DS (Java)

* Arrays

List of items of the same type and index starts with '0'.

Defining an array

```
type[] arrayName = new type[size];
```

Example

```
int[] marks = new int[3];
 marks[0] = 97
 marks[1] = 98
 marks[2] = 96
 for(int i=0; i<3; i++){
 S.O.P(marks[i]);
 }
```

Position	1	2	3
Index	0	1	2
marks	97	98	96
address	104	108	112

→ fixed size

→ continoue memory

→ primitive and Objects

* 2D-Arrays

Defining an 2D-array

```
type[][] arrayname = new type[rows][Columns];
```

Example

```
int[][] numbers = new int[rows][cols];
//Input
for(int i=0; i< rows; i++){
for(int j=0; j<cols; j++){
 numbers[i][j] = sc.nextInt();
}}
//output
for(int i=0; i<rows; i++){
for(int j=0; j<cols; j++){
 syso(numbers[i][j] +" ");
}
syso();
}
```

	0	1	2	3	4
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)

* String

Defining a String.

```
Datatype StringName = 'Value';

String StringName = Sc.next();
String StringName = Sc.nextLine();
```

Example

```
String fristName = "Devi";
String lastName = "Dolly";
String fullName = fristName + lastName;
S.yso(fullname);
```

* String Builder

Defining a String Builder

```
StringBuilder Variable-name = new StringBuilder("Value");
```

Example

```
StringBuilder Sb = new StringBuilder("Coding");

Syso(Sb.charAt(0));
Syso(Sb.setCharAt(0, 'p');
Syso(Sb.insert(0, 'l');
Syso(Sb.delete (0, 1);
```

* ArrayList

Operations:- Add, Get, modify, Remove, iterate

Size Variable
dynamic non-continoues memory
objects
Heap memory
Insert in O(n)
Search in O(1)

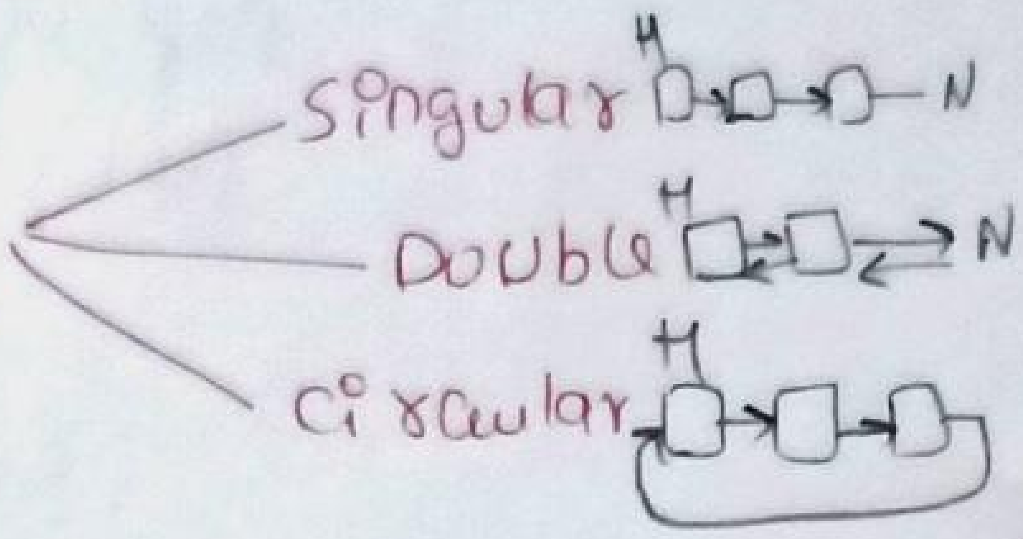
Defining a ArrayList

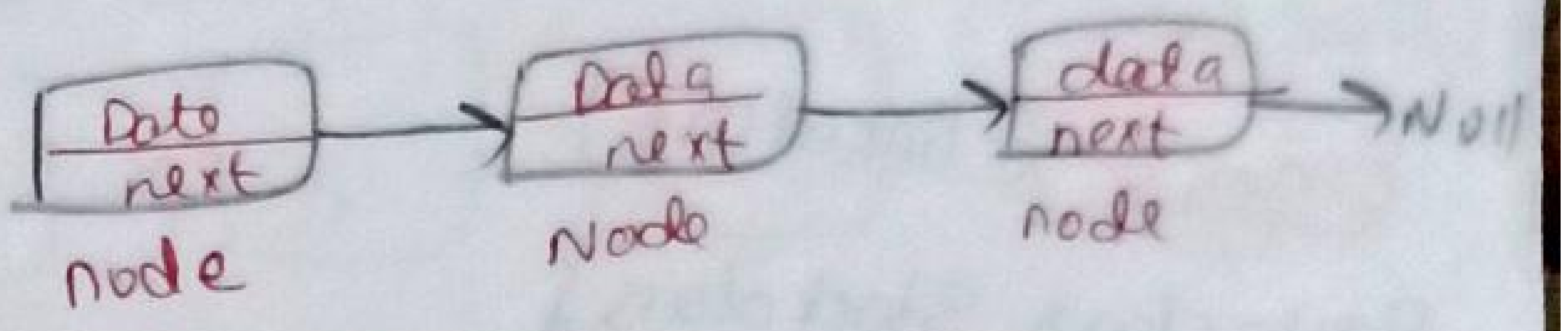
```
//Class - Integer | Float | String | Boolean

ArrayList<Integer> list1 = new ArrayList<>();
                    (or)
ArrayList<String> List2 = new ArrayList<String>();
```

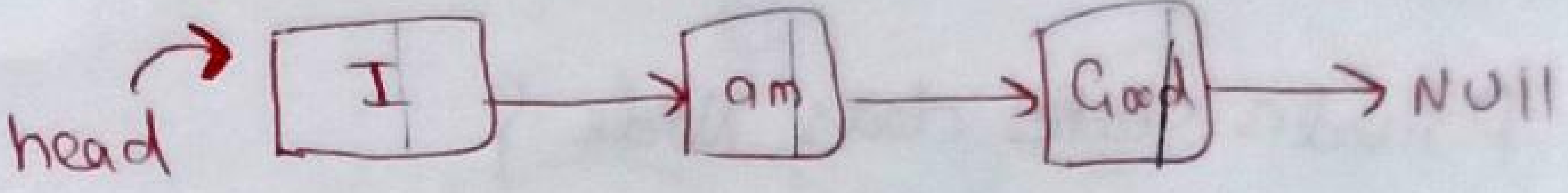
Examle

```
ArrayList <Integer> list = new ArrayList <Integer>();
list.add(1);
List.add(5);
List.add(6);
Syso(list);
```

* Linkedlist







→ Variable size

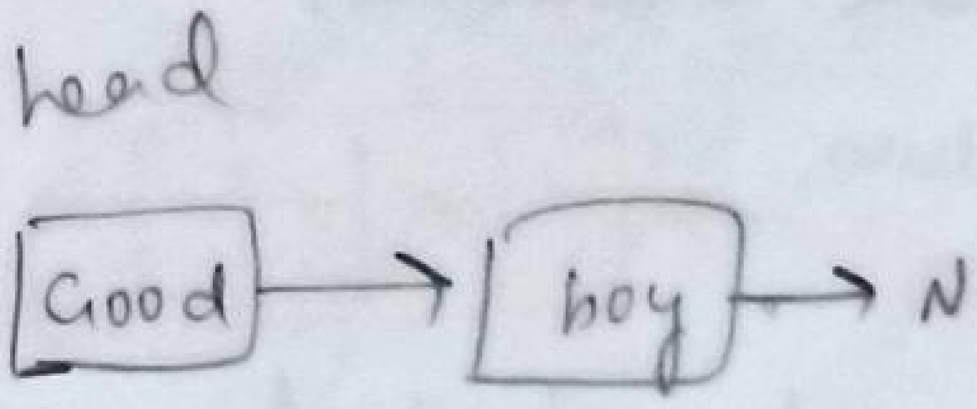
→ non-contiguous memory

→ insert in O(1)

→ search in O(n)

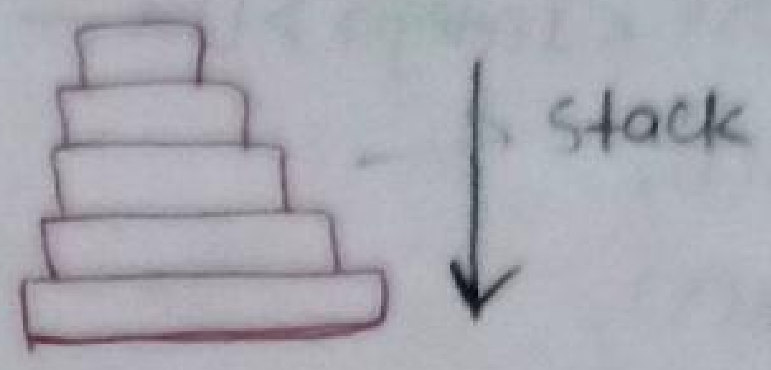
Examle

```
Class LL{
  class Node {
   String data;
   Node next;
   Node(String data){
     this.data=data;
     this.next =null;
  }}
  // add
  Public Void addFirst(String data) {
   Node newNode = new Node(data);
    if(head == Null){
      head = newNode;
     return;
    }
    newNode.next= head;
    head = newNode;
   }
   PSVM(String args[]) {
   LL list = new LL();
    List.addFirst("boy");
    List.addFirst("Goo");

   }
   }
```



* Stack

LIFO



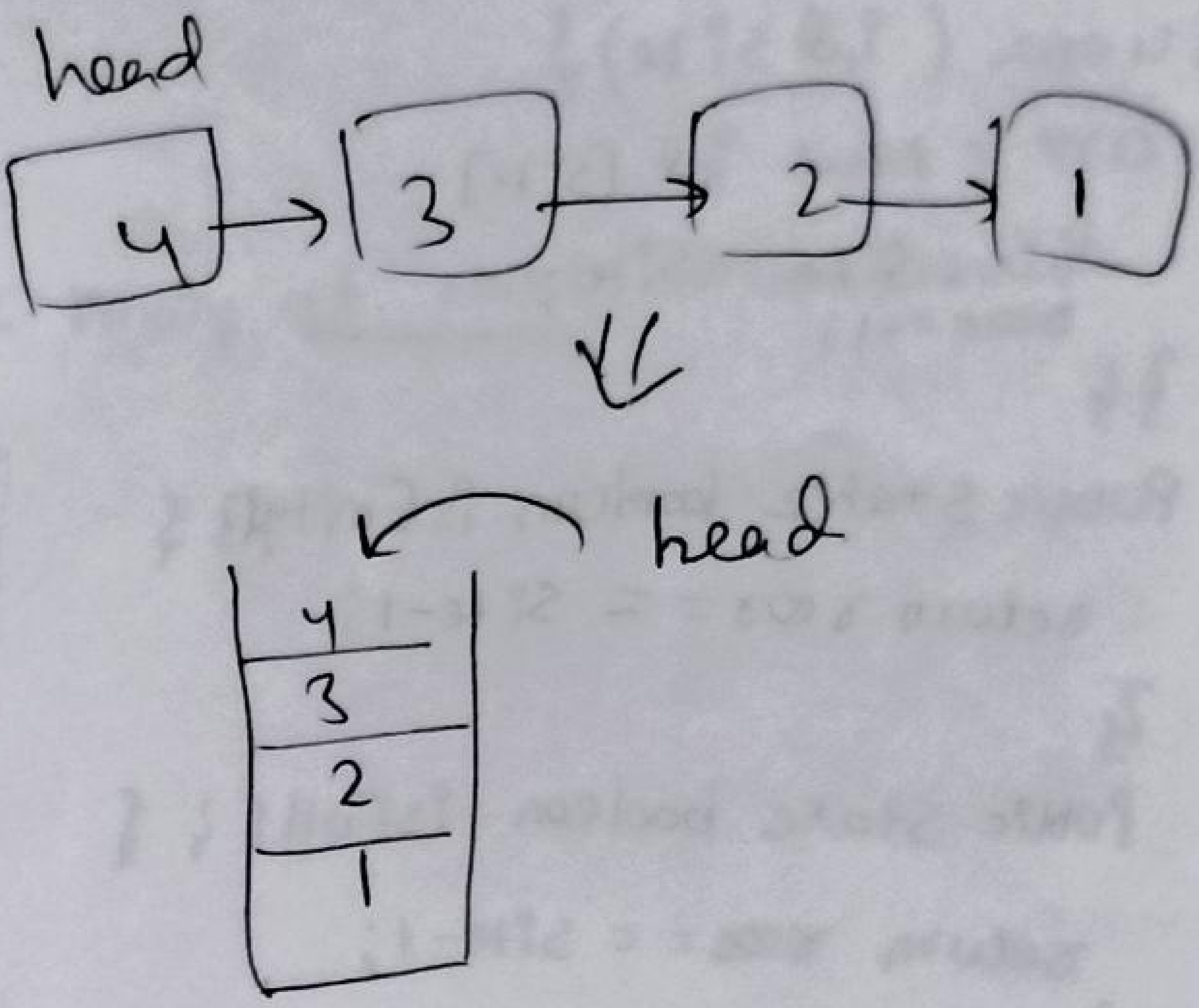
Push O(1)
Pop O(1)
Peek O(1)

Implementation

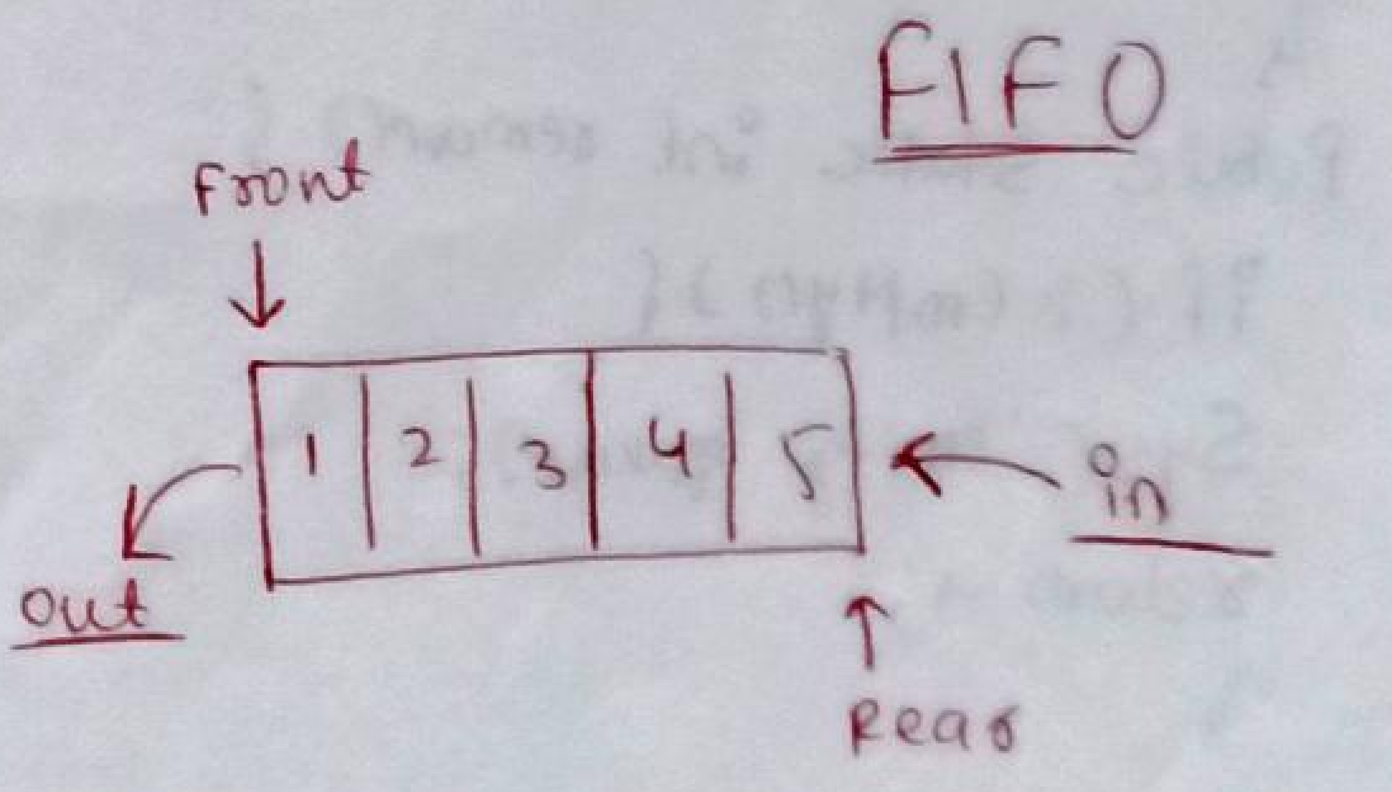
Array	ArrayList	LinkedList.	Collection
hectic approch (fixed size)	variable memory	Variable memory	framework

Example (linkedlist)

```
Public class Stackclass {
 Private static class Node {
      int data;
      Node next;
      Node (int data) {
         this.data = data;
         next = null;
      }}
    Static class Stack {
     Public static Node head = null;
      Public static void push (int data) {
      Node newNode = new Node (data);
      if (head == Null) {
      head = newNode;
      return;
      }
     newNode.next = head;
      head = newNode;
    }
     Public static boolean isEmpty() {
      return head = null;
      }
     Public static int pop() {
      if (isEmpty()) {
           return -1;
      }
Node top = head;
head = head.next;
 return top.data;
 }
 Public static int peek() {
   if (isEmpty()) {
     return-1;
   }
   Node top = head;
   return top.data;
 }}
 PSVM (String args[]) {
   stack stack = new Stack();
    stack.push(1);
    stack.push(2);
    Stack.push(3);
    Stack.push(4);
     while (!stack.isEmpty()) {
       syso(stack.peek());
      stack.pop();
    }}}
```



* Queue :- Circular Queue



Operation

enque(Add)

Dequeue(Remove)

front (Peek)

Implementation

Array   ArrayList   Linkedlist   Collectionframework

Example (using Array)

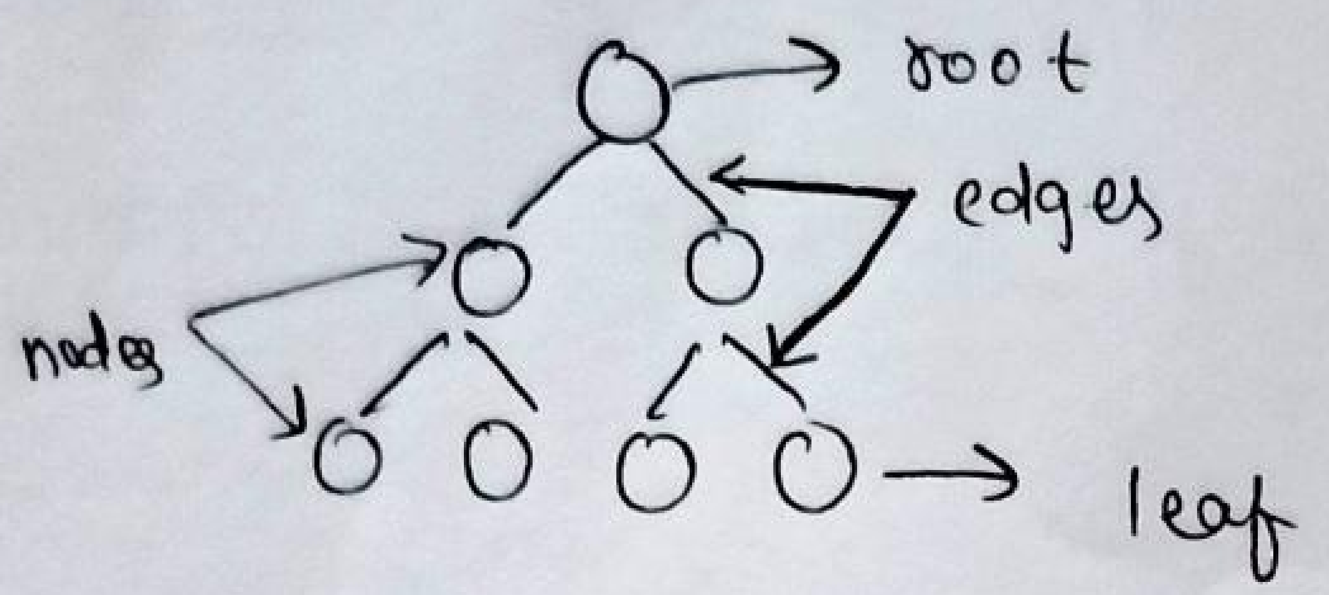
```
Public class Queuey{
 Static class Queue {
  Static int arr[];
  Static int size;  Static int rear;
 Queue ( int size){
   arr = new int [size];
    this . size = size;
    rear=-1;
  }}
  Public static boolean isEmpty{} {
    return rear== size-1;
  }
   Public Static boolean isFull{ } {
    return rear = = size-1;
   }
   Public Static void add(int data
   if(isfull()) {
    S.O.P("overflow");
    return; }
  arr[++rear]= data;
  }
  Public static int remove() {
    if (isEmpty()) {
    Syso("empty queue");
    return -1;
    }
    int front = arr[0];
    for( int i=0; i<rear; i++) {
      arr[i] = arr[i+1];
     }
     rear--;
     return front;
    }
```

```
Public Static int peek{}{
if(isempty()){
Syso("empty queue");
return-1;
}
return arr[0];
 } }
PSVM(String args[]){
Queue q= new Queue(5);
q.add(1);
q.add(1);
q.add(3);
Syso(q.remove());
Syso(q.peek());

}}
```

```
1
2
3
```

Trees

Tree is a non-linear DS



Binary tree - A tree with each node having at most 2 children

Defining Node

```
Class Node {
    Node left , right ;
    int data;
   Public Node (int data) {
     this.data = data ;
   } }
```

// Implementation

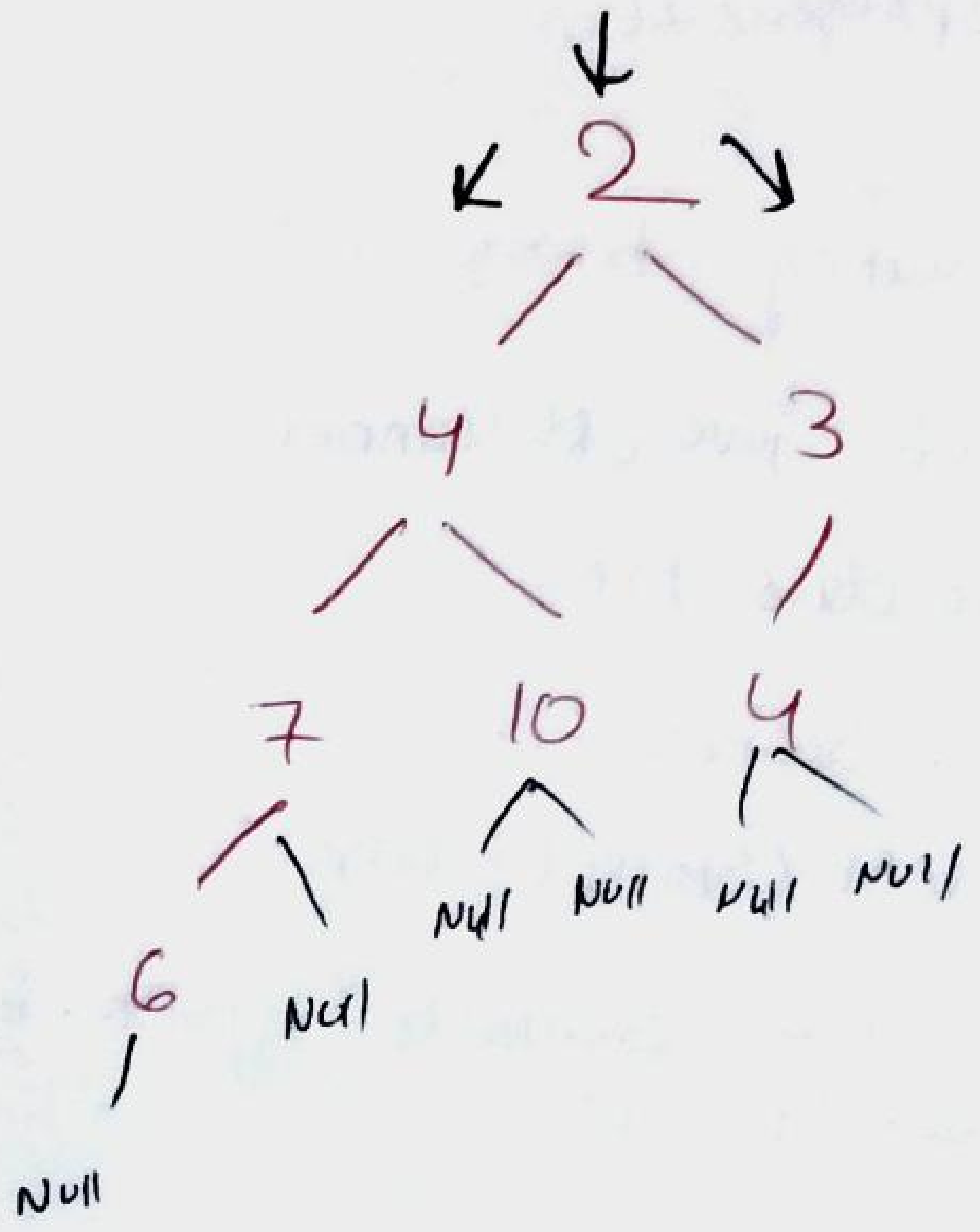
Array representation

linkedrepresentation

Ex:- Creating binary tree

```
     import java.util.scanner;
     Public class Tree
     static Scanner sc = null;
      P S V M (String [] args) {
       sc = new scanner (system. in);
       createTree();
       }
```

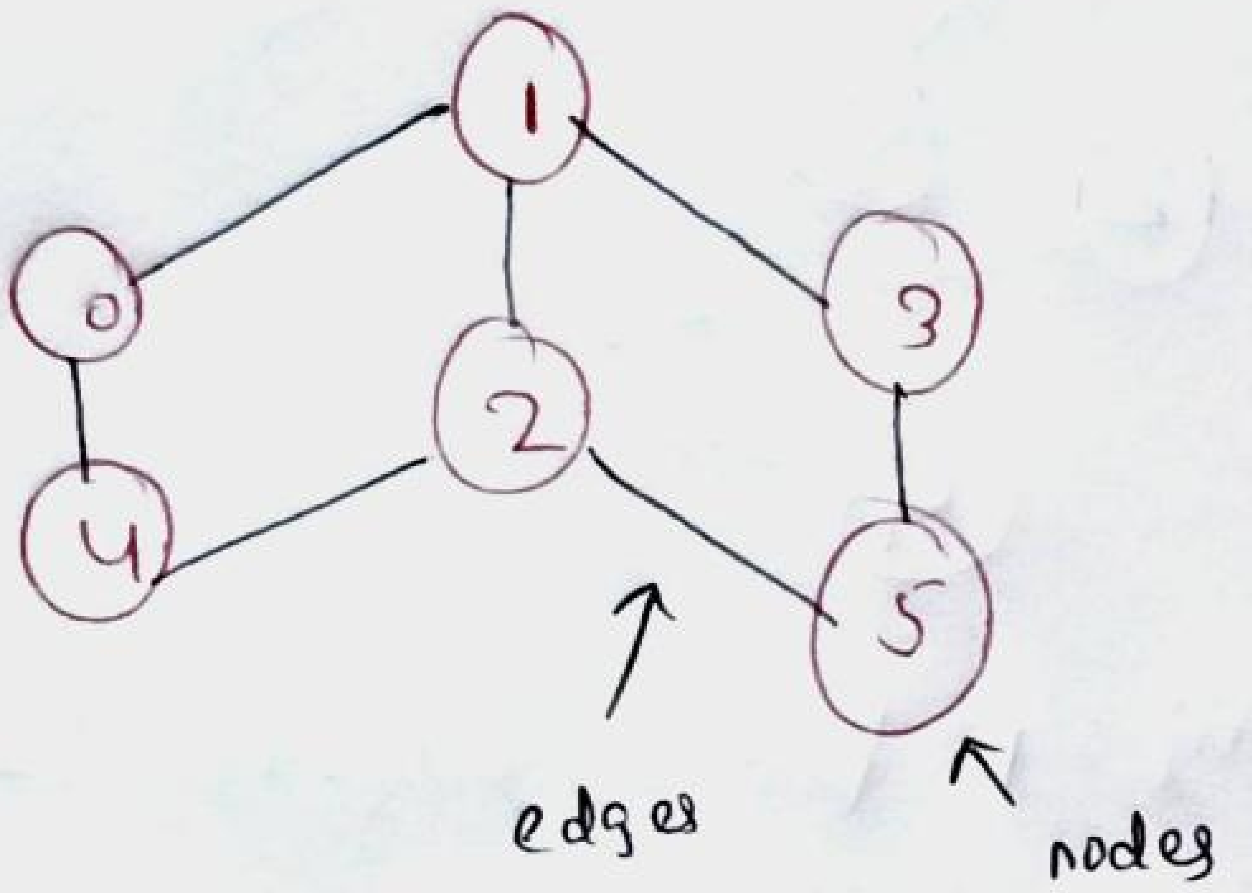
```
Static Node CreateTree() {
Node root = Null;
Syso("enter data: ");
int data = sc.nextInt();
 if ( data == -1) return null;
  root = new Node(data);
  Syso("enter left for " + data);
  root.left = createTree();
   Syso("enter right for " + data);
   root.right = createTree()

    return root;
  } }
 class Node {
  Node left, right;
   int data;

  }
```



* Graph

Graph is a non-lineare ds

Graph is a collection of nodes connected through edges



$V = \{0, 1, 2, 3, 4, 5\}$

$E = \{\{0,1\}, \{1,2\}, \{1,3\}, \{2,4\}, \{1,5\}, \{3,5\}, \{0,4\}\}$

$G = (V, E)$

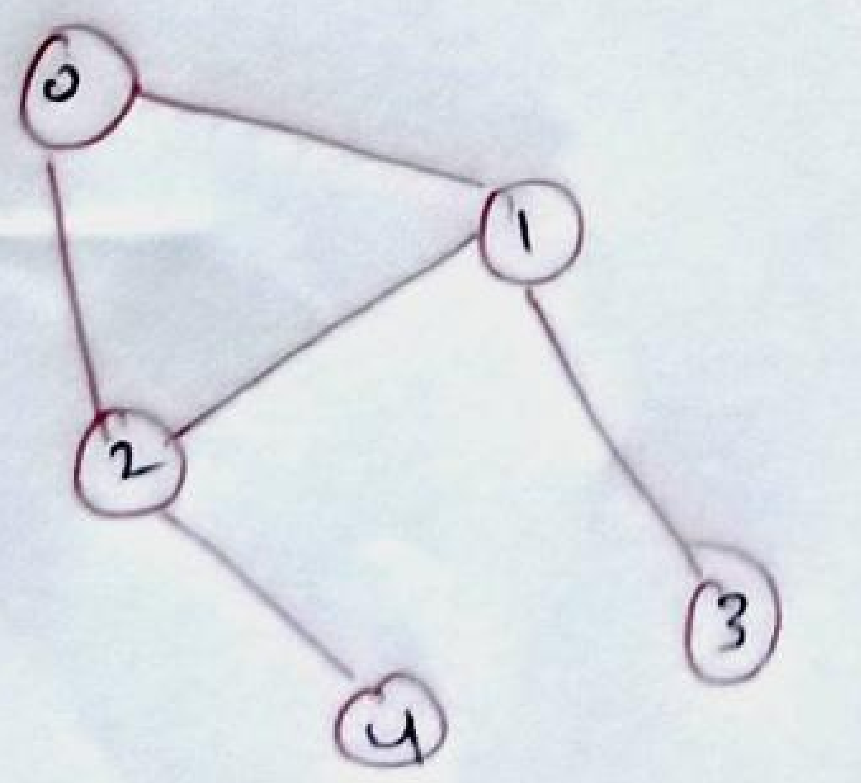
* Applications

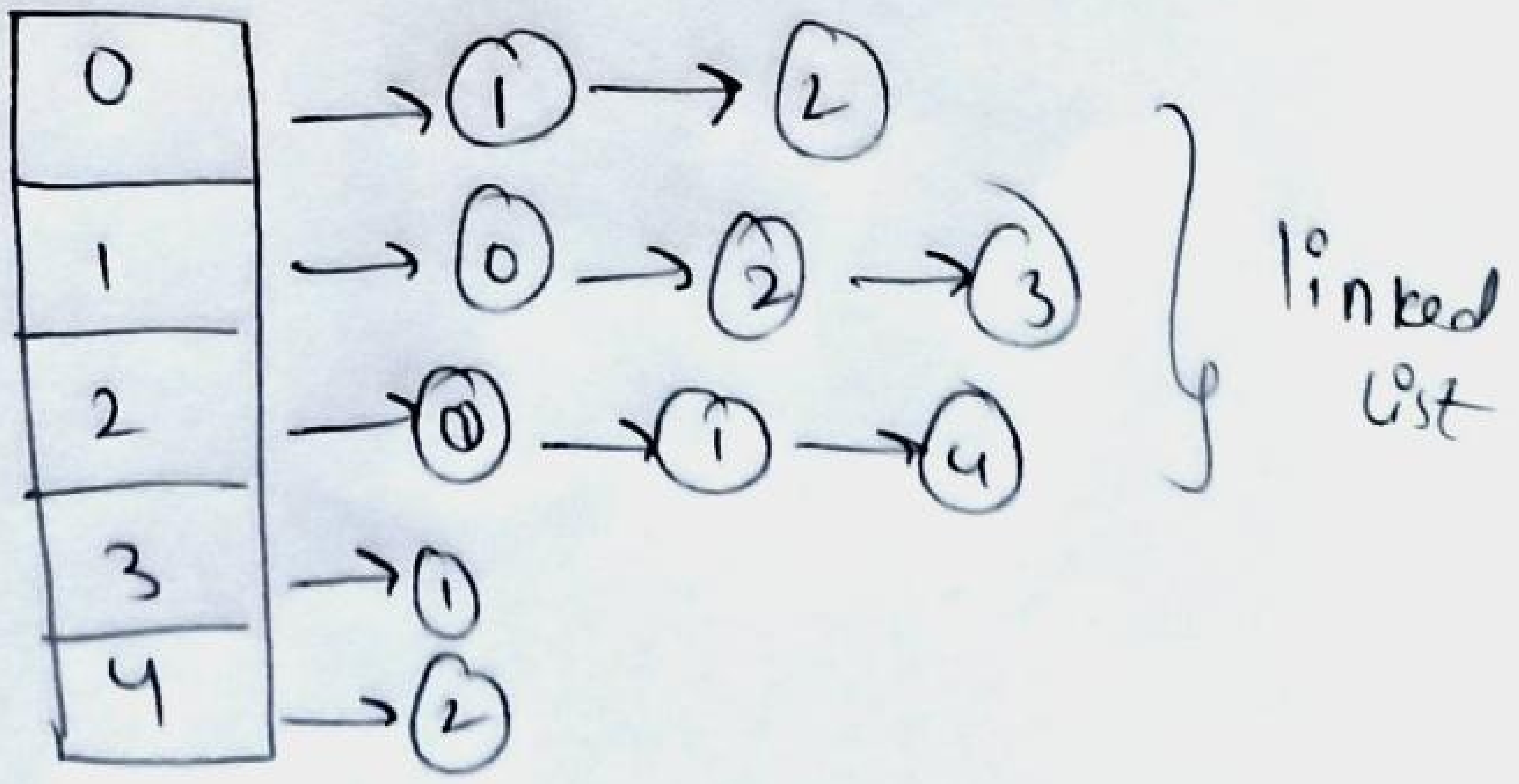
model paths of city, social networks, website backlinks, internal employee network.

* Implemenation

- Adjacency matri (1)
- Adjacency list (2)

• 2

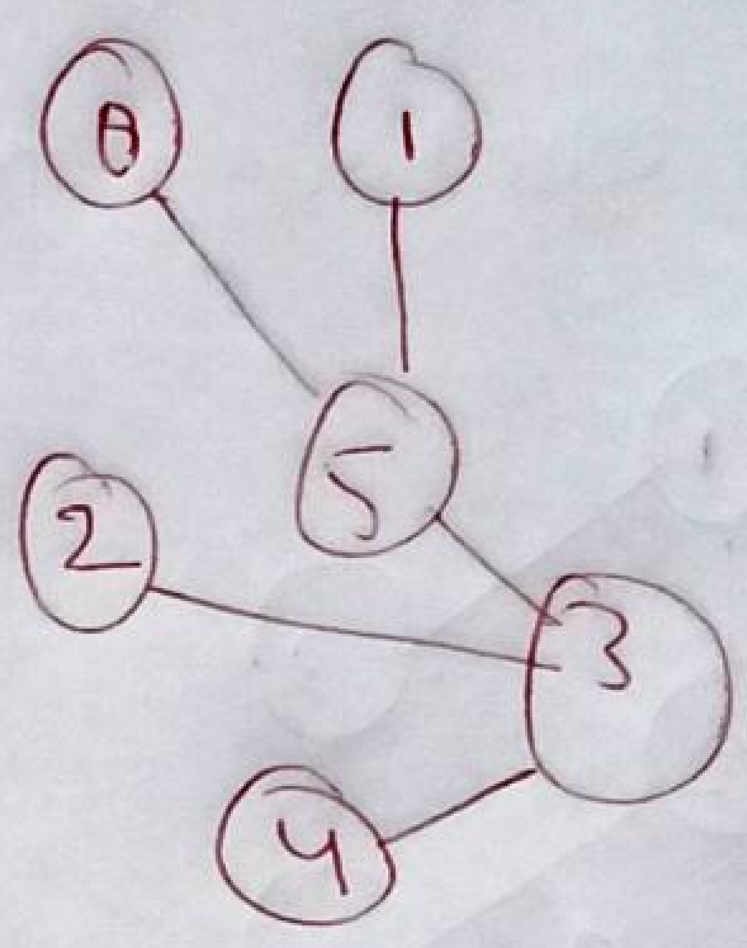




marks the node coith the list of its neishbours

• (2)

	0	1	2	3	4	5
0	0	0	0	0	0	1
1	0	0	0	0	0	1
2	0	0	0	1	0	0
3	0	0	1	0	1	1
4	0	0	0	1	0	0
5	1	1	0	1	0	0



- $A_{ij} = 1$ for an edge b/w i & j
  0 Otherwise

- Graph traversal
  - BFS
  - DFS

- Spanning tree
  - Prims Algorithm
  - kruskal Algorithm

- Single pair Shortest path
  - dijkstra Algoorithm