COMP 2150 – Fall 2021
Homework 8: LinkedList,DoublyLinkedlist,
(61 points)

Number of People: Individual. Feel free to ask me for help, or visit the Computer Science Learning Center (http://www.memphis.edu/cs/current_students/cslc.php).

Due: Sun., Nov 21, 2021 by 11:59 pm

Submission: Zip all your Java source files (you can zip the entire project folder if using an IDE) into a single file and upload it to the proper folder in the eLearn dropbox at https://elearn.memphis.edu. **Do not include Password protection on you zipped file. You have to include this question paper with answers to the submitted ZIP file**

Coding Style: Use consistent indentation. Use standard Java naming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**. Include a reasonable amount of comments.

Grader:

thpmagar@memphis.edu Abisha Thapa Magar

This assignment deals with writing programs in the areas of linkedList (working with Nodes) and performance in DoublyLinkedList.

```
// The following ListNode class is for Question1, and 2)
//ListNode is a class for storing a single node of a linked list storing
//integer values.  It has two public data fields for the data and the link to the next node in the list
// It has three overloaded constructors

public class ListNode {
 public int data;
 public ListNode next;
 // post: constructs a node with data 0 and null link
 public ListNode() {
     this(0, null);
 }
 // post: constructs a node with given data and null link
 public ListNode(int data) {
     this(data, null);
 }
 // post: constructs a node with given data and given link
 public ListNode(int data, ListNode next) {
     this.data = data;
     this.next = next;
 }
}
```

(For question 1, you will have to Cut&Paste the "text-graphics" from the question to generate answers and change the node contents number (integer))

1. (9 points) What would the given linked node diagram look like after the given code executes:

   a. (3 points)

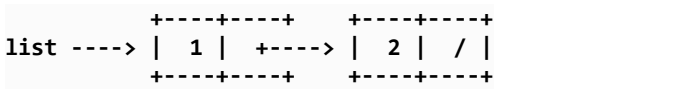
```
           +----+----+    +----+----+
list ----> |  1 |  +----> |  2 |  / |
           +----+----+    +----+----+
```

```
List.next = new ListNode(9);
```

Example: (this one is free as an example)

```
           +----+----+    +----+----+
list ----> |  1 |  +----> |  9 |  / |
           +----+----+    +----+----+
```

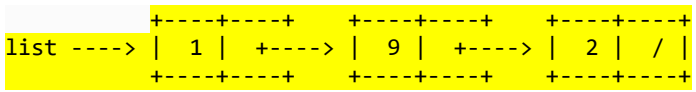
b. (3 points)



```
          +----+----+    +----+----+
list ---> |  1 |  +---> |  2 |  / |
          +----+----+    +----+----+
```

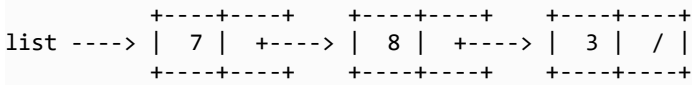
```
List.next = new ListNode(9, list.next);
```

(redraw the above diagram for the code)



```
          +----+----+    +----+----+    +----+----+
list ---> |  1 |  +---> |  9 |  +---> |  2 |  / |
          +----+----+    +----+----+    +----+----+
```

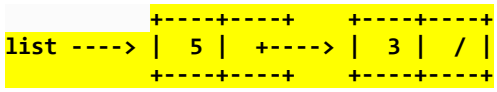
c. (3 points)



```
          +----+----+    +----+----+    +----+----+
list ---> |  7 |  +---> |  8 |  +---> |  3 |  / |
          +----+----+    +----+----+    +----+----+
```

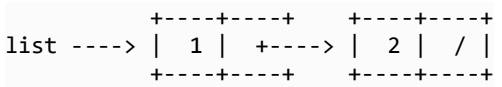
```
List = new ListNode (5, list.next.next);
```

(redraw the above diagram for the code)



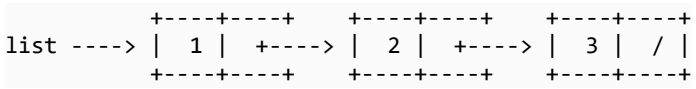
```
          +----+----+    +----+----+
list ---> |  5 |  +---> |  3 |  / |
          +----+----+    +----+----+
```

2. (22 points) Write the code that will convert the given **Before links** to the **After links** with comments.
For example:
Before:



```
          +----+----+    +----+----+
list ---> |  1 |  +---> |  2 |  / |
          +----+----+    +----+----+
```

After:

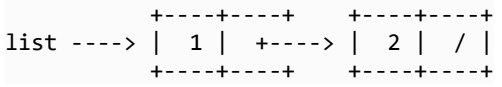


```
          +----+----+    +----+----+    +----+----+
list ---> |  1 |  +---> |  2 |  +---> |  3 |  / |
          +----+----+    +----+----+    +----+----+
```

Code: (example) you must include comments to your code, otherwise your will only get 1/2 for the total marks)
Example: the following example code illustrates the "after" links of the above diagram

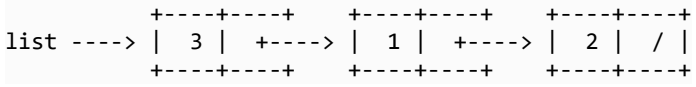
```
list.next.next = new ListNode(3, null);   // 2 -> 3
```

a) (3 points)
Before:



```
          +----+----+    +----+----+
list ---> |  1 |  +---> |  2 |  / |
          +----+----+    +----+----+
```

After:



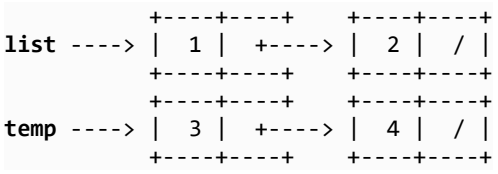
```
          +----+----+    +----+----+    +----+----+
list ---> |  3 |  +---> |  1 |  +---> |  2 |  / |
          +----+----+    +----+----+    +----+----+
```

Code: (3 points)

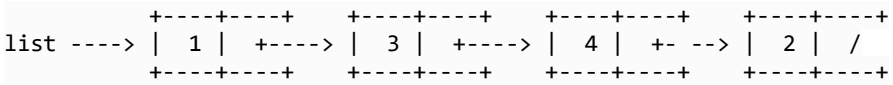
```
List = new ListNode(3, list);   // 3 -> 1 -> 2
```

b) (3 points)

Before:



After:



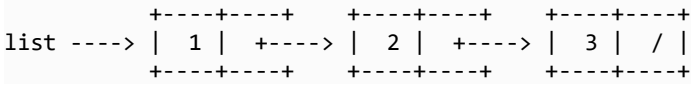
Code: (3 points)

temp.next.next = list.next; // 3 -> 4 -> 2

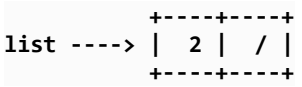
list.next = temp; // 1 -> 3 -> 4 -> 2

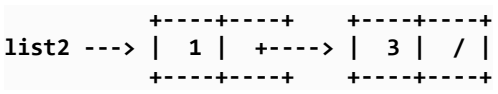
c) (5 Points)

Before



After:





Code: (5 points)

list2 = list; // 1 -> 2 -> 3

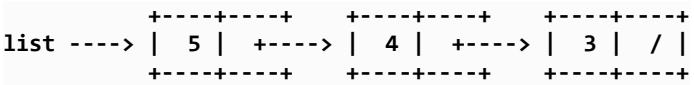
list = list.next; // 2 -> 3

list2.next = list2.next.next; // 1 -> 3

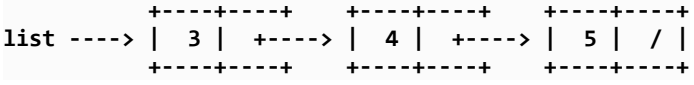
list.next = null; // 2

d) **(5 points)**

Before



After:



Code: (5 points)

list.next.next.next=list; // 3 -> 5 ->4

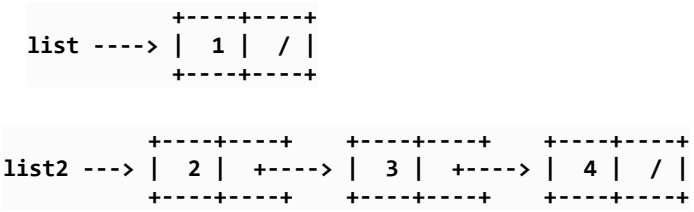
list = list.next.next; // 3 -> 5 -> 4 and back to 3 and it loops again

list.next.next.next = list.next; // 5 -> 4-> 3 -> 5

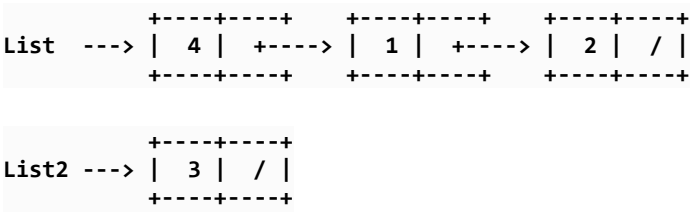
list.next.next.next = list.next; // 5 -> 4 -> 3 -> 4 -> 5 -> it continues
list.next = list.next.next; // 3 -> 4 ->5
list.next.next = null; // it disconnects the value from 5 -> 4 so it gives 3 -> 4-> 5

e) (6 points)

Before:



After:



Code: (6 points)

list2.next.next.next = list; // 2 -> 3 -> 4 -> 1
list.next = list2; // 1 -> 2 -> 3 -> 4
list = list2.next.next; // 3 -> 4
list2 =list2.next; // 2 -> 3
list.next.next.next = null; // 4-> 1 -> 2
list2.next = null; // 3

3. (30 points) For this question, you are converting the given `MyLinkedList.java` (written for singly linkedList) to a doubly-linked list type. The MyLinkedList.java should run with the given TestMyLinkedList.java generating the following result. The test program reports the duration (msecs) for the linkedList type object to exercise through multiple data movement patterns (adding (first, last), getting/setting (first, last), removing (first, last), checking the contents and the occurrence number). No worries, you do not have to write any of the above data movement codes. From the given MyLinkedList.java program, you modify the code for the single "one way" (.next) operation (from head to tail) to the doubly-linked list approach using the ".previous" to go backward. There are 10 locations (hints) marked with (for example, item 8)

   (// ***** (item 8) add codes for the doublyLink (item 8)

   I would keep the programs (MyLinkedList.java and TestMyLinkedList.java) in a separate package and create a new package for your MyDoublyLinkedList.java and the TestMyDoublyLinkedList.java (slightly modified from TestMyLinkedList.java) for the new MyDoublyLinkedList type). That way, you can test run and compare the performance of the two types separately.

   BTW, there is a method getNodeAt(in index) (// commented out in MyLinkedList.java) that you can use it for the previous node variable in the static Node<E> class at the end of the program.

Before making any changes to your codes, you need to run the test program (TestMyDoublyLinkedList.java) to verify the results. For the new program(MyDoublyLinedList.java), I would "refactor" the MyLinkedList to DoublyLinkedList, and modify the MylinkedList type in the (new) TestDoublyLinkedList.java to DoublyLinkedList.

Notice the DURATION (msec) at the end of the display. Your doubly linked list codes should display the same result except for the DURATION. You should see improvement by about >2x time faster. It depends on the kind of computer that you are using. It will help if you run and test the codes continuously after modifying each item. Be sure to keep the original programs in a separate package and develop your MyDoublyLinkedList.java program separately to illuminate confusion.

```
Initialized with {A,B,A,A}:     [A, B, A, A]

Adding elements:                [A, B, A, A]
  - '*' at first                [*, A, B, A, A]
  - 'C' at last                 [*, A, B, A, A, C]
  - 'D'                         [*, A, B, A, A, C, D]
  - '#' @ 2                     [*, A, #, B, A, A, C, D]

Getting elements:               [*, A, #, B, A, A, C, D]
  - First Element               *
  - Last Element                D
  - Element at 1                A
  - Element at 20               null

Setting elements:               [*, A, #, B, A, A, C, D]
  - Element @ 0: '*' -> '+'     [+, A, #, B, A, A, C, D]
  - Element @ 2: '#' -> '-'     [+, A, -, B, A, A, C, D]
  - Element @ 9: 'null' -> '&'  [+, A, -, B, A, A, C, D]

Removing elements:              [+, A, -, B, A, A, C, D]
  - First element '+'           [A, -, B, A, A, C, D]
  - Last element 'D'            [A, -, B, A, A, C]
  - Element @ 1 '-'             [A, B, A, A, C]
  - Element @ 9 'null'          [A, B, A, A, C]

Checking:                       [A, B, A, A, C]
  - Contains 'A'?               true
  - Contains 'Z'?               false
  - First occurence of 'A' @    0
  - Last occurence of 'A' @     3
  - First index of 'Z' @        -1
  - Last index of 'Z' @         -1

 DURATION = 39.5768 msecs
----------------------
```

After converting the give MyLinkedList to MyDoublyLinkedList, running the program, everything should remain the same except:

```
DURATION = 12.5354 msecs (on my laptop)
```