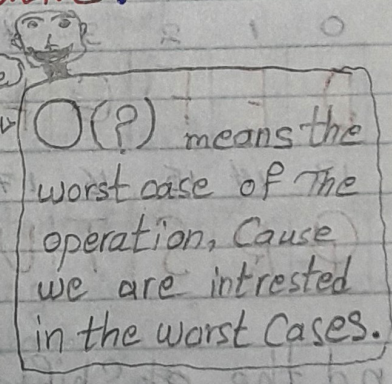
Fundamental Data Structures :-

Array D.S :-

Def: here we know from the Basics The array is some of consecutive cells in the memory, which stores some values of the same Type.

and now we should know The Time Complexity of The array Data Structure for some operations:

1. size is Fixed.
2. Memory must be free Consecutive.
3. Access ⟶ O(1)
4. Delete ⟶ O(n)
5. Insert ⟶ O(n)
6. Search ⟶ O(n)



and we have an application on it called Caesar Cipher is a program makes you encode or decode any massage like:

message : The Eagle Is In Play ; Meet At Joe's.
Secret : WKH HDJOH LV LQ SODB; PHHW DW MRH'V.

of course you didn't understan The Secret message and That what we are want.

This way Based on shifting any character by 3 Characters, and this a Special case of Shift Cipher approach and we represent it in the following figure

Decode:

0	1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	E	F	G	H	I	J	K	L	M

13	14	15	16	17	18	19	20	21	22	23	24	25
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

EnCode:

0	1	2	3	4	5	6	7	8	9	10	11	12
D	E	F	G	H	I	J	K	L	M	N	O	P

13	14	15	16	17	18	19	20	21	22	23	24	25
Q	R	S	T	U	V	W	X	Y	Z	A	B	C

and the ~~ch~~ Shifted Cipher approack is to shift by 0, 1, 2, 3, ..., 25 characters To encode it

we can Defeat this approach by Trying all The 26 possible Shift Ciphers and we will Hack it.

for the next is an implementation for the CaesarCipher Class :

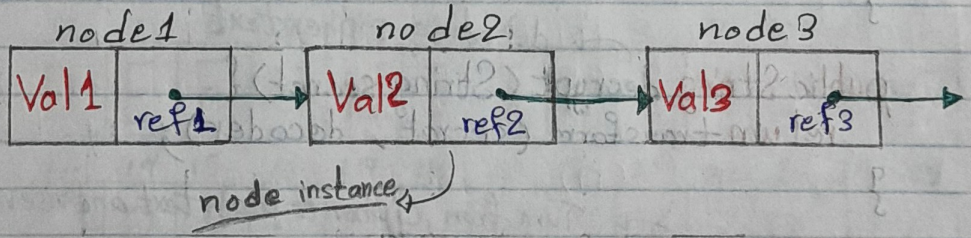
Code:

```
public class CaesarCipher {
  protected char[] encoder = new Char[26];
  protected char[] decoder = new char[26];
                    ↗ in case of caesarCipher rotation = 3
  public CaesarCipher(int rotation) {
    for (int K = 0; K < 26; K++) {
      encoder[K] = (char)('A' + (K + rotation) % 26);
      decoder[K] = (char)('A' + (K - rotation + 26) % 26);
    }
  }
                    ↗ encoding message
  public String encrypt(String message){
    return transform(~~secret~~ message, encoder);
  }
                    ↗ decoding ciphertext
  public String decrypt(String secret) {
    return transform(secret, decoder);
  }
                    ↗ Turn from cipher to plain text and vise verse
  private string transform(String original, char[] code){
    char[] msg = original.toCharArray();
    for (int K = 0; K < msg.Length; K++)
      if (Character.isUpperCase(msg[K])) {
        int j = msg[K] - 'A';
        msg[K] = code[j];
      }
    return new String(msg);
  }
```

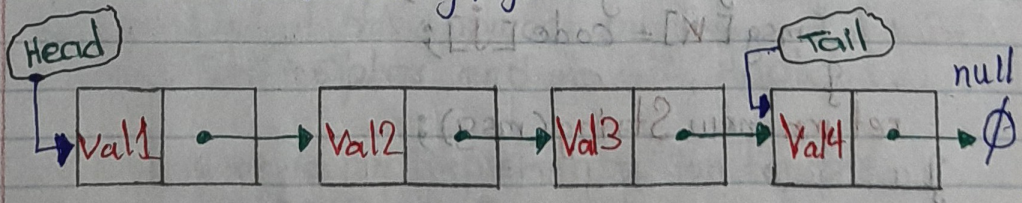
Singly Linked List:

Def: In the array we couldn't store at any size as we want and if we want to delete or insert an element it can be consuming if many elements must be shifted.

we solved These problems by a new data structure Known as Linked list, which provides an Alternative To an array based structure. it's like a Collection of elements called Nodes and we will study The Linear or Singly Linked List. every element in The Linked list has Two Things Value and a reference To the next node like the following figure:



The final node points or referes to Null Object. now every linked list As has To initial pointers or References Tail LastElement and Head firstElement as shown in the following figure:

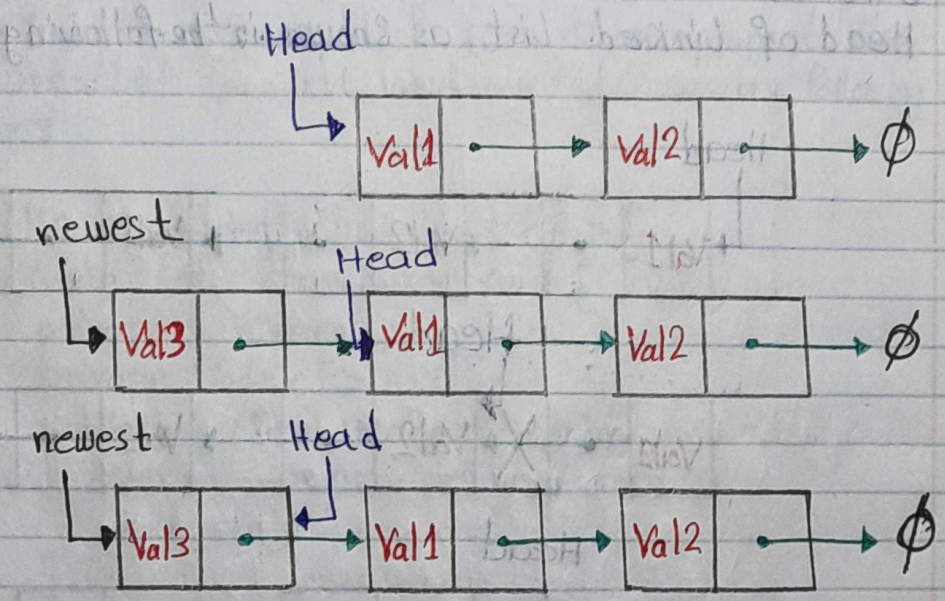


we called The process of Traversing here Link hopping.

Now we want to know how to Insert an element at the Head of a Singly Linked List.

It's garunteed that The Linked List has a dynamic Size.

when we want to add onothere element or Node from The head of the Linked list we want to make it refere To the Previous Head element and make it as a Head by make the head reference referes to the recently add node as following figure:



Algorithm:- addFirst(e) O(1)

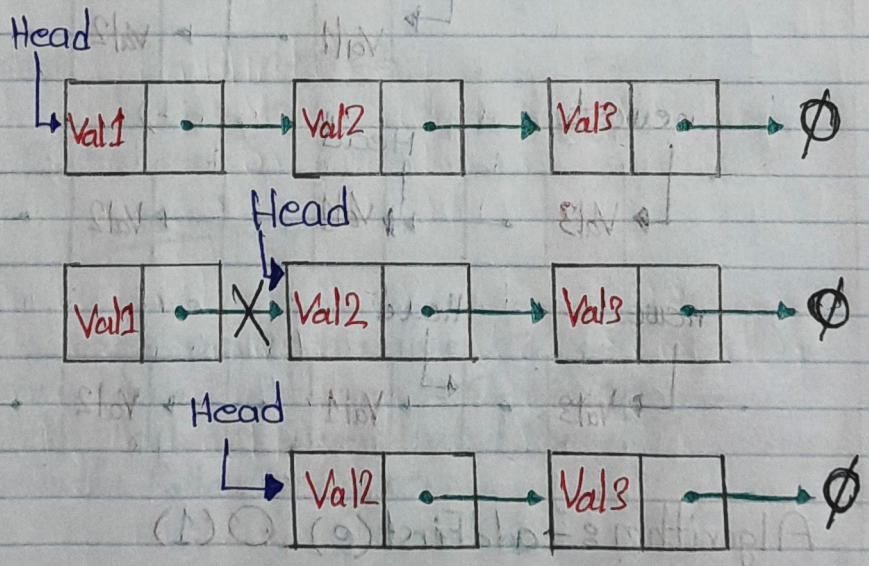
```
newest = Node(e);
newest.next = head;
head = newest;
Size++;
```

The same operation when you Insert from the Tail of linked list.

Algorithm: addLast(e) O(1)

```
newest = Node(e);
newest.next = null;
Tail.next = newest;
tail = newest;
size++;
```

but If we want to remove an element from The Head of linked list as shown in the following figure:



Algorithm: removeFirst(e) O(1)

```
if head == null then the list is empty
head = head.next;
size--;
```

now we should make a implementation for the linkedlist DataStructure and we should implement These methods:

1. size(): Returns the number of elements in the list.
2. isEmpty(): Returns true if the list is empty.
3. ~~Returns~~ first(): Returns the first element in the list.
4. Last(): Returns the last element in the List.
5. addFirst(e): Adds new element to the front of list.
6. addLast(e): Add new element to the end of the List.
7. removeFirst(): Remove and returns the first element list.

we will use two Concepts of java to implement the Singly LinkedList Generics and Nested classes as following Syntax:

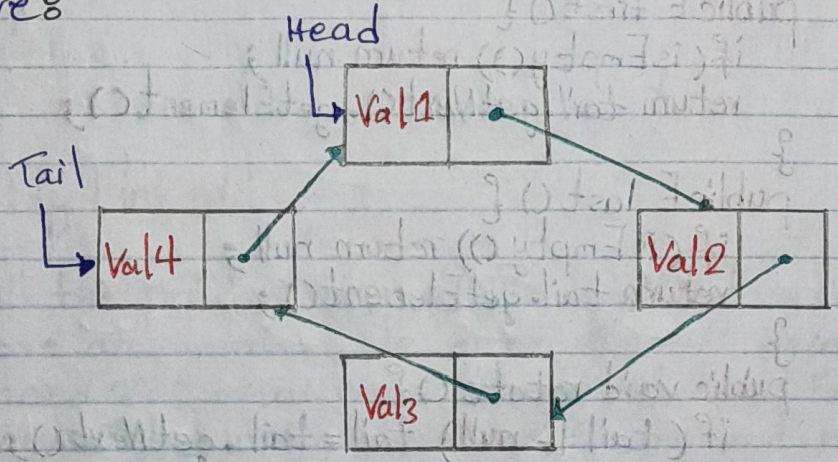
Code:

```
public class SinglyLinkedList<E>{
  private static class Node<E>{
    private E element;
    private Node<E> next;
    public Node(E e, Node<E> n){
      element = e;
      next = n;
    }
    public E getElement(){return element;}
    public Node<E> geNext(){return next;}
    public void setNext(Node<E> n){next = n;}
  }

  //end of The Node class start of The linkedlist.
```

```
private Node<E> head = null;
private Node<E> tail = null;
private int size = 0;
public SinglyLinkedList() {}

public int Size() { return size; }
public boolean isEmpty() { return (size == 0); }
public E first() {
    if (isEmpty()) return null;
    return hea.getElement();
}
public E last() {
    if (isEmpty()) return null;
    return tail.getElement();
}
public void addFirst(E e){
    head = new Node<>(e, head);
    if (size == 0) tail = head;
    size++;
}
public void addLast(E e) {
    Node<E> newest = new Node<>(e, null);
    if (isEmpty()) head = newest;
    else tail.setNext(newest);
    tail = newest;
    size++;}
public E removeFirst(){
    if (isEmpty()) return null;
    E answer = head.getNext();
    size--;
    if(size == 0) tail = null;
    return answer;}
```

Circularly Linked List:

Def: it's a Type of Linked List data structure but designed To be Circular such that there is no tail or head for it implicitly and any can't be refere to a Null value and we called it Circularly Linked List which its tail referes to its head and we can show that in the following figure:



we will update our Singly Linked list by rotate() method which Moves the first element To the end of The List.

and we need to make Additional Optimization That The Head of Linked List, Should be The next of The Tail of The Linked list.

if we want to addFirst we want to make head refere to the newest and if we want ~~Add~~ addLast we want to ~~refere~~ make tail refer to the newest and if we want To remove_First we want to make head refere to head.next and make next of Tail is The new head. now let's implement The Circularly Linked List Datastructure.

Code:

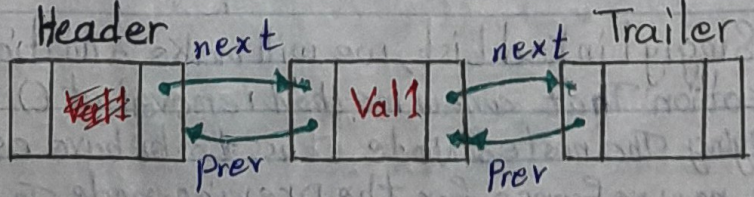
```
public class CircularlyLinkedList <E>{
// (The nested node class is the same as in SinglyLinkedList)
  private Node<E> tail = null;
  private int size = 0;
  public CircularlyLinkedList(){}
  public int size(){ return size; }
  public boolean isEmpty(){ return (size == 0); }
  public E first(){
     if(isEmpty()) return null;
     return tail.getNext().getElement();
  }
  public E last(){
     if(isEmpty()) return null;
     return tail.getElement();
  }
  public void rotate(){
     if(tail != null) tail = tail.getNext();
  }
  public void addFirst(E e){
     if(isEmpty()){
        tail = new Node<>(e, null);
        tail.setNext(tail);
     }else{
        Node<E> newest = new Node<>(e,
            tail.getNext());
        tail.setNext(newest);}
     size++;}
  public void addLast(E e){
     addFirst(e);
     tail = tail.getNext();
  }
```

```
public E removeFirst (){
    if (isEmpty()) return null;
    Node <E> head = tail.getNext ();
    if (head == tail ) tail = null;
    else tail. setNext ( head. getNext ());
    Size--;
    return head. getElement ();
  }
}
```

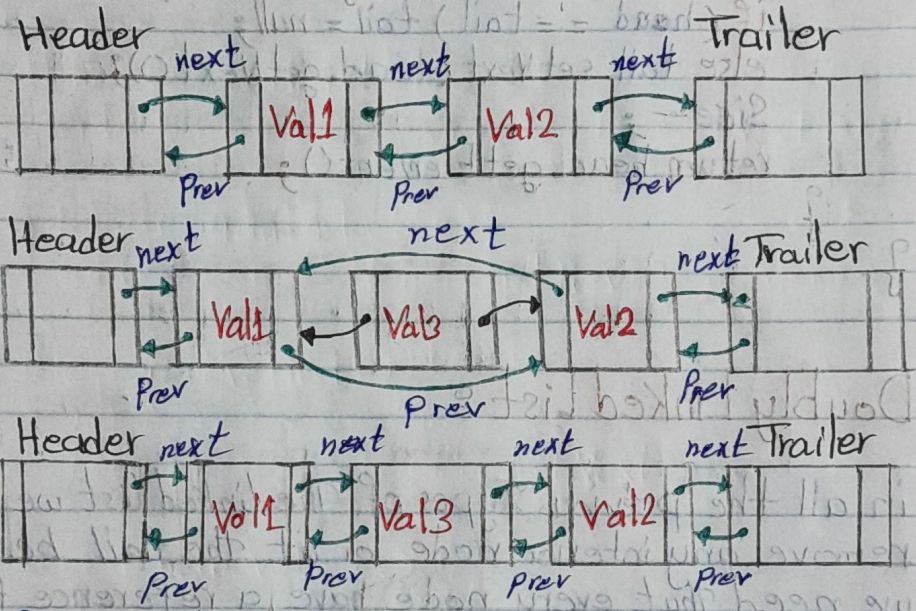
Doubly Linked List :

Def: in all the previous Types of The linked list we couldn't remove any interior node or at the tail because we need That every node have a reference for the previous ~~el~~ node and a reference for the next node. we called that version of linkedList a DoublyLinkedList Datastructure. These Lists allow a greater Variety of $O_{(1)}$ time update operations, including insertions and deletions at arbitrary positions within this list.

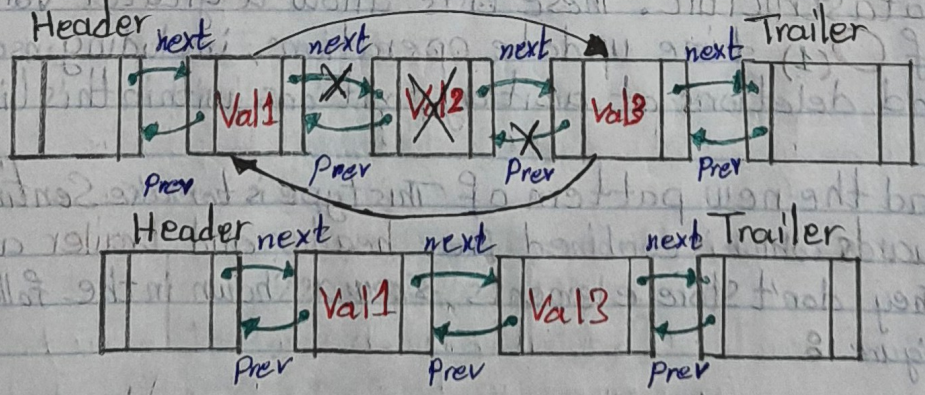
and the new pattern of This type is to make Sentinels or guards which is combined from header and trailer and They don't store elements, ~~show~~ as shown in the following figure :



now let's talk about Insert an element into the DoublyLinked list as shown in the following figure:



and if we want to ~~do~~ Delete an element from the DoublyLinked List as shown in the following figure:



in our SinglyLinkedList, we will make Addational optimization That a new method removeLast() and modifying The nested Node Class to behave as we have a new refrence for the previos node To implement DoubleLinkedList Class.

Code:

```
public class DoublyLinkedList<E> {
  private static class Node<E> {
    private E element;
    private Node<E> prev;
    private Node<E> next;
    public Node(E e, Node<E> p, Node<E> n){
      element = e;
      prev = p;
      next = n;
    }
    public E getElement() { return element; }
    public Node<E> getPrev() { return prev; }
    public Node<E> getNext() { return next; }
    public void setPrev(Node<E> p) { prev = p; }
    public void setNext(Node<E> n) { next = n; }
  }

  private Node<E> Header;
  private Node<E> trailer;
  private int Size = 0;
  public DoublyLinkedList() {
    header = new Node<>(null, null, null);
    trailer = new Node<>(null, header, null);
    header.setNext(trailer);
  }
  public int Size() { return size; }
  public boolean isEmpty() { return (Size == 0); }
  public E first() {
    if (isEmpty()) return null;
    return header.getNext().getElement();
  }
  // Continue the implementation in the next Page.
```

```
public E Last(){
    if (isEmpty()) return null;
    return trailer.getPrev().getElement();
}
public void addFirst(E e) {
    addBetween(e, header, header.getNext());
}
public void addLast(E e){
    addBetween(e, trailer.getPrev(), trailer);
}
public E removeFirst(){
    if (isEmpty()) return null;
    return remove(header.getNext());
}
public E removeLast() {
    if (isEmpty()) return null;
    return remove(trailer.getPrev());
}
// Private update methods.
private void addBetween(E e,
    Node<E>predecessor, Node<E>successor){
    Node<E>newest=new Node<>(e, predecessor,
        Successor);
    predecessor.SetNext(newest);
    Successor.SetPrev(newest);
    Size++; }
private E remove(Node<E> node){
    Node<E> predecessor=node.getPrev();
    Node <E> Successor=node.getNext();
    predecessor.SetNext(Successor);
    Successor.SetPrev(Predecessor);
    Size--; return node.getElement();}}
```

Equivalence Testing:

Testing equality of the data structures ~~is~~ are different first we say that if 2 arrays are equal should provide The ~~following~~ following Points :

1. if The Two arrays are refere t a null value (True). or the same address (True).
2. on of the Two arrays refere to null (false).
3. The Two arrays have different lengths (false).
4. 2 arrays have the same elements and have the same Type (True).
5. for the 2 dimensional arrays all rows are equivalent To the others from the second Array (True).

you can use the static deepEquals(a, b) for the Two dimensional arrays in Arrays Class.

now we will implement equals method for The Linked lists DataStructure. we need To check on The following Points:

1. one of Two Linked Lists is null (false) and don't have The same size (false)
2. both of them aren't the same Class (false).
3. all the nodes in one list are equal to the other list's nodes (True).

here The implementation for the equals method to SinglyLinkedList class:

Code:

```
public boolean equals(Object o){
    if (o == null) return false;
    if (getClass() != o.getClass()) return false;
    SinglyLinkedList other = (SinglyLinkedList) o;
    if( size != other.size) return false;
    Node walkA = head;
    Node walkB = other.head;
    while (walkA != null) {
        if (!walkA.getElement().equals(walkB.
        walkB.getElement())) return false;
        walkA = walkA.getNext();
        walkB = walkB.getNext();
    }
    return true;
}
```

Cloning Data Structures:

Def: Cloning is a concept of The deep Copy for The instance. we can make a copy of any object iff The Class of it implements **Cloneable** interface and use The Clone method of object class or Override it To implement your own process of deep Copy as you want. let me show you how to Clone arrays:

Code:

```
Person[] guests = new Person[contacts.length];
for(int K = 0; K < Contacts.length; K++)
    guests[K] = (Person) contacts[K].clone();
```

and here The Clone process in the 2 dimensional Arrayss

Code:

```
public static int[][] deepClone(int[][]original){
  int[][] backup = new int[original.Length][];
  for(int K=0; K<original.length; K++)
     backup[K] = original[K].clone();
  return backup;
}
```

now let's implement the Clone method for The SinglyLinkedList Class:

Code:

```
public SinglyLinkedList<E> clone() throws
     CloneNotSupportedException {
  SinglyLinkedList<E>other = (SinglyLinkedList)
     super.clone();
  if (size> 0){
     other.head = new Node<>(head.getElement(),null);
     Node<E> walk = head.getNext();
     Node<E>otherTail = other.head;
     while (walk != null){
        Node<E>newest =new Node<>(walk.getElement(),
           null);
        otherTail.setNext(newest);
        otherTail = newest;
        walk = walk.getNext();}}
        return other;}
```