Linked List

The Class LinkedList

- A **linked data structure** is a collection of objects, each of which contains data and a reference to another object in the collection.
- Like ArrayList, LinkedList is another implementation of the ADT list.
- After importing LinkedList from the package java.util. you create a new instance of LinkedList from the package java.util, you create a new instance of LinkedList by invoking its default constructor.

- For example the following statement creates myList as a list of strings:

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

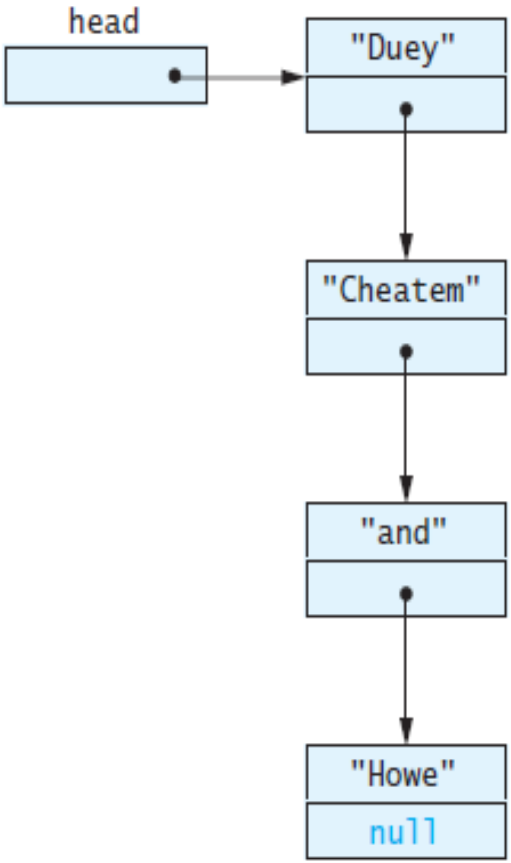
- You can then go on to use the methods of linkedList –such as Add, set, get, remove and size.

- Both of the classes ArrayList and LinkedList implement the interface List

Linked Lists

- A linked list is a dynamic data structure that links the items in a list to one another.
- Like all linked data structures, a linked list consists of objects known as nodes.
- In the figure, the nodes are boxes that are divided in half by a horizontal line.
- Each node has a place for some data and a place to hold a link to another node.
- The links are shown as arrows that point to the node they “link” to.

FIGURE 12.4 A Linked List



The LinkedList Class

- **LinkedList** is a generic class that has this declaration

  class LinkedList<E>

- Here, **E** specifies the type of objects that the list will hold.

LinkedList

- Here we can see example for basic operations like creating object for LinkedList,
  - adding objects into LinkedList,
  - searching an object in LinkedList,
  - checking whether the LinkedList is empty or not,
  - and finally size of the LinkedList.

LinkedList

- List method addAll appends all elements of a collecton to the end of a `List`.
- List-Iterator method set replaces the current element to which the iterator refers with the specified object.
- List method subList obtain a a portion of a `List`.
  - This is a so-called range-view method, which enables the program to view a portion of the list.

`LinkedList` (cont.)

- List method clear remove the elements of a `List`.
- List method size returns the number of items in the `List`.
- ListIterator method hasPrevious determines whether there are more elements while traversing the list backward.
- ListIterator method previous gets the previous element from the list.

Iterators

- A variable that allows you to step through a collection of nodes in a linked list
  - For arrays, we use an integer
- Common to place elements of a linked list into an array
  - For display purposes, array is easily traversed

Iterators

- Consider an iterator that will move through a linked list
  - Allow manipulation of the data at the nodes
  - Allow insertion, deletion of nodes

Java `Iterator` Interface

- Java formally considers an iterator to be an object
- Note interface named Iterator with methods
  - **`hasNext`** – returns boolean value
  - **`next`** – returns next element in iteration
  - **`remove`** – removes element most recently returned by **`next`** method

Program: How to add all elements of a list to LinkedList?

Code:

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class MyLinkedListNewCollection {

    public static void main(String a[]){

        LinkedList<String> arrl = new LinkedList<String>();
        //adding elements to the end
        arrl.add("First");
        arrl.add("Second");
        arrl.add("Third");
        arrl.add("Random");
        System.out.println("Actual LinkedList:"+arrl);
        List<String> list = new ArrayList<String>();
        list.add("one");
        list.add("two");
        arrl.addAll(list);
        System.out.println("After Copy: "+arrl);
    }
}
```

Description:
Here we can see example for copying another collection instance objects to existing LinkedList.

Output:
Actual LinkedList:[First, Second, Third, Random] After Copy: [First, Second, Third, Random, one, two]

Program: How to copy LinkedList to array?

Code:

?

```
import java.util.LinkedList;

public class MyLinkedListArray {

    public static void main(String a[]){

        LinkedList<String> arrl = new LinkedList<String>();
        arrl.add("First");
        arrl.add("Second");
        arrl.add("Third");
        arrl.add("Random");
        System.out.println("Actual LinkedList:"+arrl);
        String[] strArr = new String[arrl.size()];
        arrl.toArray(strArr);
        System.out.println("Created Array content:");
        for(String str:strArr){
            System.out.println(str);
        }
    }
}
```

Description:
Here we can see example for copying all content of LinkedList to an array. You can get this done by calling toArray() method.

Output:
Actual LinkedList:[First, Second, Third, Random] Created Array content: First Second Third Random

Program: How to sort LinkedList using Comparator?

Description:

This example gives you how to sort a LinkedList using Comparator. The LinkedList contains user defined objects. By using Collections.sort() method you can sort the LinkedList. You have to pass Comparator object which contains your sort logic. The example sorts the Empl objects based on highest salary.

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

public class MyLinkedListSort
{

   public static void main(String a[])
{
      LinkedList<Empl> list = new LinkedList<Empl>();
      list.add(new Empl("Ram",3000));
      list.add(new Empl("John",6000));
      list.add(new Empl("Crish",2000));
      list.add(new Empl("Tom",2400));
      Collections.sort(list, new MySalaryComp());
      System.out.println("Sorted list entries: ");
      for(Empl e : list)
       {
         System.out.println(e);
      }
   }
}
```

- Check Full Code: EmplComp

Program: LinkedList push(), pop() operations examples.

```
import java.util.LinkedList;

public class MyPushPopOpr {

    public static void main(String a[]){

        LinkedList<String> arrl = new LinkedList<String>();
        arrl.add("First");
        arrl.add("Second");
        arrl.add("Third");
        arrl.add("Random");
        System.out.println(arrl);
        arrl.push("push element");
        System.out.println("After push operation:");
        System.out.println(arrl);
        arrl.pop();
        System.out.println("After pop operation:");
        System.out.println(arrl);
    }
}
```

Description:
Below example shows how to call push() and pop() methods on LinkedList objects. **push():** Pushes an element onto the stack represented by this list. **pop():** Pops an element from the stack represented by this list.

Output:
[First, Second, Third, Random] After push operation: [push element, First, Second, Third, Random] After pop operation: [First, Second, Third, Random]

Priority Queue Class

- A PriorityQueue is used when the objects are supposed to be processed based on the priority.
- It is known that a Queue follows the First-In-First-Out (FiFo) struture.
- But sometimes the elements of the queue are needed to be processed according to the priority. This is the main purpose of the PriorityQueue class!
- The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at the queue construction time, depending on which constructor is used.

- **peek():** The peek() method is used to look at the front of the queue without removing it. If the queue is empty, it returns a null value.
- **poll():** The poll() method removes the beginning of the queue and returns it. If the queue is empty, it returns a null value.

```
1  // Check code PriorityQueueTest.java
2  // Standard library class PriorityQueue test program.
3  import java.util.PriorityQueue;
4
5  public class PriorityQueueTest
6  {
7     public static void main( String args[] )
8     {
9        // queue
10       PriorityQueue< Double > queue = new PriorityQueue< Double >();
11
12       // insert elements to queue
13       queue.offer( 3.2 );
14       queue.offer( 9.8 );
15       queue.offer( 5.4 );
16
17       System.out.print( "Polling from queue: " );
18
19       // display elements in queue
20       while ( queue.size() > 0 )
21       {
22          System.out.printf( "%.1f ", queue.peek() ); // view top element
23          queue.poll(); // remove top element
24       } // end while
25    } // end main
26 } // end class PriorityQueueTest
```

```
Polling from queue: 3.2 5.4 9.8
```

Create a `PriorityQueue` that stores `Double`s and orders the elements according to the object's natural ordering

Use method `offer` to add elements to the priority queue

Use method `size` to determine whether the priority queue is empty

Use method `peek` to retrieve the highest-priority element in the queue

Use method `poll` to remove the highest-priority element from the queue

Student Priority Queue Demo

- Check code: Student and StudentPriorityQueueDemo