PART A

Question 1

Consider the below linked list created using "Link" class.

Write code segment to change the above linked list to the link list given below (Hint: use "next" attribute of the Link class.)

Lets lake the link pointers as p1,p2,p3,p4

```
class  myList {
public static void main(String[] args){
        link temp = first;
        first = first.next.next;
        temp.next = temp;
        first.next = temp.next;


}

first = fisrt.next.next;
        fisrt.next.next = temp;
```

Question 2

Consider the link class and linked list class given below.

Link

int iData; Link next;
Link(int id) void displayLink()

LinkList

Link first;
void LinkList() boolean isEmpty() void displayList() Link delete(int key) boolean insertAfter(int key, int newData) Link find(int key) void insertFirst(int id)

i) Implement Find(it key) method of the LinkList class to find a link when the iData is passed to the method as key.

```
public Link find(int key){
    Link current = first;

    while(!current==null){
        if(current.iData == key){
            return current;
        }
        else{
            current=current.next;
        }
    }

    return null;
}
```

ii) Implement insertAfter(int key, int newData) method of the LinkList class. InsertAfter() method finds the link with the given key and the new link (with newData value) is inserted immediately after that.

```
public boolean insertAfter(int key, int newData){
    Link current = first;
    Link newLink = new Link(newData);

    while(!current==null){
        if(current.iData == key){
            newLink.next = current.next;
            current.next = newLink;
            return true;
        }
        else{
            current=current.next;
        }
        return false;
    }
}
```

iii) Implement the delete(int key) method of the LinkList class. delete() method finds the link with the given key and remove it from the link list.

```
public Link insertAfter(int key){
    Link current = first;
    Link previous = first;

    while(!current==null){
        if(current.iData == key){
            if(current == first){
                first = current.next;
                return current;
            }
            else{
                previous.next = current.next;
            }
        }
        else{
            previous = current;
            current=current.next;

        }
        return null;
    }
}
```

iv) Write an application to enter numbers from the keyboard to a link list.

1) Add a new link after a given number and display the list.
2) Delete a link from the link list and display the list

```
import java.util.Scanner;

class linkApp {
    public static void main(String [] args) {
```

```
        Linklist theList = new Linklist();
        Scanner sc = new Scanner(System.in);
// since the number of elements are not given we can take any value
        for (int i = 0; i < 10; i++) {
            theList.insertAfter(sc.nextInt());
        }
// 50 and 55 are hypothetical values
        if (theList.insertAfter(50, 55)) {
            theList.displayList();
        }
        if(theList.delete(50)!= null)
        {
            theList.displayList();
        }
    }
}
```

PART B

Additional Exercises:

Question 1

A double-ended list has an additional reference to the last link. Implement insertFirst(), deleteFirst() and insertLast() methods of a double-ended list. (Assume that the double-ended list is used to store integer numbers)

Double-Ended Lists

A double-ended list is similar to an ordinary linked list, but it has one additional feature: a reference to the last link as well as to the first. Figure 5.9 shows such a list.

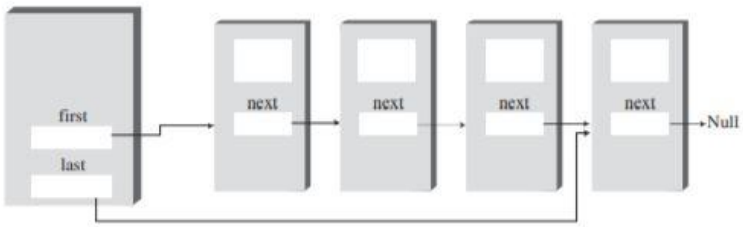

FIGURE 5.9 A double-ended list.

The reference to the last link permits you to insert a new link directly at the end of the list as well as at the beginning. Of course, you can insert a new link at the end of an ordinary single-ended list by iterating through the entire list until you reach the end, but this approach is inefficient.

Access to the end of the list as well as the beginning makes the double-ended list suitable for certain situations that a single-ended list can't handle efficiently. One such situation is implementing a queue; we'll see how this technique works in the next section.

```
class Link {
    public long dData; // data item
    public Link next; // next link in list
    // -------------------------------------------------------------

    public Link(long d) // constructor
    {
        dData = d;
    }

    // -------------------------------------------------------------
public void displayLink() // display this link
{ System.out.print(dData + " "); }

    // -------------------------------------------------------------
} // end class Link
  ////////////////////////////////////////////////////////////////
```

```
class FirstLastList {
    private Link first; // ref to first link
    private Link last; // ref to last link
    // -----------------------------------------------------------

    public FirstLastList() // constructor
    {
        first = null; // no links on list yet
        last = null;
    }

    // -----------------------------------------------------------
    public boolean isEmpty() // true if no links
    {
        return first == null;
    }

    // -----------------------------------------------------------
    public void insertFirst(long dd) // insert at front of list
    {
        Link newLink = new Link(dd); // make new link
        if (isEmpty()) // if empty list,
            last = newLink; // newLink <-- last
        newLink.next = first; // newLink --> old first
        first = newLink; // first --> newLink
    }

    // -----------------------------------------------------------
    public void insertLast(long dd) // insert at end of list
    {
        Link newLink = new Link(dd); // make new link
        if (isEmpty()) // if empty list,
            first = newLink; // first --> newLink
        else
            last.next = newLink; // old last --> newLink
        last = newLink; // newLink <-- last
    }
```

```
    // -------------------------------------------------------------
    public long deleteFirst() // delete first link
    { // (assumes non-empty list)
        long temp = first.dData;
        if (first.next == null) // if only one item

            last = null; // null <-- last
        first = first.next; // first --> old next
        return temp;
    }

    // -------------------------------------------------------------
public void displayList()
{
System.out.print("List (first-->last): ");
Link current = first; // start at beginning
while(current != null) // until end of list,
{
current.displayLink(); // print data
current = current.next; // move to next link
}
System.out.println("");
}
    // -------------------------------------------------------------
} // end class FirstLastList
```