Sukkur IBA University
Department of Computer Science

DATA STRUCTURES
Lab01 – Arrays, LinkesLists

Instructor: Saif Hassan

READ IT FIRST

Prior to start solving the problems in this assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don't waste time).
2. DEADLINE – 11th March, 2022
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – Submit this docx and pdf file.

KEEP IT WITH YOU!

1. Indent your code inside the classes and functions. It's a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.
4. Read the entire question. Don't jump to the formula directly.

I, _**Amjad Ali**_ with student ID _191-21-0001_

Section _**A**_hereby declare that I do understand the instructions above and follow them. This is my own work.

Exercises

Task1 Description

Double LinkedList

Note: Keep this code with you till the course ends.

Task 01: (Double Linked List)

Understand provided code and implement all required methods (with all possible exceptions) in DoubleLinkedList

Node.java

```java
public class Node {

    String name;
    Node prev, next;

    Node (String name)
    {
        this.prev = null;
        this.next = null;
        this.name = name;
    }
}
```

DoubleLinkedList.java

```java
public class DoubleLinkedList {

    Node head;

    // Add node with name in beginning of linkedlist, name as param
    public void insertAtBeginning(String name)
    {

    }
    // Add node in beginning of linedlist, node as param
    public void insertAtBeginning(Node node)
    {

    }
    // Add node in end of linedlist, name as param
    public void insertAtEnd(String name)
    {

    }
    // Add node in end of linedlist, node as param
    public void insertAtEnd(Node node)
    {

    }
```

```java
    // Add node after name which is provided as param , name and node as params
    public void insertAfterName(String name, Node node)
    {

    }
    // Add node before name which is provided as param , name and node as params
    public void insertBeforeName(String name, Node node)
    {

    }

    // Make double linkedlist as Circular Double LinkedList
    public void makeCircular()
    {

    }

    // Print all the nodes in linkedlist, make sure it works on circular double linkedl
ist
    public void printAll()
    {

    }
    // Test the class
    public static void main(String[] args) {
        // Test all above methods

    }

}
```

Solution:

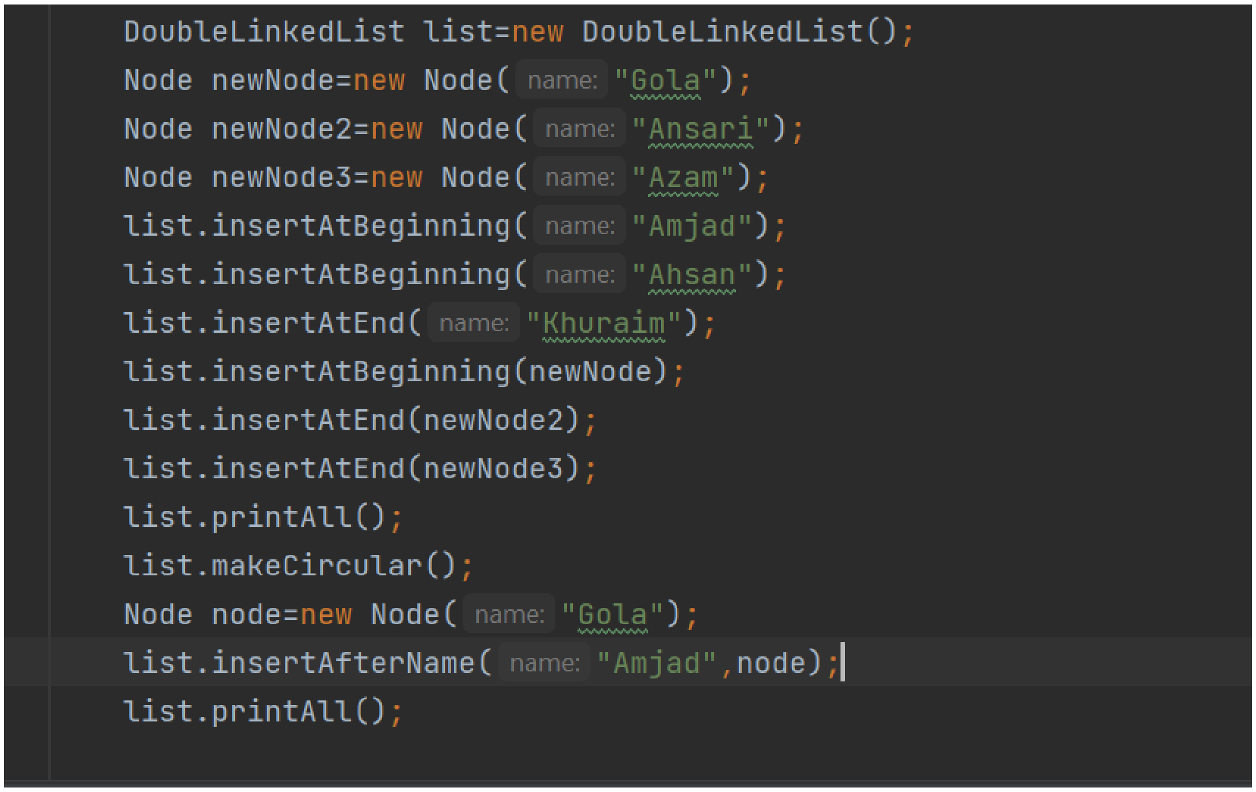
Code

```
public class DoubleLinkedList {

    Node head;

    // Add node with name in beginning of linkedlist, name as param
    public void insertAtBeginning(String name) {

        Node newNode = new Node(name);
        if (head == null) {
             head = newNode;
          } else {
             head.prev = newNode;
             newNode.next = head;
             head = newNode;

          }
      }
      // Add node in beginning of linedlist, node as param

      public void insertAtBeginning(Node node) {

          if (head == null) {
             head = node;
          } else {
             head.prev = node;
             node.next = head;
             head = node;

          }
      }
      // Add node in end of linedlist, name as param

      public void insertAtEnd(String name) {

          Node newNode = new Node(name);
          if (head == null) {
             head = newNode;
          } else {
             Node current=head;
             while(current.next!=null)
             {
                 current=current.next;
             }

```

```
                current.next=newNode;
                newNode.prev=current;
            }

        }
        // Add node in end of linedlist, node as param

        public void insertAtEnd(Node node) {

            if (head == null) {
                head = node;
            } else {
                 Node current=head;
                while(current.next!=null)
                {
                    current=current.next;
                }

                current.next=node;
                node.prev=current;
            }

        }
        // Add node after name which is provided as param , name and node as params

        public void insertAfterName(String name, Node node) {
            boolean a=false;
            Node  current=head;
            while(current!=null && current.next!=head)
            {
                if(current.name==name)
                {
                    a=true;
                    break;
                }
                current=current.next;
            }
            if(a)
            {
                Node temp=current.next;
                current.next=node;
                node.prev=current;
                node.next=temp;
                temp.prev=node;
                System.out.println("Sucessfully Added");
            }
            else{
                System.out.println("Name Doesn't Exist");
            }


        }

        // Add node before name which is provided as param , name and node as params
        public void insertBeforeName(String name, Node node) {
             boolean a=false;
            Node  current=head;
            while(current!=null && current.next!=head)
            {

                if(current.name==name)
                {
                    a=true;
                    break;
                }
                current=current.next;
            }
            if(a)
            {
                Node temp=current.prev;
                temp.next=node;
                node.prev=temp;
                node.next=current;
                current.prev=node;
                System.out.println("Sucessfully Added");
            }
            else{
                System.out.println("Name Doesn't Exist");
            }

        }

        // Make double linkedlist as Circular Double LinkedList
        public void makeCircular() {
           Node current=head;
           while(current.next!=null)
           current=current.next;

           current.next=head;
           head.prev=current;

        }

        // Print all the nodes in linkedlist, make sure it works on circular double linkedlist
        public void printAll() {
```

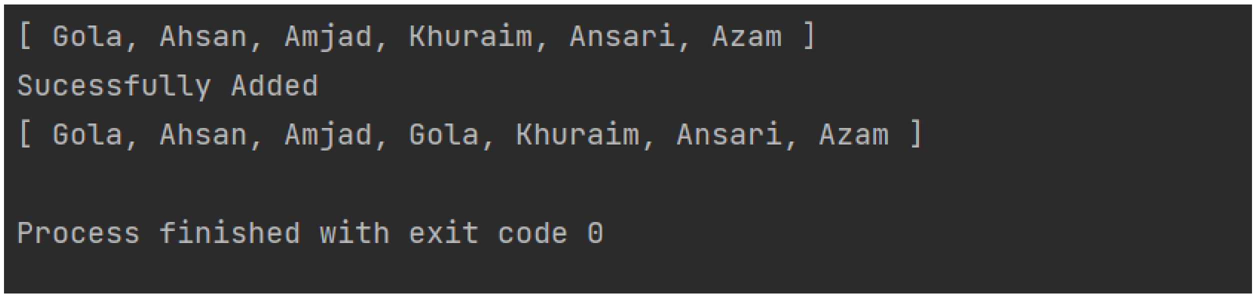
```
        Node current=head;
        StringBuffer str=new StringBuffer();
        str.append("[ ");
        while(current.next!=null && current.next!=head )
        {
            str.append(current.name+", ");
            current=current.next;
        }
        str.append(current.name+" ]");
        System.out.println(str);

    }
    // Test the class

    public static void main(String[] args) {
        // Test all above methods
        DoubleLinkedList list=new DoubleLinkedList();
      Node newNode=new Node("Gola");
      Node newNode2=new Node("Ansari");
      Node newNode3=new Node("Azam");
      list.insertAtBeginning("Amjad");
      list.insertAtBeginning("Ahsan");
      list.insertAtEnd("Khuraim");
      list.insertAtBeginning(newNode);
      list.insertAtEnd(newNode2);
      list.insertAtEnd(newNode3);
      list.printAll();
      list.makeCircular();
      Node node=new Node("Gola");
      list.insertAfterName("Amjad",node);

      list.printAll();



    }

}
```

<u>Sample Input:</u>



```
    DoubleLinkedList list=new DoubleLinkedList();
    Node newNode=new Node( name: "Gola");
    Node newNode2=new Node( name: "Ansari");
    Node newNode3=new Node( name: "Azam");
    list.insertAtBeginning( name: "Amjad");
    list.insertAtBeginning( name: "Ahsan");
    list.insertAtEnd( name: "Khuraim");
    list.insertAtBeginning(newNode);
    list.insertAtEnd(newNode2);
    list.insertAtEnd(newNode3);
    list.printAll();
    list.makeCircular();
    Node node=new Node( name: "Gola");
    list.insertAfterName( name: "Amjad",node);
    list.printAll();
```

<u>Sample Output</u>



```
[ Gola, Ahsan, Amjad, Khuraim, Ansari, Azam ]
Sucessfully Added
[ Gola, Ahsan, Amjad, Gola, Khuraim, Ansari, Azam ]

Process finished with exit code 0
```

Task2 Description

Task02

In previous labs, you have designed single/double linkedlist with all possible common methods with only head.

Now your task is to implement following methods (Single/Double LL) but this time you have to make another variable say **`tail`** for accessing last element directly.

- All types of methods for inserting (Beginning, End)
- All types of methods for removing (Beginning, End)

Compare these methods with those which were designed without **`tail`**.

Solution:

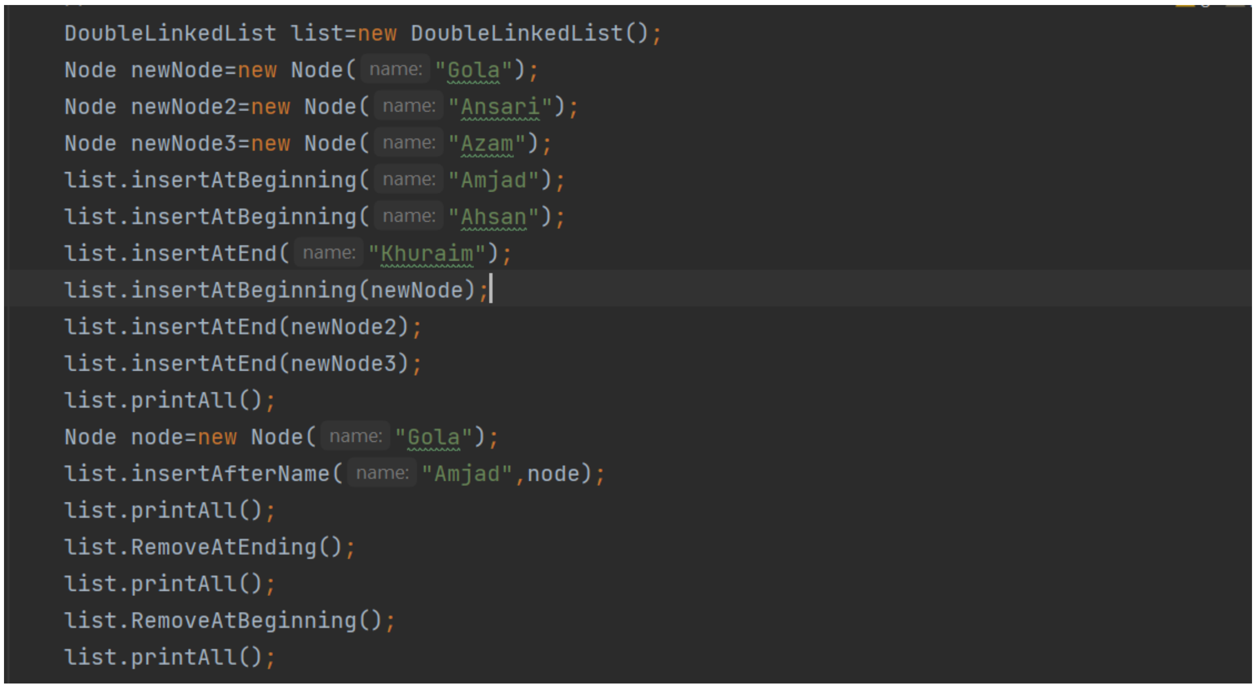
(Part 1 For Double Linked List with Tail.)

Code

```
public class DoubleLinkedList {


    Node head;
    Node tail;

    // Add node with name in beginning of linkedlist, name as param
    public void insertAtBeginning(String name) {

        Node newNode = new Node(name);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            head.prev = newNode;
            newNode.next = head;
            head = newNode;

        }
    }
    // Add node in beginning of linedlist, node as param

    public void insertAtBeginning(Node node) {

        if (head == null) {
            head = node;
            tail = node;
        } else {
            head.prev = node;
            node.next = head;
            head = node;

        }
    }
    public void RemoveAtBeginning()
    {
        if(head==null)
            System.out.println("List is Empty");
        else{
            Node temp=head.next;
            head=temp;
            head.prev=null;
            System.out.println("Succesfully Deleted");
        }
    }

    public void RemoveAtEnding()
    {
        if(tail==null)
            System.out.println("List is Empty");
        else{
            Node temp=tail.prev;
            tail=temp;
            tail.next=null;
            System.out.println("Succesfully Deleted");
        }
    }

    // Add node in end of linedlist, name as param

    public void insertAtEnd(String name) {

        Node newNode = new Node(name);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            newNode.prev = tail;
            tail.next = newNode;
            tail = newNode;
        }

    }
    // Add node in end of linedlist, node as param

    public void insertAtEnd(Node node) {

        if (head == null) {
            head = node;
            tail = node;
        } else {
            node.prev = tail;
            tail.next = node;
```

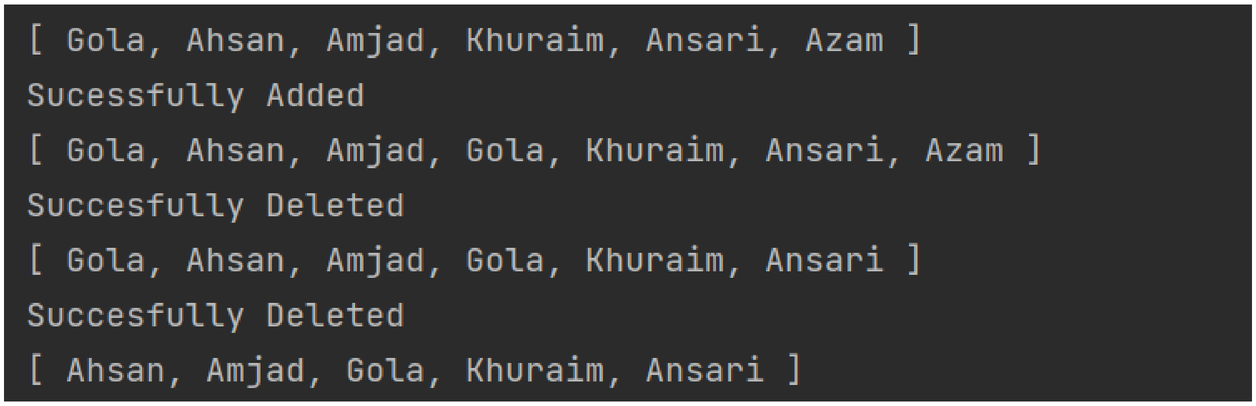
```java
            tail = node;
        }

    }
    // Add node after name which is provided as param , name and node as params

    public void insertAfterName(String name, Node node) {
        boolean a=false;
        Node  current=head;
        while(current!=null && current.next!=head)
        {
            if(current.name==name)
            {
                a=true;
                break;
            }
              current=current.next;
          }
          if(a)
          {
              Node temp=current.next;
              current.next=node;
              node.prev=current;
              node.next=temp;
              temp.prev=node;
              System.out.println("Sucessfully Added");
          }
          else{
              System.out.println("Name Doesn't Exist");
          }


      }

      // Add node before name which is provided as param , name and node as params
      public void insertBeforeName(String name, Node node) {
          boolean a=false;
          Node  current=head;
          while(current!=null && current.next!=head)
          {

              if(current.name==name)
              {
                  a=true;
                  break;
              }
              current=current.next;
          }
          if(a)
          {
              Node temp=current.prev;
              temp.next=node;
              node.prev=temp;
              node.next=current;
              current.prev=node;
              System.out.println("Sucessfully Added");
          }
          else{
              System.out.println("Name Doesn't Exist");
          }

      }

      // Make double linkedlist as Circular Double LinkedList
      public void makeCircular() {
          tail.next=head;
          head.prev=tail;
          System.out.println("Successfully Made Circular");
      }

      // Print all the nodes in linkedlist, make sure it works on circular double linkedlist
      public void printAll() {
          Node current=head;
          StringBuffer str=new StringBuffer();
          str.append("[ ");
          while(current.next!=null && current.next!=head )
          {
              str.append(current.name+", ");
              current=current.next;
          }
          str.append(current.name+" ]");
          System.out.println(str);

      }
      // Test the class

      public static void main(String[] args) {
          // Test all above methods
          DoubleLinkedList list=new DoubleLinkedList();
          Node newNode=new Node("Gola");
          Node newNode2=new Node("Ansari");
          Node newNode3=new Node("Azam");
          list.insertAtBeginning("Amjad");
          list.insertAtBeginning("Ahsan");
          list.insertAtEnd("Khuraim");
          list.insertAtBeginning(newNode);
          list.insertAtEnd(newNode2);
          list.insertAtEnd(newNode3);
          list.printAll();
          Node node=new Node("Gola");
          list.insertAfterName("Amjad",node);
          list.printAll();
          list.RemoveAtEnding();
          list.printAll();
          list.RemoveAtBeginning();
          list.printAll();




      }

}
```

Sample Input:



```
DoubleLinkedList list=new DoubleLinkedList();
Node newNode=new Node( name: "Gola");
Node newNode2=new Node( name: "Ansari");
Node newNode3=new Node( name: "Azam");
list.insertAtBeginning( name: "Amjad");
list.insertAtBeginning( name: "Ahsan");
list.insertAtEnd( name: "Khuraim");
list.insertAtBeginning(newNode);
list.insertAtEnd(newNode2);
list.insertAtEnd(newNode3);
list.printAll();
Node node=new Node( name: "Gola");
list.insertAfterName( name: "Amjad",node);
list.printAll();
list.RemoveAtEnding();
list.printAll();
list.RemoveAtBeginning();
list.printAll();
```

Sample Output



```
[ Gola, Ahsan, Amjad, Khuraim, Ansari, Azam ]
Sucessfully Added
[ Gola, Ahsan, Amjad, Gola, Khuraim, Ansari, Azam ]
Succesfully Deleted
[ Gola, Ahsan, Amjad, Gola, Khuraim, Ansari ]
Succesfully Deleted
[ Ahsan, Amjad, Gola, Khuraim, Ansari ]
```

(Part 2 For Single Linked List with Tail.)

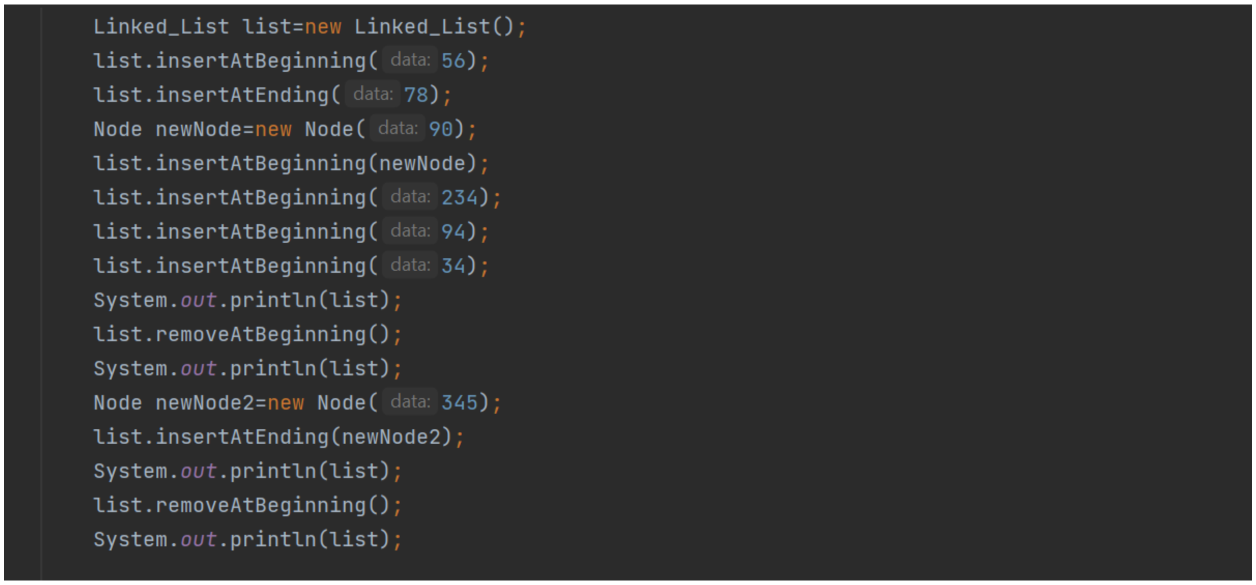
Code:

```
package com.company;
public class Linked_List implements List{

    private int size=0;
    private Node Head;
    private Node Tail;

    public void reverseList(){
        if(Head==null)
            System.out.println("List is Empty");
          else{
            Node current=Head;
            Node previous=null;
            while(current.next!=null)
            {
                Node temp=current.next;
                current.next=previous;
                previous=current;
                current=temp;
            }
            current.next=previous;
            Head=current;
            System.out.print("List Successfully Reversed");


          }

     }
     @Override
     public void incSize(){
         size++;
     }

     @Override
     public void decSize()
     {
         size--;
     }


     @Override
     public boolean isEmpty()
     {
         return Head==null;
     }

     @Override
     public void insertAtBeginning(int data)
     {
         Node newNode=new Node(data);
         if(isEmpty())
         {
            Head=newNode;
            Tail=newNode;
         }
         else{
            Node temp=Head;
            newNode.next=Head;
            Head=newNode;
         }
         incSize();

     }
     @Override
     public void insertAtEnding(int data){
         Node newNode=new Node(data);
         if(Tail==null)
         {
            Head=Tail=newNode;
         }
         else
         {
            Tail.next=newNode;
            Tail=newNode;
         }
         incSize();
     }

     public void insertAtEnding(Node node){

         if(Tail==null)
         {
            Head=Tail=node;
         }
         else
         {
```

```java
                Tail.next=node;
                Tail=node;
            }
            incSize();
        }

        public void insertAtBeginning(Node node){

            if(isEmpty())
            {
                Head=node;
                Tail=node;
            }
            else{
                Node temp=Head;
                node.next=Head;
                Head=node;
            }
            incSize();
        }
        public void removeAtBeginning(){
            if(Head==null)
                System.out.println("List is Empty");
            else if(Head.next==null)
            {
                Head=null;
                decSize();
            }
            else{
                Head=Head.next;
                decSize();
            }
        }
        public void removeAtEnding(){
            if(Head==null)
                System.out.println("List is Empty");
            else if(Head.next==null)
            {
                Head=null;
                decSize();
            }
            else{
                Node current=Head;
                Node previous=Head;
                while(current.next!=null)
                {
                    previous =current;
                    current=current.next;

                }
                previous.next=null;

                decSize();
            }
        }

        @Override
        public int size() {

            return size;
        }

        @Override
        public void add(int index, int item) {

            if(index>size)
                System.out.println("index Out of Bound");
            else{
                int i=1;
                Node current=Head;
                while(i<index)
                {
                    current=current.next;
                    i++;
                }
                current.data=item;
                System.out.println("Successfully Added");
            }
        }

        @Override
        public void removeIndex(int index) {
            if(size<index)
                System.out.println("Index out of bound");
            else if(index==1)
            {
                Head=Head.next;
            }
            else{
                int i=1;
                Node current=Head;
                while(i+1!=index)
                {
                    current=current.next;
                    i++;
                }
                current.next=current.next.next;
                System.out.println("Successfully Removed");
                size--;
            }

        }

        public void remove(int item) {
            if (size == 0) {
                System.out.println("List Is Empty.");
            } else {
                boolean cond = false;
                Node Current = Head;
                Node oneBackCurrent = Head;
                while (Current.next != null) {

                    if (Current.data==item) {
                        cond = true;
                        break;

                    }
                    oneBackCurrent = Current;
                    Current = Current.next;



                }
                if (Current.data==item)
                    cond = true;

                if(Head.data==item)
                {
                    Head=Head.next;
                    size--;
                }
                else if (cond) {
                    oneBackCurrent.next= oneBackCurrent.next.next;
                    System.out.println("Succesfully Removed");
                    size--;
                } else {
                    System.out.println("No Such Element In the List");
                }
            }
        }
        public List duplicateReversed()
        {
            Linked_List list=new Linked_List();
            Node Current=Head;
            for(int i=0;i<size;i++)
            {

                list.insertAtEnding(Current.data);
                Current=Current.next;
            }


            return list;

        }

        public List duplicate()
        {
            Linked_List list=new Linked_List();
            Node Current=Head;
            for(int i=0;i<size;i++)
            {
                Node newNode=new Node(Current.data);

                if(list.isEmpty()) {
                    list.Head = newNode;
                }
                else{
                    Node Check=list.Head;
                    while(Check.next!=null)
                    {
                        Check=Check.next;
                    }
                    Check.next=newNode;


                }
                list.incSize();
                Current=Current.next;

            }

            return list;
        }


        public String toString()
        {
            String Str ="[ Size:("+size+")-->";
            Node Current=Head;
            while(Current.next!=null)
            {
                Str+=Current.data+", ";
                Current=Current.next;
            }
            Str+=Current.data+" ]";

            return Str;
        }

        public static void main(String[] args) {
            Linked_List list=new Linked_List();
            list.insertAtBeginning(56);
            list.insertAtEnding(78);
            Node newNode=new Node(90);
            list.insertAtBeginning(newNode);
            list.insertAtBeginning(234);
            list.insertAtBeginning(94);
            list.insertAtBeginning(34);
            System.out.println(list);
            list.removeAtBeginning();
            System.out.println(list);
            Node newNode2=new Node(345);
            list.insertAtEnding(newNode2);
            System.out.println(list);
            list.removeAtBeginning();
            System.out.println(list);





        }

```

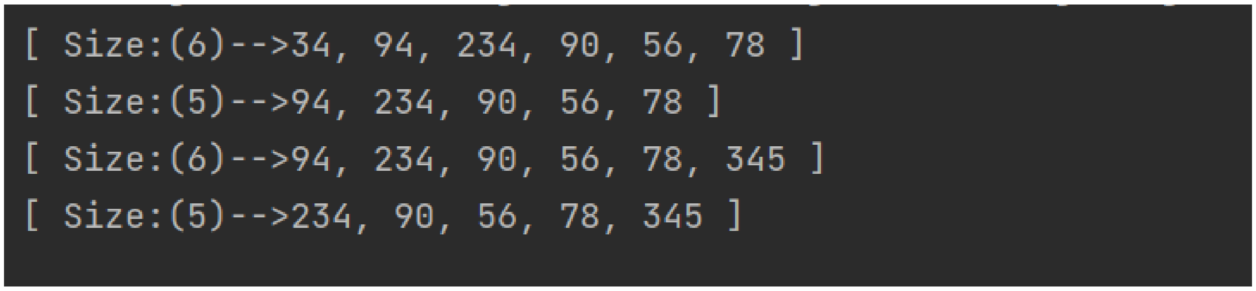
```
}
```

Sample Input:



```
Linked_List list=new Linked_List();
list.insertAtBeginning( data: 56);
list.insertAtEnding( data: 78);
Node newNode=new Node( data: 90);
list.insertAtBeginning(newNode);
list.insertAtBeginning( data: 234);
list.insertAtBeginning( data: 94);
list.insertAtBeginning( data: 34);
System.out.println(list);
list.removeAtBeginning();
System.out.println(list);
Node newNode2=new Node( data: 345);
list.insertAtEnding(newNode2);
System.out.println(list);
list.removeAtBeginning();
System.out.println(list);
```

Sample Output:

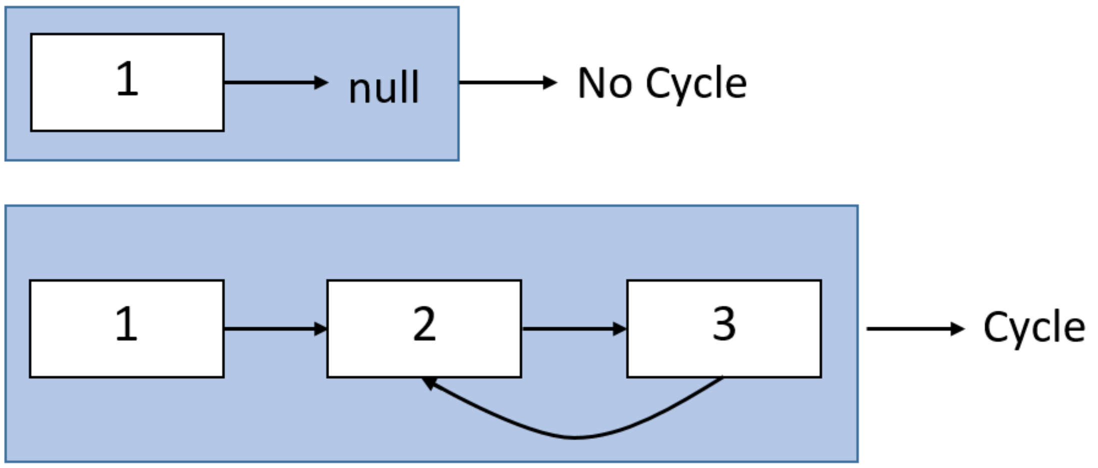


```
[ Size:(6)-->34, 94, 234, 90, 56, 78 ]
[ Size:(5)-->94, 234, 90, 56, 78 ]
[ Size:(6)-->94, 234, 90, 56, 78, 345 ]
[ Size:(5)-->234, 90, 56, 78, 345 ]
```

Task3 Description

Task03

Design a method that takes head as param and detect whether linked list contains cycle or not? Cycle exists in a linked list if any node is visited twice while traversing whole traversing.

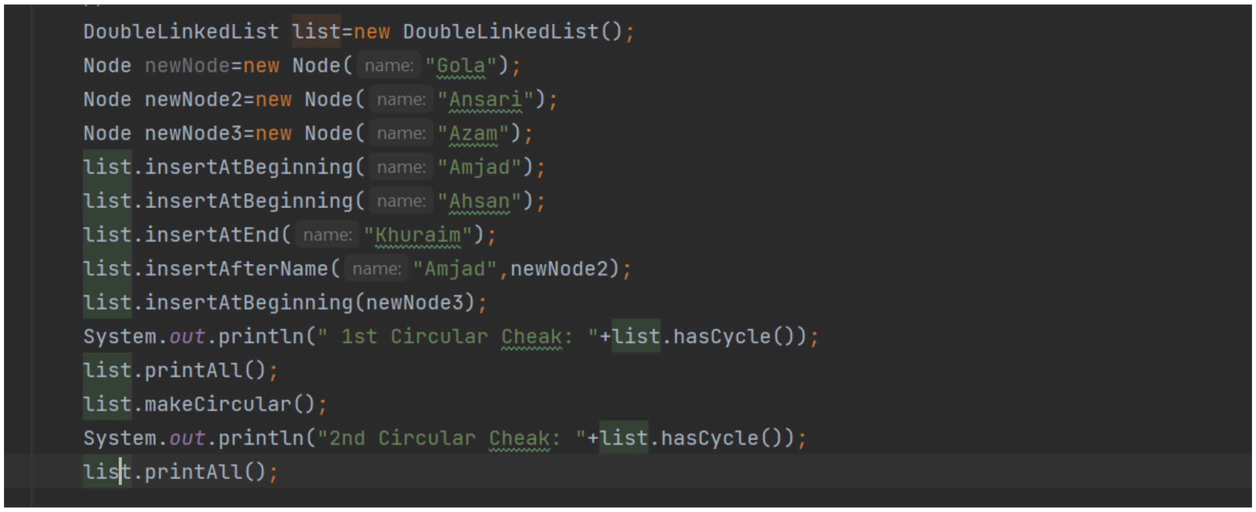


Solution:

Code:

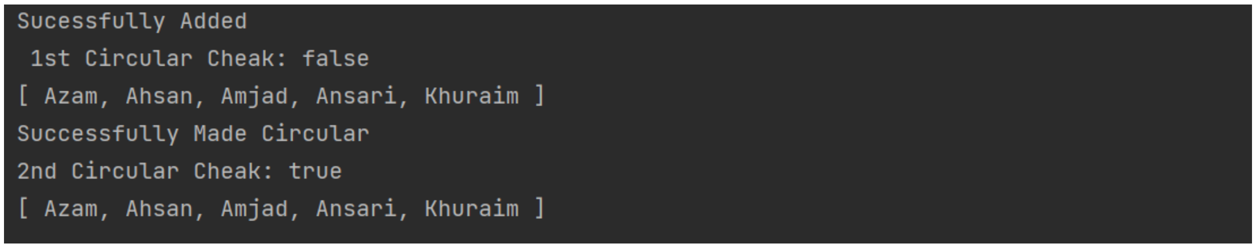
```
public boolean hasCycle(ListNode head) {
        if(head==null)
            return false;


        ListNode current=head;
        ArrayList<ListNode> Array=new ArrayList<>();
        while(current.next!=null)
        {
              if(Array.contains(current))
              {
                  return true;
              }
              else
              {
                  Array.add(current);
              }

              current=current.next;

          }

          return false;

      }
```

<u>Sample Input:</u>



```
        DoubleLinkedList list=new DoubleLinkedList();
        Node newNode=new Node( name: "Gola");
        Node newNode2=new Node( name: "Ansari");
        Node newNode3=new Node( name: "Azam");
        list.insertAtBeginning( name: "Amjad");
        list.insertAtBeginning( name: "Ahsan");
        list.insertAtEnd( name: "Khuraim");
        list.insertAfterName( name: "Amjad",newNode2);
        list.insertAtBeginning(newNode3);
        System.out.println(" 1st Circular Cheak: "+list.hasCycle());
        list.printAll();
        list.makeCircular();
        System.out.println("2nd Circular Cheak: "+list.hasCycle());
        list.printAll();
```

Sample Output



```
Sucessfully Added
 1st Circular Cheak: false
[ Azam, Ahsan, Amjad, Ansari, Khuraim ]
Successfully Made Circular
2nd Circular Cheak: true
[ Azam, Ahsan, Amjad, Ansari, Khuraim ]
```