**Solutions of Homework 3**

**Question 1**

(a) Total sum of all the magnitudes present in the linked list is displayed in the below screen shot. (Please note that the sign of the magnitude is considered in this case).

```java
public static void main(String[] args) { 
    LinkedList list = new LinkedList(); 
    //first value i.e., id is sign, second value i.e., dd is magnitude 
    list.insertFirst(1,3.0); 
    list.insertFirst(1,1.99); 
    list.insertFirst(0,2.2); 
    list.insertFirst(0,83.6); 
    list.insertFirst(1,6.3); 
    list.insertFirst(0,4.0); 
    Link current = list.deleteFirst(); 
    double sum = 0.0; 
    //traverse through the list and calculate sum 
    while(current != null) { 
        //if iData is 1, add dData to the sum 
        if(current.iData == 1) 
            sum += current.dData; 
        //if iData is 0, subtract dData from the sum 
        else 
            sum -= current.dData; 
        //incrementally print out the addition 
        System.out.println("current.iData = " + current.iData); 
        //set current to the next element if it exists 
        if(!list.isEmpty()) 
            current = list.deleteFirst(); 
        else 
            current = null; 
    } 
    System.out.println("Total sum is " + sum); 
}
```

The Main method of the above function is given below:
sum += current.dData;
//if iData is 0, subtract dData from the sum
else
    sum -= current.dData;
//incrementally print out the addition
System.out.println(current.iData + current.dData);
//set current to the next element if it exists
if(!list.isEmpty())
    current = list.deleteFirst();
else
    current = null;
}
System.out.println("Total Sum is " + sum);

(b) The below method is used to insert links at the end of the list.

public void insertLast(int sign, double mag){
    Link current = first;
    Link temp = new Link(sign,mag);
    //check if the list is empty
    if(current == null)
        first = temp;
    //find the last link by traversing through the list
    else{
        while(current.next != null){
            current = current.next;
        }
        current.next = temp;
    }
}

(c) The below method is used to delete the objects in the linked list from the end of the Linked list.

public Link deleteLast(){
    Link previous = first;
    Link temp;

    //if the list is empty return null
    if(first == null)
        return null;
    //if only one element is present
    else if(first.next == null){
        temp = first;
        first = null;
        return temp;
    }
    //Else find the last link
    else{
        while(previous.next.next != null){
            previous = previous.next;
        }
    }
}
temp = previous.next;
previous.next = null;
return temp;
}

Question 2

(a) The methods are included in the class StringQueue.java. Enqueue() method runs in O(n) complexity since the elements are enqueued at the end of the queue, dequeue() method runs in O(1) since the elements are dequeued from the start of the queue, and isEmpty() runs in O(1) complexity.

public class StringQueue {
  QueueItem first;

  public StringQueue(){
    first = null;
  }

  //enqueue a new item item at the end of the linked list queue
  public void enqueue(char c){
    QueueItem temp = new QueueItem(c);
    QueueItem current = first;

    //if the queue is empty, set first to the enqueued item
    if(first == null){
      first = temp;
    }
    else{
      while(current.next != null)
        current = current.next;
      current.next = temp;
    }
  }

  //dequeue the first item of the queue
  public char dequeue(){
    QueueItem temp = first;
    first = first.next;
    return temp.data;
  }

  //return true if queue is empty
  public boolean isEmpty(){
    return (first == null);
  }

  //return true if queue contains the given char
  public boolean contains(char c){

QueueItem current = first;
while(current != null){
    if(current.data == c)
        return true;
    current = current.next;
}
return false;

public class QueueItem{
    public char data;
    public QueueItem next;

    public QueueItem(char c){
        data = c;
    }
}

(b) The below screenshot shows the working of StringQueueDiscard.java using the class StringQueue.java. The screenshot includes the string after the * is dequeued from the string l*LO**VE**EC*E*242*FA**L*L*2013**

public class StringQueueDiscard {

    public static void main(String[] args) {
        StringQueue sq = new StringQueue();
        String enqueueString = "I*LO**VE**EC*E*242*FA**L*L*2013**";

        //enqueue the string and print the result of dequeue;
        System.out.println(stringEnqueue(enqueueString,sq));
    }

    public static String stringEnqueue(String s, StringQueue sq){
        String out = "";
        //enqueue the string character by character
        for(int i=0;i<s.length();i++){
            //if the character is a letter, enqueue it
            if(Character.isAlphabetic(s.charAt(i))){
                sq.enqueue(s.charAt(i));
            }
            //if the character is an *, dequeue
            else if(s.charAt(i) == '*'){
                //make sure the queue isn't empty, takes care of trailing *'s
                if(!sq isEmpty())
                    out+=sq.dequeue();
            }
            //if the character is a number, do nothing
        }
        return out;
    }
}
(c) The below screenshot shows the working of the class StringQueueWithNumbers.java. The screenshot includes the queue which contains letters and queue which contains numbers.

```java
class StringQueueWithNumbers {
    public static void main(String[] args)
    {
        String enqueueString = "I*LO**VE**EC*E*242*FA**F*L*2013**";
        //enqueue the string and print the result after dequeue
        System.out.println(selectiveEnqueue(enqueueString));
    }

    public static String selectiveEnqueue(String s)
    {
        StringQueue stringQ = new StringQueue();
        StringQueue numberQ = new StringQueue();

        String out = "Letters in the queue: ";
        //enqueue the string character by character
        for(int i=0;i<s.length();i++){
            char temp = s.charAt(i);
            //if the character is a letter, enqueue it to the stringQ
            if(Character.isAlphabetic(temp)) {
                stringQ.enqueue(temp);
            } else if(s.charAt(i) == '*') {
                //make sure the queue isn't empty, takes care of trailing '*s'
            }
        }
        return out;
    }
}
```

if the character is an *, dequeue
else if (temp == '*') {
    // make sure the queue isn't empty, takes care of trailing *
    if (!stringQ.isEmpty())
        out += stringQ.dequeue();
}

// if the character is a number, enqueue it to the numberQ
else if (Character.isDigit(temp)) {
    // check if the number exists in numberQ else enqueue it
    if (!numberQ.contains(temp)){
        numberQ.enqueue(temp);
    }
}

out += "\nNumbers in the Queue: ";
// dequeue the items in numberQ till it's empty
while (!numberQ.isEmpty())
    out += numberQ.dequeue();

return out;
Question 3

(a) Stack after inserting 15 and 45.

```
  45
  15
:  --> rest of the stack.
```

The elements in a stack are pushed on top of the stack.

Stack after removing the two elements.

```

:  :
```

Elements from the stack are popped out from top of the stack.

Stack after inserting 25 and 35.

```
  35
  25
:  --> rest of the stack.
```

Stack after removing on item i.e., 35 since the element is popped from top of the stack.

```
  25
:  :
```

25 is left at the top of the stack.

(b) Pushing and popping of elements in an array implementation of stack are of O(1) complexity since when an element is pushed it is added at the end of the array and when popped the last element in the array is removed i.e., a[n-1].

The complexity of enqueue and dequeue for an array implementation of a queue is O(1). When an element is added to a queue it is simply added at the end of the array and when an element is deleted from a queue i.e., after a[n-1], element at the beginning of the array i.e.,a[0] is removed irrespective of the size of the array.
(c) Two references need to be changed namely the head pointer and the next pointer. The head pointer now points to the new node and the next pointer points to the previous first node in the singly linked list.

d) We would need to traverse twice through the singly linked list. The first traversal would be for finding the link i.e., element with the largest key in the list. The second traversal through the list would be to delete the item.