temp.name) > 0) {
            temp.next = head;
            head = temp;
        }
```

```
        else {
                StudentNode p = head;
                while ((p.next != null) &&
                                (((p.next).name).compareTo (temp.name) < 0))
                {               p = p.next;     }
                temp.next = p.next;
                p.next = temp;
            }
        }

    public void printList() {
        StudentNode p = head;

        while (p != null) {
            System.out.println (p.name + ": " + p.mark);
            p = p.next;
        }
    }
```

```
public void remove(String _name) {
    if ((head.name).compareTo( _name) == 0)
        head = head.next;
    else
    {
        StudentNode p = head, previous = head;
        while ( (p != null) && (_name.compareTo(p.name) != 0)) {
            previous = p;
            p = p.next;
        }

        if(p == null)
            System.out.print("Node not found. Nothing removed");
        else
            previous.next = p.next;
    }
  }
} // end of class from slide 4
```

```
//ListTest.java
public class ListTest {
    public static void main(String [ ] args) {

        StLinkedList myList = new StLinkedList();
        myList.insertInOrder("Laura", 73);
        myList.insertInOrder("Alice", 85);
        myList.insertInOrder("James", 56);
        myList.insertInOrder("Wendy", 91);
        myList.printList();
        System.out.println();
        myList.remove("Laura");
        myList.printList();

    }
}
```

The program execution

%java ListTest

Alice: 85

James: 56

Laura: 73

Wendy: 91

Alice: 85

James: 56

Wendy: 91

The LinkedList Class

- LinkedList is defined in *java.util* for building a linked list.
- LinkedList contains a number of methods for operations such as insertion, deletion and traversal etc.

Methods in LinkedList

- void add(int index, Object element)

  Inserts the specified element at the specified position in this list.

- boolean add(Object o)

  Appends the specified element to the end of this list.

- void addFirst(Object o)

  Inserts the given element at the beginning of this list.

- void addLast(Object o)
  Appends the given element to the end of this list.

- void clear()
  Removes all of the elements from this list.

- boolean contains(Object o)
  Returns true if this list contains the specified element.

- Object get(int index)
  Returns the element at the specified position in this list.

- Object getFirst()
  Returns the first element in this list.

- Object getLast()
  Returns the last element in this list.

- int indexOf(Object o)
  Returns the index in this list of the first occurrence of the specified element, or -1 if the list does not contain this element.

- int lastIndexOf(Object o)
  Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

- Object remove(int index)
  Removes the element at the specified position in this list.

- boolean remove(Object o)
  Removes the first occurrence of the specified element in this list.

- Object set(int index, Object element)
  Replaces the element at the specified position in this list with the specified element.

- int size()
  Returns the number of elements in this list.

- Object[] toArray()

  Returns an array containing all of the elements in this list in the correct order.

- ListIterator listIterator(int index)

  Returns a ListIterator object which is pointing to the specified position in the list.

Primitive Auto-boxing/unboxing

- The LinkedList class (and other collections classes) are designed to take objects rather than primitives.
- Thus when you insert a primitive such as an `int` rather than an object such as `Integer` into a LinkedList, the compiler automatically converts it to type `Integer`. This is called Auto-boxing.
- On retrieval of the value from the collection the compiler again converts it from its object to its primitive type, this is called Auto-unboxing.

Primitive Auto-boxing/unboxing

Example

```
LinkedList mylist = new LinkedList();
int a = 200;                                  //This is a primitive
String name = “Alice”;                        //This is an object
Integer x = new Integer(200);                 //This is an object
mylist.add(name);                             //No auto-boxing required
mylist.add(x);                                //No auto-boxing required
mylist.add(a);      //Auto-boxing - compiler will need to wrap the
                    //primitive int into an Integer object before adding
```

Primitive Auto-boxing/unboxing

- Essentially auto-boxing/unboxing means that you may use an `Integer` object as if it were a primitive `int`.
- Do not get too creative with this feature! Whilst auto-boxing/unboxing is very useful for collections (as demonstrated on the previous slide), it does have performance repercussions.
- Use this feature for collections but avoid using it elsewhere. Do not replace `int` with `Integer` objects, unless necessary.

The ListIterator Interface

- ListIterator is defined in *java.util*.
- A variable of ListIterator is somehow like a pointer pointing to a link. Operations such as adding, removing and re-setting occur around the link.
- We normally use the LinkedList and ListIterator classes together to build a linked list.

Methods in ListIterator

- void add(Object o)
  Inserts the specified element into the list.

- boolean hasNext()
  Returns true if this list iterator has more elements when traversing the list in the forward direction.

- boolean hasPrevious()
  Returns true if this list iterator has more elements when traversing the list in the reverse direction.

- Object next()

  Returns the next element in the list.

- int nextIndex()

  Returns the index of the element that would be returned by a subsequent call to next.

- Object previous()

  Returns the previous element in the list.

- int previousIndex()

  Returns the index of the element that would be returned by a subsequent call to previous.

- void remove()

  Removes from the list the last element that was returned by next or previous (optional operation).

- void set(Object o)

  Replaces the last element returned by next or previous with the specified element.

An Example of LinkedList and ListIterator

```
//ListTest.java by Horstmann
import java.util.LinkedList;
import java.util.ListIterator;

public class ListTest {
   public static void main(String [ ] args) {
      LinkedList staff = new LinkedList();
      staff.addLast("Dick");
      staff.addLast("Harry");
      staff.addLast("Romeo");
      staff.addLast("Tom");
```

```
        ListIterator iterator = staff.listIterator(); // |DHRT
        iterator.next(); // D|HRT
        iterator.next(); // DH|RT
        iterator.add("Juliet"); // DHJ|RT
        iterator.add("Nina"); // DHJN|RT
        iterator.next(); // DHJNR|T
        iterator.remove(); // DHJN|T

        iterator = staff.listIterator();
        while (iterator.hasNext())
           System.out.println(iterator.next());
    }
}
```