ArrayList And LinkedList

Ex.1

```
package A1ArrayList;

import java.util.ArrayList;

import java.util.Collections;

public class A1arrylist {


        public static void main(String[] args) {

                ArrayList<Integer> list= new ArrayList<>();

           ArrayList<String> list2 = new ArrayList<String>();

           ArrayList<Boolean> list3 = new ArrayList<Boolean>();


                //add element

                list.add(3);

                list.add(6);

                list.add(5);


                System.out.println(list); //Ans:[3, 6, 5]


                //get element

                int element = list.get(2);//here 2 is index

                System.out.println("Using get method: "+element);//Ans:5


      // add element in between

                list.add(0,8);  //here 0  is index and 8 is an element that want to add

                System.out.println(list);//Ans: [8, 3, 6, 5]
```

```
                    //set element
                    list.set(0, 4);  //here at index 0 we set new element as 4
                    System.out.println(list);//Ans:[4, 3, 6, 5]


                    //delete element
                    list.remove(1);  //here 1 is index
                    System.out.println(list);//Ans:[4, 6, 5]


                    //size
                    int size= list.size();
                    System.out.println("Size :"+size); //Ans:3


                    //loops
                    for(int i=0;i<list.size();i++) {
                              System.out.print(list.get(i)+" ");  //Ans: 4 6 5
                    }
                    System.out.println();


                    //Sorting
                    Collections.sort(list);  //need to import Collections
                    System.out.println(list); //Ans: [4, 5, 6]


          }


}
```

ArrayList And LinkedList

//LinkedList

Ex.2
```
package A2Linklist;
//Different operations on Linklist
public class A1LinklistEx1 {
	Node head;
	private int size;
	A1LinklistEx1(){
		this.size=0;
	}


	   class Node{  //In java Node represented as the form of class
		   String data;
		   Node next;    //here next means the next node hence DataType is Node

		   Node(String data){  //constructor of Node
			   this.data=data;
			   this.next=null;   //next is by default null when we create a
node
			   size++;
		   }
	   }



	    //add- first   here we inserting an element at start of the Linklist
	   public  void addFirst(String data) {
		   Node newNode = new Node(data);
		   if(head==null) {
			   head=newNode;
			   return;
		   }
		   newNode.next=head;
		   head=newNode;
	   }

	   //add-last   //here we add data as a last Node in Linklist
	   public void addLast(String data) {
		   Node newNode = new Node(data);
		   if(head==null) {
			   head=newNode;
			   return;
		   }

		   Node currentNode;
		   currentNode=head;
		   while(currentNode.next!=null) {  //through this we find the last node
			   currentNode=currentNode.next;
		   }
		   currentNode.next=newNode; //at the next of last node we add newNode;

	   }
	//Print   here we print the LinkList
	public void printList() {
		if(head==null) {
			System.out.print("List is empty");
			return;
		}
		Node currNode;
		currNode=head;
		while(currNode!=null) {
			System.out.print(currNode.data+" ");
			currNode=currNode.next;
		}
		System.out.println("null");
	}


	//delete first  here we delete the first node of LinkList
	public void deleteFirst() {
		if(head==null) {
			System.out.println("The List is empty");
			return; //here we right return means we break the function or 
come out of the function deleteFirst
		}
		 size--;
		//My ans
//		Node currntNode;
//		currntNode=head;
//		currntNode.next=head;
		//Apna clg di ans
		head=head.next;//here we mention second node as head so first node 
that was head is deleted so that way we delete first node

	}

	//delete last here we delete the last node of LinkList
	public void deleteLast() {
		if(head==null) {
			System.out.println("The List is Empty");
			return;
		}
		size--;
		if(head.next==null) {  //when we have only one node in linklist
			head=null;
			return;
		}
		Node secondLast=head;
		Node lastNode=head.next;
		while(lastNode.next!=null) {
			lastNode=lastNode.next;
			secondLast=secondLast.next;
		}
		secondLast.next=null; //here we make last node as null
		//Note: For Imagination draw linkList using pen and paper
```

```
          }


          //size  here this function gives size of LinkList
          public int getSize() {
                return size;
          }

       public static void main(String[] args) {
             A1LinklistEx1 ll = new A1LinklistEx1();//here we create object of own
LinkList class
          ll.addFirst("a");
          ll.addFirst("is");
          ll.printList(); //Ans: is a null

          ll.addLast("List");
          ll.printList(); //Ans : is a List null

          ll.addFirst("This");
          ll.printList();//Ans:This is a List null

          ll.deleteFirst();
          ll.printList();//Ans:is a List null

          ll.deleteLast();
          ll.printList();//Ans: is a null

          System.out.println(ll.getSize());//Ans:2
          ll.addFirst("thid");
          System.out.println(ll.getSize());//Ans:3

       }

}
```

Ex.3

```
package A2Linklist;
import java.util.LinkedList; //need to import LinkedList
public class A2LinklistEx2 {

       public static void main(String[] args) {
             LinkedList<String> list = new LinkedList<String>();  //format to create
LinkedList
             list.addFirst("a");
             list.addFirst("is");
             System.out.println(list);//Ans: [is, a]
        list.addLast("list");
        System.out.println(list);//Ans: [is, a, list]
        list.addFirst("this");
        System.out.println(list);//Ans:[this, is, a, list]
        list.add("Saurabh"); //In LinkedList when we write only add then it added to 
the last node
                             //by default add() means addLast() in LinkedList
        System.out.println(list);//Ans: [this, is, a, list, Saurabh]
        list.add(2,"new");
        System.out.println("list after noe added at specific index: "+list);//Ans: 
[this, is, new, a, list, Saurabh]
        System.out.println(list.size());//Ans:5

        //loop
        for(int i=0; i<list.size();i++) {
             System.out.print(list.get(i)+"->");  //Ans:this->is->a->list->Saurabh->
        }
        System.out.println("null");  //Ans:this->is->a->list->Saurabh->null

        //remove
        list.removeFirst();
        System.out.println(list);//Ans: [is, a, list, Saurabh]
        list.removeLast();
        System.out.println(list);//Ans: [is, a, list]
        list.remove(1);//here 1 is index we can delete the node at specific index
        System.out.println(list);//Ans:[is, list]

       }

}
```

Ex.4

```
package A2Linklist;


public class A3ReversedLinkedList {
       Node head;
       private int size;
       A3ReversedLinkedList(){
              this.size=0;
       }


          class Node{  //In java Node represented as the form of class
                 String data;
                 Node next;    //here next means the next node hence DataType is Node

                 Node(String data){  //constructor of Node
                        this.data=data;
                        this.next=null;   //next is by default null when we create a 
node
```

```
                size++;
            }
        }



         //add- first   here we inserting an element at start of the Linklist
        public  void addFirst(String data) {
                Node newNode = new Node(data);
                if(head==null) {
                        head=newNode;
                        return;
                }
                newNode.next=head;
                head=newNode;
        }

        //add-last   //here we add data as a last Node in Linklist
        public void addLast(String data) {
                Node newNode = new Node(data);
                if(head==null) {
                        head=newNode;
                        return;
                }

                Node currentNode;
                currentNode=head;
                while(currentNode.next!=null) {  //through this we find the last node
                        currentNode=currentNode.next;
                }
                currentNode.next=newNode; //at the next of last node we add newNode;

        }


        //Print   here we print the LinkList
        public void printList() {
                if(head==null) {
                        System.out.print("List is empty");
                        return;
                }
                Node currNode;
                currNode=head;
                while(currNode!=null) {
                        System.out.print(currNode.data+"->");
                        currNode=currNode.next;
                }
                System.out.println("null");
        }


        //delete first  here we delete the first node of LinkList
        public void deleteFirst() {
                if(head==null) {
                        System.out.println("The List is empty");
                        return; //here we right return means we break the function or
come out of the function deleteFirst
                    }
                     size--;
                    //My ans
//                  Node currntNode;
//                  currntNode=head;
//                  currntNode.next=head;
                    //Apna clg di ans
                    head=head.next;//here we mention second node as head so first node
that was head is deleted so that way we delete first node

            }

            //delete last here we delete the last node of LinkList
            public void deleteLast() {
                    if(head==null) {
                            System.out.println("The List is Empty");
                            return;
                    }
                    size--;
                    if(head.next==null) {  //when we have only one node in linklist
                            head=null;
                            return;
                    }
                    Node secondLast=head;
                    Node lastNode=head.next;
                    while(lastNode.next!=null) {
                            lastNode=lastNode.next;
                            secondLast=secondLast.next;
                    }
                    secondLast.next=null; //here we make last node as null
                    //Note: For Imagination draw linkList using pen and paper
            }


            //size  here this function gives size of LinkList
            public int getSize() {
                    return size;
            }


            //this method for reverse the LinkedList using space complexity: O(1)
            public void reverseIterate() {  //for better understanding see diagram or
draw diagram using pen and paper
                    if(head==null  || head.next==null) {
                            return;
                    }

                    Node prevNode=head;
                    Node currNode=head.next;
                    while(currNode!=null) {
                                    Node nextNode=currNode.next;
                                    currNode.next=prevNode;
```

```
                         //update
                         prevNode=currNode;
                         currNode=nextNode;

                 }
                 head.next=null;
                 head=prevNode; //here we make last element that is prevNode as a head

             }

             //Reverse a LinkList using Recursive way
             public Node reverseRecursive(Node head) {
                 if(head==null || head.next==null) {
                         return head;
                 }
                Node newHead = reverseRecursive(head.next);
                 head.next.next=head;
                 head.next=null;
                 return newHead;
             }

         public static void main(String[] args) {

                 //In this example we only reverse the linkedList using reverseIterate
method other things we done copy paste from
                 //previous code
                 A3ReversedLinkedList ll=new A3ReversedLinkedList();
             ll.addLast("1");
             ll.addLast("2");
             ll.addLast("3");
             ll.addLast("4");
             ll.addLast("5");
             ll.addLast("6");
             ll.printList(); //Ans:1->2->3->4->5->6->null
            // ll.reverseIterate();
            // ll.printList();//Ans: 6->5->4->3->2->1->null
             ll.head=ll.reverseRecursive(ll.head);
             ll.printList();//Ans: 6->5->4->3->2->1->null
         }

}
```

Ex.5

```
package A2Linklist;
import java.util.LinkedList;
public class A4RemoveNthNodeFromLast {
  //Que. Remove nth node from end of the list  ex. Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
        //Ans Using build-in method of LinkedList
        public static void main(String[] args) {
                LinkedList<Integer> list = new LinkedList<Integer>();
```

```
             list.add(1);
             list.add(2);
             list.add(3);
             list.add(4);
             list.add(5);
             System.out.println("Original list: "+list);//Ans: [1, 2, 3, 4, 5]
             int n=2;
             int size=list.size();
             int nthFromStart= size-n+1;
             list.remove(nthFromStart-1);
             System.out.println("After operation :"+list);//Ans:[1, 2, 3, 5]


      }

}
```

Ex.6

```
package A2Linklist;
//LeetCode question
//Que. Remove nth node from end of the list  ex. Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
 // ex. Input: head = [1], n = 1  Output: []  ex. Input: head = [1,2], n = 1  Output:
[1]

public class A5RemoveNthNodeFromLast {
//Just for understanding copy from leetcode for detail understanding visit question
on leetcode

      /**
       * Definition for singly-linked list.
       * public class ListNode {
       *     int val;
       *     ListNode next;
       *     ListNode() {}
       *     ListNode(int val) { this.val = val; }
       *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
       * }
       */
 /*   class Solution {
          public ListNode removeNthFromEnd(ListNode head, int n) {

              if(head.next==null){
                  return null;
              }
               int size=0;
               ListNode currNode=head;
               while(currNode!=null){
                  currNode=currNode.next;
                  size++;
               }
```

```
                 if(size==n){
                    return head.next;
                 }
                 System.out.print(size);
                 int indexToSearch= size-n;
                  System.out.print(indexToSearch);
                int i=1;
                ListNode prevNode=head;
                while(i<indexToSearch){
                    prevNode=prevNode.next;
                    i++;
                }
                if(prevNode.next!=null){
                   prevNode.next = prevNode.next.next;
                }

                return head;
        }

        }*/
}
```

Ex.7

```
package A2Linklist;

public class A6Palindrom {
  //LeetCode Question
  //Que.Given the head of a singly linked list, return true if it is a palindrome or
false otherwise.
  //Input: head = [1,2,2,1] Output: true


       public static void main(String[] args) {
              // Copied from leetcode for understanding use leetcode for better
understading this question

        /*
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
/*class Solution {
    public boolean isPalindrome(ListNode head) {
        if(head==null || head.next==null){
            return true;  //if linklist is empty or have 1 element then it is
palindrom
        }

        int size=0;
        ListNode forSize=head;
        while(forSize!=null){  //here we find the size of linkedlist
            forSize=forSize.next;
            size++;
        }
        int mid =size/2;
       ListNode prev=head;
       ListNode curr=head.next;
       int k;
       k=1;
       //System.out.print(mid);
       while(k<=mid){
        prev=prev.next;  //we devide second part of linkedlist as new list
        k++;
       }
       curr=prev.next;
       prev.next=null;  //here we make secondlist first as null and then reverse the
second part only
       while(curr!=null){
        ListNode next= curr.next;

        curr.next=prev;
        //update
        prev=curr;
        curr=next;

       }
       ListNode nextHead=prev;  //second list's prev means nextHead
       System.out.print(nextHead.val);
         System.out.print(head.val);
       boolean isPali=true;
       for(int i=0;i<mid;i++){    //here we check values are palindrom or not
         if(head.val==nextHead.val){
            head=head.next;
            nextHead=nextHead.next;
         }else{
            isPali=false;
            break;
          }
       }
       if(isPali){
        return true;
       }else{
           return false;
       }

    }
}
        *
        *
```

```
          * */
       }

}
```

Ex.8

```
package A2Linklist;

public class A7CheckCycle {
  //leetcode question
  //Linked list cycle
       public static void main(String[] args) {

              /**
               * Definition for singly-linked list.
               * class ListNode {
               *     int val;
               *     ListNode next;
               *     ListNode(int x) {
               *         val = x;
               *         next = null;
               *     }
               * }
               */
       /*     public class Solution {
                  public boolean hasCycle(ListNode head) {
                      if(head==null){
                          return false;
                      }
                      ListNode hare=head;  //hare means sassa
                      ListNode turtal=head; //turtal means kasav
                      boolean isCycle=false;
                      while(hare.next !=null && hare.next.next!=null){
                          hare=hare.next.next;
                          turtal=turtal.next;
                          if(hare==turtal){
                             isCycle=true;
                             break;
                          }
                      }
                      if(isCycle){
                          return true;
                      }else{
                          return false;
                      }

                  }
              } */

       }  }
```

ArrayList And LinkedList