CS110 Assignment #9
Due *Wednesday*, November 13th

Exam #3 is scheduled for Thursday, November 14th. If you have ACCESS accommodations, please schedule with the ETC.

For this assignment, it is expected that you will use good programming practices, including appropriate use of exception handling. All classes should be fully tested, although you do not need to submit test drivers. You do not need to submit any files that were provided to you. See "what to submit" on page 3 of this assignment.

1. In class, we discussed a reference based implementation of a linked list. Another potential implementation would be `ArrayList` based. Both classes extend the `ListInterface`. I have provided a complete `ArrayList` implementation. In the driver file I provided, you'll find the line:

   ```
   ListInterface list1 = new ListArrayListBased();
   ```

   Implement the reference based linked list. After successfully implementing your reference based list, the following statement will also work:

   ```
   ListInterface list2 = new ListReferenceBased();
   ```

   Now, compare the run time of the operations of the list for both implementations. The method
   `System.currentTimeMillis();`
   will return the current time in milliseconds. If you take a time stamp, execute a segment of code and then take another time stamp, you can get a rough estimate of the time it took to complete the segment.

   ```
   long time1,time2, diff;
   time1 = System.currentTimeMillis();
   list1.add(i,new Integer(r.nextInt()));
   time2 = System.currentTimeMillis();
   diff = time2-time1;
   ```

   However, single operations take such little time to execute, you often have to perform the same operation a significant (100000?) number of times to get a reasonable time measurement. Experiment with the number of iterations to find a value that consistently registers a time > 0 for one implementation, but doesn't take minutes for the other implementation.

   ```
   for (int i =1;i<= 100000;i++)
        list1.add(i,new Integer(r.nextInt()));
   ```

   I would like you to evaluate the following operations:
   - Add at beginning of list
   - Add at end of list
   - Remove from beginning of list
   - Remove from end of list
   - List traversal

   The output produced by the driver should indicate the result of each evaluation.

   You'll find notes on reference based lists at the end of this document.

2. In class this week, we will develop a `Stack` class and a `Queue` class. Use these classes to implement an HTML balanced tag checker. Your assignment is to write a program that prompts the user for an .html file name, opens the file and scans it to ensure that each start tag has a matching end tag.

This is a simple html file with properly matched tags. The opening tag and closing tag match by name, with the closing tag beginning with a /.

```
<html>
<body>
    This is my first web page
</body>
</html>
```

- It is acceptable for tags to be nested.
- To make the assignment more manageable, we will assume that each tag must be opened and closed (true in XHTML but not HTML).
- Tag matches should be case insensitive. It is ok to convert all tags to lower (or upper) case.
- You should prompt the user for a filename. If that file does not exist, you may exit gracefully, you do not need to ask again

The objectives of this assignment do not include practice with file parsing. To that end, you may make the following (some realistic, some unrealistic) assumptions:

- tags can not cross multiple lines
- tags may contain no white space
- tags may have no attributes, they are one, single word (or character)

Plan of Attack

Create a `Tag` class capable of holding the information of one tag. To make life easier, you'll want a `Tag` to be able to identify its base tag (ie, `b, body, html`) as well as whether it's an opening or closing tag.

The algorithm:

```
Load all tags from the file into a queue
Create a stack
for each tag in the queue
   if it is an opening tag
        push it onto the stack
   if it is a closing tag,
        if stack is empty
             MISMATCH
        otherwise
             pop the top element from the stack - and you should have a
             match
             if match
                  keep going
             otherwise
                  MISMATCH
```

If you find a mismatch, report the error and terminate the program.

If your program was run on this file:

```
<body>
<html>
     <B>This is an html file</B>
</body>
```

The output would be:ï

```
</b> matches <b>
Found </body> to match <html> terminating program
```

If your program was run on this file:

```
<HTML>
<HEAD>
<TITLE>My Web Page</TITLE>
</HEAD>
<BODY>
<P>This is where you will enter all the text and images you want displayed
in a browser window.</P>
</BODY>
</HTML>Ï
```

The output would be:ï

```
</title> matches <title>
</head> matches <head>
</p> matches <p>
</body> matches <body>
</html> matches <html>
All tags in file matched
```

What to submit

Program #1:

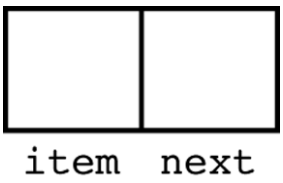
```
ListReferenceBased.java
ListComparer.java (the "main" program)
```

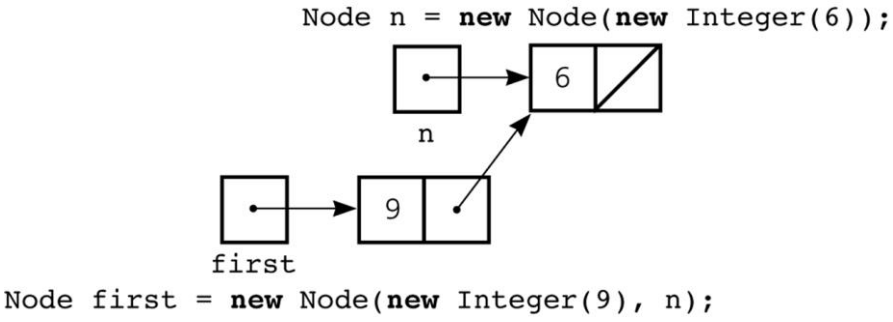
Program #2:

```
Tag.java
StackException.java
StackInterface.java
StackReferenceBased.java
QueueException.java
QueueInterface.java
QueueReferenceBased.java
TagChecker.java (the "main" program)
```

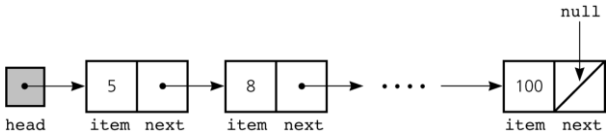
Reference Based Linked Lists

Linked list
- Contains nodes that are linked to one another
- A node
  - Contains both data and a "link" to the next item
  - Can be implemented as an object



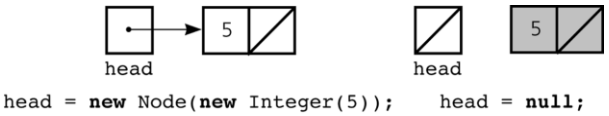


Data field `next` in the last node is set to `null`
- `head` reference variable
  - References the list's first node
  - Always exists even when the list is empty



•`head` reference variable can be assigned `null` without first using `new`
–Following sequence results in a lost node

```
head = new Node(); // Don't really need to use new here
head = null; // since we lose the new Node object here
```



Programming with Linked Lists,
Displaying the Contents of a Linked List

`curr` reference variable
- References the current node
- Initially references the first node

To display the data portion of the current node

```
System.out.println(curr.getItem());
```

To advance the current position to the next node

```
curr = curr.getNext();
```

To display all the data items in a linked list

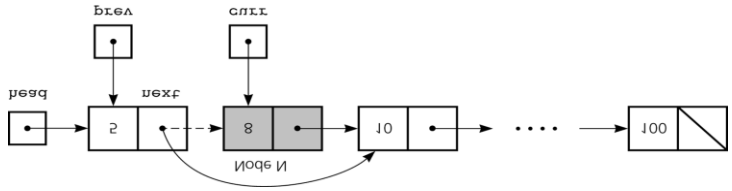
```
for (Node curr = head; curr != null; curr = curr.getNext()) {
     System.out.println(curr.getItem());
} // end for
```

Deleting a Specified Node from a Linked List

To delete node N which `curr` references

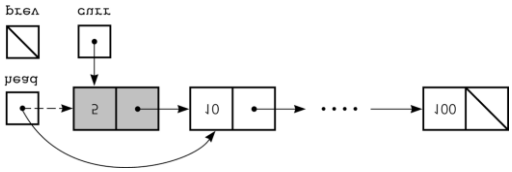
–Set `next` in the node that precedes N to reference the node that follows N

```
prev.setNext(curr.getNext());
```



Deleting the first node is a special case

```
head = head.getNext();
```



Deleting a Specified Node from a Linked List

To return a node that is no longer needed to the system

```
curr.setNext(null);
curr = null;
```

Three steps to delete a node from a linked list

–Locate the node that you want to delete

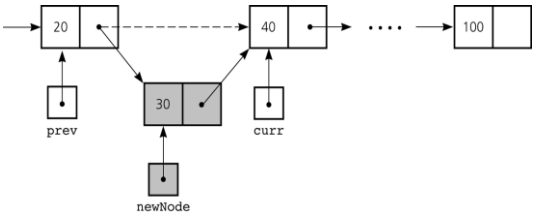
–Disconnect this node from the linked list by changing references

–Return the node to the system

Inserting a Node into a Specified Position of a Linked List

- To create a node for the new item

```
newNode = new Node(item);
```

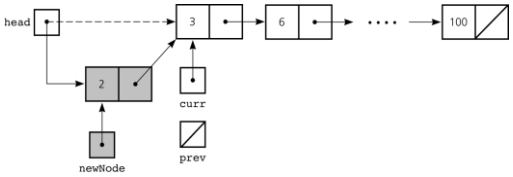


- To insert a node between two nodes

```
newNode.setNext(curr);
prev.setNext(newNode);
```

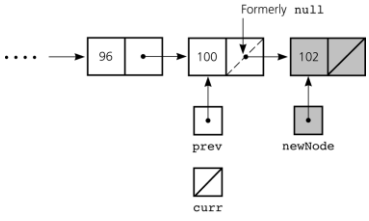
To insert a node at the beginning of a linked list

```
newNode.setNext(head);
head = newNode;
```



Inserting at the end of a linked list is not a special case if `curr` is `null`

```
newNode.setNext(curr);
prev.setNext(newNode);
```



Three steps to insert a new node into a linked list
–Determine the point of insertion
–Create a new node and store the new data in it
–Connect the new node to the linked list by changing references

Passing a Linked List to a Method

- A method with access to a linked list's `head` reference has access to the entire list
- When head is an actual argument to a method, its value is copied into the corresponding formal parameter