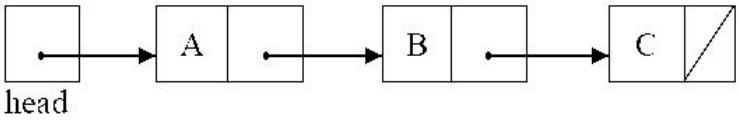
Tutorial 07 Linked list, Stacks, and Queues

Question One:

Write Java statements that create a linked list pictured as follows.



The `Node` class (see also file Node.java) is given as:

```
public class Node {
      private Object item;
      private Node next;
      public Node(Object item) {
            this.item = item;
            next = null; }
      public Node(Object item, Node next) {
            this.item = item;
            this.next = next; }
      public Object getItem() {
            return item; }
      public void setItem(Object item) {
            this.item = item; }
      public Node getNext() {
            return next; }
      public void setNext(Node next) {
            this.next = next; }
      public static void printList(Node list) {
            if (list != null) {
                  System.out.println(list.getItem());
                  printList(list.getNext());   }
      }
}
```

a) Write a method called `createLinkedList1` to create the above linked list beginning with an empty linked list, first create and attach a node for A, then create and attach a node for B, and finally create and attach a node for C.

```
public static Node createLinkedList1() {
      //complete your code in T07Q1.java, please
```

b) Similar to Part a), write a method called `createLinkedList2` to create the above linked list, but instead create and attach nodes in the order C, B, A.

```
public static Node createLinkedList2() {
      //complete your code in T07Q1.java, please
```

Question Two:

Complete `howMany` method to count how many times a data object is contained in linked list.

```
public class ListReferenceBased implements ListInterface {
      private Node head; // head of the linked list
      private int numItems; // number of items in list
      // some other methods …
      public int howMany(Object item) {
            //complete this code here on paper
```

Question Three:

Trace the execution of the following program. Draw the contents of the Stack and the Queue every time there is a change in contents. Also write the output of the program in the end.

```
QueueInterface q = new QueueReferenceBased();
StackInterface s = new StackReferenceBased();
s.push(new Integer(2));
s.push(new Integer(1));
q.enqueue(s.pop());
s.push(new Integer(3));
q.enqueue(new Integer(5));
q.enqueue(new Integer(0));
System.out.print(q.dequeue());
s.push(q.dequeue());
System.out.print(q.dequeue());
System.out.print(s.pop());
```

Question Four:

Please complete the following code to implement a queue using two stacks s1 and s2. You may use **only** the operations: `pop(), push(Object o)` and `isEmpty()`, of the standard stack ADT. (Note that the queued objects are enqueued into one stack and dequeued from the other.)

```
public class TwoStacksQueue {
      private StackReferenceBased s1; // Enqueueing stack
      private StackReferenceBased s2; // Dequeueing stack
      public TwoStacksQueue {
            s1=new StackReferenceBased ( );
            s2=new StackReferenceBased ( );
      }
      public void enQueue(Object o) {
            //complete this code here on paper


      }
      public Object deQueue( ) {
            //complete this code here on paper
```

Answer

Q1.a

```
public static Node createLinkedList1() {
      Node head = new Node ("A");
      Node secondNode = new Node ("B");
      head.setNext(secondNode);
      Node thirdNode = new Node ("C", null);
      secondNode.setNext(thirdNode);
      return head;
}
```

Q1.b

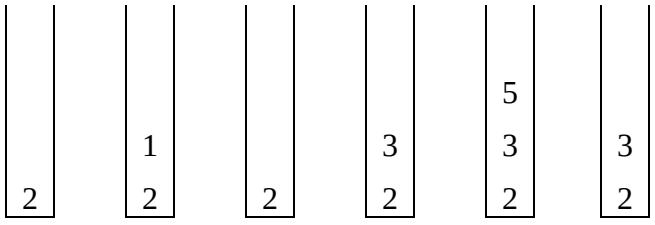
```
public static Node createLinkedList2() {
      Node head = new Node ("C", null);
      Node newNode = new Node ("B", head);
      head = newNode;
      newNode = new Node ("A", head);
      head = newNode;
      return head;
}
```

Q2

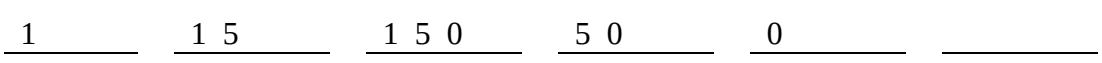
```
public int howMany(Object item) {
      Node curr = head;
      int count = 0;
      while (curr != null) {
            if ( (curr.getItem()).equals(item) )
            count = count+1;
            curr = curr.getNext();
      }
      return count;
}
```

Q3

Stack s:



Queue q:



Output: 105

Q4

```
public void enQueue(Object o) {
s1.push(o);
}
public Object deQueue( ) {
if (!s2.isEmpty()) return s2.pop();
if (s1.isEmpty()) return null;
while(!s1.isEmpty()) s2.push(s1.pop());
return s2.pop();
}
}
```