project → package → class

system . out . println( );

a % 2 == 0 ← odd / even.

Arithmetic operator

(+, -, *, /, %)

// — for comment

```
if (condn) {

}
else {

}
```

a == b — comparison op.

a = b (RHS to LHS) — Assignment op.

- && → T (if both are True)
- a = a + 1 (a++
- SI = (p*r*t)/100;
- A = p* (Math.pow(1 + $\frac{r}{100}$), t)
- CI = A - P.
- import java.lang.Math;

C = $\frac{5}{9}$ (F - 32).

- Parenthesis > post > pre > Multiplication > Addition > Relational > logical.
- While (condn) {

  }

□ != □ < T (if not equal) / F (if equal)

scanner sc = new Scanner (System.in);

import java.util.Scanner.

int a = sc.nextInt(); sc.nextLine (Long, float, double, Boolean)

Java

pattern -

1. n = no. of rows we want
2. i - I am working on which row
3. nst & nsp - no. of stars in 1st row
4. while (i <= n) {
5. cst & csp - As we enter no. of * printed
6. cst → nst
7. write code for 1st line
8. cst & csp)
9. write code for next line

i++;

Mirroring — Jo upper Half me Ho raha hai uska ulta karna.

for loop 1) declar.n 2) condn 3) updn.

```
for (int i=0; i<n; i++){

}.
```

prime . 2 → n

```
if n % i == 0 {
  not prime
}
else prime.
```

Time complexity - How much time code take to run.

Fibonacci a = 0, b = 1

c = a + b

a = b

b = c.

sum of digits

%10 - last digit get

/10 - last digit remove.

*10 - shift 1 digit

```
while (n > 0) {
  rem = n % 10;
  sum = sum + rem;
  n = n / 10;
}.
```

Reverse

```
while (n > 0) {
  rem = n % 10;
  sum = sum * 10 + rem;
  n = n / 10;
}.
```

HCF

```
while (rem > 0){
  rem = divident % divisor
  divident = divisor
  divisor = rem;
}
```

HCF = divisor

Decimal to Binary

```
sum = 0, mul = 1
while (n > 0) {
  rem = n % 2;
  sum = rem * mul + sum
  mul = mul * 10;
  n = n / 2;
}.
```

Data type → Stack (primitive) / Heap (Non primitive)

Bool - 1 bit, Byte - 1, short - 2, long - 4, int 8

double - 8, float - 4, char - 2

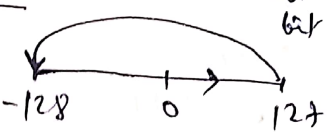
1 bit = □

1 byte = 8 bit

Data stored = $2^n - 1$

Type casting MSB = 0 (+ve) = left bit = 1 (-ve)

cycle



1's comp. change 1 ← 0

2's comp. add 1 in 1s comp.

Data type = (Data type) ☐

ASCII Code. a → 97 A → 65

Z → 97+26 Z = 65+26

prefix. - first ↑ then print

postfix first print then ↑.

FM syntax Public static void Name() {any

}.

Inverse a Number

place = 1

ans = 0

while (n > 0)

rem = n % 10

ans = ans + (int) (place * Math.pow(10, rem-1));

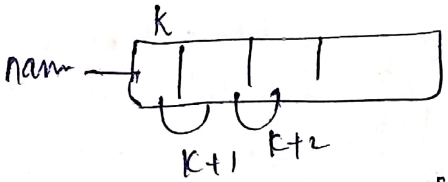
place++;

n = n / 10;

}.

Pascal Δ. $n_{C_r} = \frac{n!}{(n-r)!\,r!}$

ARRAYS: Non primitive / continuous



declar) data type [] Nam = new data type [size]

data type [] nam = {- -}

swap temp = a; a = b; b = temp;

Max. ans = Math.max(ans, arr[i]);

a = Integer.MIN_VALUE; = -2^31.

b = Integer.MAX_VALUE;

Array Reverse - Two ptr.

1 2 3 4 5 6

i ↔ j swap i++, j--

Rotating array

k = k % n.

rev(0, n-1, arr)

rev(0, k-1, arr)

rev(k, n-1, arr)

Product of array except itself

Rain water trapping.

SORTING:

Bubble [T.C = n^2]

pass = 1 → pass < arr.L

i = 0 → i < arr.L

if arr[i] > arr[i+1]

swap

}.

insertion sort

i = 1 → i < arr.L

item = arr[i]

j = i

while (j > 0 && arr[j-1] > item)

arr[j] = arr[j-1]

j--

}

j++

arr[j] = item

}.

selection sort

i = 0 → i < arr.L

minidx = i

j = i+1 → j < arr.L

if arr[j] < arr[minidx]

minidx = j

}

swap

}.

T.C

Space		B.C	Av	Worst
1	Bubble	n	n^2	n^2
1	Sel. Sort	n^2	n^2	n^2
1	Ins. sort	n	n^2	n^2
n	quick	nlogn	nlogn	n^2
n	Merge	nlogn	nlogn	nlogn
1	Heap	nlogn	nlogn	nlogn

Scanned with OKEN Scanner

KADANE ALGO = Max^n subarray sum.

```
ans = Integer.MIN_VALUE;
sum = 0
for (i=0 — arr.leng')
  sum += arr[i]
  ans = Math.max(ans, sum);
  if (sum < 0) {
     sum = 0
  }
}.
```

. **Binary search** → T.C = log n

```
lo = 0
hi = arr.length-1;
While (lo <= hi) {
  mid = (lo+hi)/2
  if (arr[mid] == target) {
     return mid;
  }
  if arr[mid] > target) {
     hi = mid - 1;
  }
  else {
     lo = mid + 1;
  }
}.
System.currentTimeMillis();
While (i<n) {   — O(n).
}
```

if i = i*2   n = n/2
T.C = log n (base 2)   TC = log n

if i++ — O(n)
i += K = O(n/K).

. i=0 — i<n   ] T.C = n^2
j = 1 — j<i   ]

. i=0 — i*i <= n {] TC = $\sqrt{n}$
}

2D Array

4x5 – array = 5 (4+1)
1 1darray — Here array addr

```
int arr[][] = new int[n][m];
```

Traverse.
Row – i < arr.length
col → i < arr.length

String: Non primitive / Heap Mem
Declar^n Data type nam = "value";
O.T nam = new OT("value");

Function.
① s.equals ( ) → T / F
② s.length ( )
③ s.charAt(i)
④ s.substring (i,j) — i inclusive, j exclusive

• String to Int
Integer.parseInt(string);

Array List:
ArrayList <D.T> nam = new ArrayList <> ();
Import java.util.ArrayList;

Wrapper Class
Java Coll^n Framework (ArrayList) ie can only store object not primitive data type. Then it is used.

Function
list.size ()
list.add (element)
list.get (i)
list.remove (elem).
list.set (elem, idx).
enhanced for loop
```
for( int i : arr) {
  sysout (i);
}.
```

. To convert String — Array.
s.toCharArray()

SORTING

Collections.sort (name)
import java.util.Collections;

REVERSE.
Collections.reverse (name);

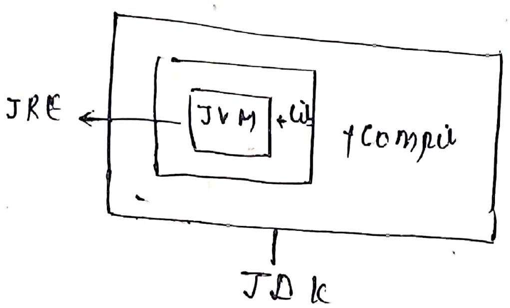
RECURSION

Tower of Hanoi

```
p.s. void
toh(int n, src, help, dest) {
  if(n == 0) {
    sysout("Move" + n + "th disc from" + src + "to" + dest);
    return 1;
  }
  toh (n-1, src, dest, help);
  toh (n-1, help, src, dest);
}
```

. List <List <String>> list = new ArrayList <>();

OOps.



JVM → Byte code → Machine code

Java → .classfile → Machine code
JVM interpret it (all OS understand it).

Object – Real world Entity / Non prim / Heap / Instance

class – Blueprint.

object form^n – using new word.

when obj form constructor call autom.

. ClassName. Datamember → to acces prop of object

. this. DM →

Static – don't make obj of it

Error Handling.
try {
}
catch (excep^n e) {
}
finally – allways run
}

Stack – D.S / linear LIFO / FILO
/ Heap / Non prim. / class in Java.
F^ns. – pop / push / is Empty, is full, capacity, peek, clear(), display()

Access Modifier.

Java Stack
import java.util.Stack
Stack <Integer> st = new Stack <>();

Interface – Queue / List / Set / Map
– Blueprint define a set of abstract method & const.
↓ w/o a body.

QUEUE – DS / FIFO / NonP.
Heap / not a class / Interface
linear
F^n. Enqueue, Dequeue, is Empty, is full, Display, clear, peak ..

· Dynamic stack – arraylist used in place of array OR inherit stack class & override push f^n.

TERNARY OPERATOR
(cond^n) ? ___ : ___
T F

Dynamic Queue.
– similar to stack just override Enqueue.

Inheritance.
Parent ki chiz child ko milna.
extends keyword.
· P class id, f() f1()
C class has f1
(extends id, f() f1())

Stack using Q.
{push efficient
{pop efficient

Queue using stack
{Enqueue efficient
{Dequeue efficient

polymorphism – multiple form (shape)
Method overload (Run time P)
Method override (compile T)
· @ override annot^n.
· To write similar f^n in C as present in P. & can modify it
overloading – has Multiple methods with same name but different parameter list.

Linked List.
collection of nodes

Linked List middle.
while (fast != NULL && fast.next != null)
move fast & slow
return slow.

· Reverse LL.
curr = head.next
head.next = prev
prev = head
head = curr
return prev.

· cycle detect^n
fast == slow
cyclic(L)

TREES

- Non linear D.S.
- Not built in Java

Generic Tree | Binary Tree (at most 2 child) | Binary search Tree

Max^m value in Tree:
- Get leftmax & Get right max. and compare both.

Traversal

- **Preorder:** Root → Left → Right
- **Post order:** Left → Right → Root
- **Inorder:** Left → Root → Right
- **Level order** – Travel level by level
  - Queue used

DFS – Go in depth

BFS – Not Go in depth.

Inbuilt Queue:

```
Queue<Node> q = new LinkedList<>();
```

BST – For every node left child should be smaller & right child should be greater

Generic:

Heap: → Complete Binary Tree, Priority

- Max (Parent > child)
- Min (Parent < child).

Heap	
add	O(log(n))
Remove	O(log n)
get	O(1).

Heap in built in Java

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
```

Abstraction:
Process of hiding certain details & showing essential information to user.

Abstract class
- should have atleast one abstract method.

Abstract method
- don't have body.

Interface:
It is abstract class used to group related method with empty body

Access Modifier:
- **Private** – same class only
- **Protected** – same package
- **public** – subclass + outside package
- **default** – strictly inside package

MAP – Interface key value pair | key – unique

declaration:

```
HashMap<key, value> map = new HashMap<>();
```

Functions:
1. map.put(k,v); — O(1)
2. map.containKey("K") — O(1)
3. map.get("K") — O(1).
4. map.remove("K") — O(1)

HashMap	LinkedHashMap	TreeMap
key order not mapped	key order maintain (insertion order preserved)	(key in sorted order)
	lmap.put(v,k)	tmap.put(k,k)

Set

- Insertion order maintain / collection of distinct values / No duplicate

HashSet	LinkedHashSet	TreeSet
insertion not maintain.	insertion order maintain.	sorted

- set.add()
- set.contains()
- set.remove()

— O(1)

Graph :- Non Linear / Edges vertices / No Hierarchy / connect b/w vertices.

- HashMap of HashMap used.

Scanned with OKEN Scanner