Mo Tu We Th Fr Sa Su

Date / /

* DSA WITH KUNAL - JAVA

⇒ Lecture-1:-

Types of languages → Procedural
→ Functional
→ Object-Oriented.

- Procedural
  - Specifies a series of well structured steps & procedures to compose a program.
  - Contains a systematic order of statements, functions and commands to complete a task.

- Functional
  - writing a program only in pure functions i.e never modify variables but only create new ones on output.
  - Used in situations where we've to perform lots of diff operations on the same set of data, like ML.

- Object Oriented
  - Revolves around objects.
  - Code + Data = Object
  - Developed to make it easier to develop, debug, reuse & maintain software.

Mo Tu We Th Fr Sa Su

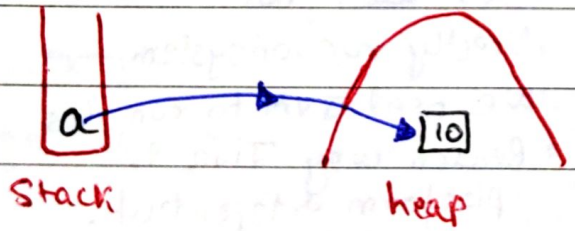
Date / /

## Static Vs Dynamic

- Static
  - Perform type checking at compile time
  - Error will show at compile time.
  - Declare datatype before you use it
  - More control.

- Dynamic
  - Performs type checking at runtime.
  - Errors might not show till program is runned
  - No need to declare datatype of variables.
  - Saves time in writing code but might give errors at runtime.

- Stack vs Heap Memory.




a=10 → object
↓
ref variable.

- more than one ref var can point to same object.
- If any one ref var change the obj, it'll appear for all ref va
- JAVA has only pass by reference.

Mo Tu We Th Fr Sa Su

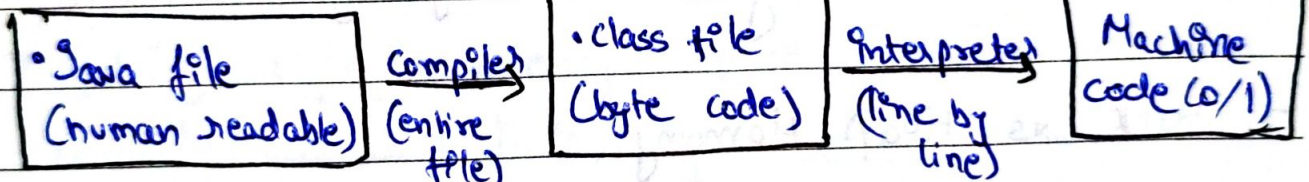
Date / /

- Garbage Collection
  - object with no ref var → this'll be removed when garbage collection hits.

- Flowchart & Pseudocode

| | |
|---|---|
| ⬭ → start/stop<br>▱ → input/output<br>▭ → process<br>◇ → condition | Pseudocode is similar to python code 😊. Easy eng. that tells what will prog do at each step. |

- JAVA BASICS ( Compiling / Interpretation)




- this is the source code
- this code'll not directly run on system.
- We need JVM to run this
- Reason why Java is platform independent.

Date / /

## COMPILE TIME

JAVA FILE
↓ (java compilation)
.class file

- JVM Execution

Interpreter:-

- line-by-line executions.
- when one method is called many times, it'll interpret again
- JIT
- those methods that are repeated, JIT provides direct machine code so reinterpretation isn't req.
- makes execution faster.
- Garbage collector.

## RUNTIME

Class Loader
↓
Byte code verifier
↓
Interpreter
↓
Runtime
↓
Hardware

- (How JVM WORKS) Class loader

- LOADING
  - reads class file and generates binary file
  - an object of this class is created in heap.
- LINKING
  - JVM verifies class file
  - allocates memory for class variables & def. values.
  - replace symbolic ref from the byte with direct ref.
- INITIALIZATION
  - all static var are assigned with their values def in the code and static block

JVM contains the stack & Heap memory allocation.

Good judgement comes from

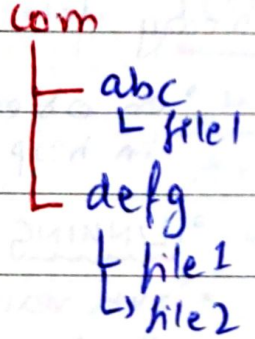
Mo Tu We Th Fr Sa Su

Date / /

## Some basics to start Java.

- If you are creating class - first letter should be capital.
- In Java, where your program starts is the main function.
  → public static void main (String[] args) { }.
- javac is used to compile in terminal.

- Public means that this class can be accessed from anywhere.
- main function is the entry point of the program.
- Static → we want to run the main function without creating object of class.
- void returns nothing
- whatever args you pass at command line, are saved in the variable args.
- package com.abc OR com.defg




- System.out.println ("Hello"); This means print the output on the std output stream (here, terminal)

  System → class
  out → var
  println → fn/method

println → adds new line

print → no new line is added.

Date / /

* Scanner input = new Scanner (System.in);
  - Scanner → class that allows us to take input.
  - new → creating obj
  - System.in → take input from std input (here, Keyboard)

* Primitives → any datatype you can't break any further.

int, char, float, double, long, boolean
- int → 4 bytes
- char → 4 bytes
- float → 4 bytes
- double → 8 bytes
- long → 8 bytes

for float → we add (f) because by default decimal points are considered as double.

* Literals & Identifiers,

int a = 10;
- a → identifier
- 10 → literal

Literals :- Java literals are syntactic representations of boolean, character, numeric or string data.

Identifiers :- Identifiers are the names of variables, methods, classes, packages and interfaces.

- int a = 234_000_000
  ↳ the value of a will be 234000000, underscores will be ignored.

⇒ • nextInt • next() • nextLine()
↳ takes integers ↳ takes string upto space. ↳ takes entire string with space

⇒ floating point → rounds it off → thus its not accurate.

- Type casting and Type conversion

⇒ Widening or Automatic Type Conversion
→ Two datatypes are automatically converted.
→ two conditions :-
① we assign value of smaller datatype to bigger
② two datatypes must be compatible
byte → short → int → long → float → double

⇒ Narrowing or Explicit Type Conversion :
→ This happens when we want to assign a value of larger datatype to a smaller one.

double → float → long → int → short → byte

eg :- int num = (int)(67.56f)
↳ output = 67.

Mo Tu We Th Fr Sa Su

Date / /

- Automatic Type Promotion in Expressions

→ while evaluating expressions, the intermediate value may exceed the range of operands & hence the expression value will be promoted

→ conditions :-

⊙ Java automatically promotes each byte, short, char to int when evaluating an expression.

⊙ Long, float or double the whole expression is promoted to long, float or double.

eg :- (f * b) + (i / c) - (d * s)

float int double = double.

converted to biggest one

* Explicit type casting in expression.

- If we want to store large value into small datatype.

eg :- byte b = 50

b = (byte) (b * 2) → b*2 → int so we cast into into byte.

Mo Tu We Th Fr Sa Su

Date / /

## * Switch Cases / Nested Case

String comparison

if :-

a → "Vishu"

b →

i.e a,b are both pointing towards same object ; a==b ⇒ TRUE (a.equals("Vishu"))

↳ checks position or obj

↳ checks value

∴ Thus, to compare strings we use .equals when we need to check only value.

### Switch Case SYNTAX

In switch statements, you can jump to various cases based on your expression.

```
Switch (expression) {
    // cases
    Case one:
        // do something
        break;
    default:
        // do something
}
```

Mo Tu We Th Fr Sa Su

Date / /

NOTE:

- cases have to be the same type as expressions, must be a constant or literal.
- duplicate case values are not allowed.
- break is used to terminate the sequence
- if break is not used, it'll continue to next
- default will execute when no cond. is satisfied.
- if default not at the end, put break after it.

ENHANCED SWITCH SYNTAX :-

```
switch (expression) {
        case one → do this;        (auto break)
        case two → do that;
        default → do this;
    }
```

* You can also use nested switch statements as per the question/situation requires.

Mo Tu We Th Fr Sa Su

Date / /

## * FUNCTIONS / METHODS

=> a block of code which only runs when it is called.

=> it is defined once and can be used multiple times

Syntax:

```
access modifier  return-type name() {
                    // body
                    return statement ← ends here
                }
```

① return-type

A return type statement causes the program to transfer back to caller of the method.

A return type may be primitive type like int, char or void type (returns nothing).

* The type of data returned by method must be compatible with the return type specified by the method.

* The variable receiving the value returned by a method must also be compatible with the return type specified for the method.

* No pass by reference on JAVA, only pass by value.

Mo Tu We Th Fr Sa Su

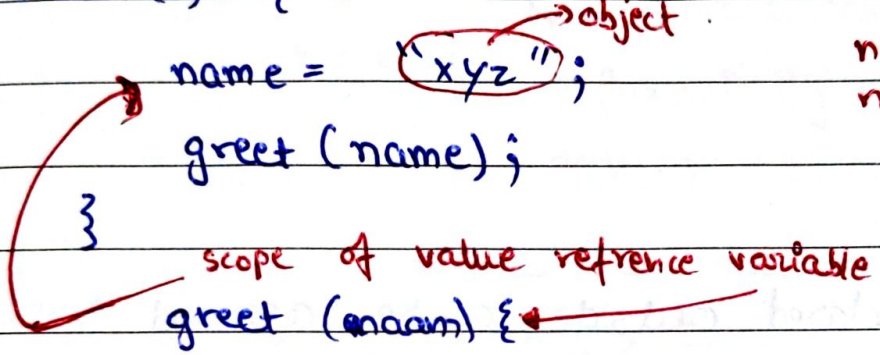
Date / /

## ① Arguments.

⇒ these are used to pass the value of numbers / arguments when you are calling the func on the main() method.

- Pass By Value

```
main () {
    name = "xyz";
    greet (name);
}
greet (naam) {
    print(naam)
}
```




// Therefore if in function we swap or change value it doesn't change coz variable value doesn't change local variable has local scope.

* But in case of array's, if we change value at any position in a method it will reflect in all as it changes the object in main array.

* 1. Primitive data type : → Just pass value

2. Object & reference : → Pass value of reference.

Date / /

Mo Tu We Th Fr Sa Su

* Strings are final classes, you cannot modify it they are immutable for security reasons.

⑥ Scoping

⇒ fn scope : variables defined/declared inside a method/fn scope (means inside method) can't be accessed outside the method.

⇒ block scope:

variables declared outside can be assessed inside. But declared inside cannot be used outside.

⇒ loop scope: Variables declared inside loop are having loop scope.

• Shadowing

Practice of using variables within the same in overlapping scopes where the variable in low-level scope overrides the variable of high level scope.

Mo Tu We Th Fr Sa Su

Date / /

eg:- public class shadowing () {

static int x = 90;

psvm () {

int x;

x = 40;

}

}

scope of this local variable that shadows the class variable is not beginning at the point where the local scope begins but shadowing begins actually when local variable is declared.

↳ x = 40;

- Variable Arguments (varArgs)

Variable arguments is used to take a variable no. of arguments

A method that takes a variable no. of args is a varargs method.

fnc called be called by more than one argument.

The input will always be of type array.

variable length arguments always at end.

Syntax: int ... v or String ...v

Mo Tu We Th Fr Sa Su

Date / /

- Function Overloading / Method overloading.

Two or more functions can exist of the same name if the parameters are diff$^n$.

eg:-

```
fun (int a) {
        //code
    }
fun (string a) {
        //code
    }
```

This is allowed having diff$^n$ arguments with same method name.

⇒ At compile time, it decides which fn to run based on passed arg$^n$.

Mo Tu We Th Fr Sa Su

Date / /

# ① Arrays / Array List

## *SYNTAX

⇒ datatype[] var_name = new datatype[size];

(or directly)

⇒ datatype[] var_name = { your array };

1. datatype represents datatypes of the elements in the arrays
2. all datatypes in an array will be the same.
3. int[] ros; ← declaration of array, ros is getting defined in stack.
4. ros = new int[5]; ← actually here obj is being created in the memory heap

(initialisation)

## ② Working of an Array

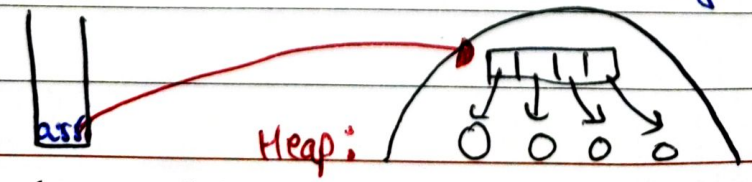
compile time | runtime

int[] arr = new int[5];

- int[] → datatype
- arr → ref variable
- new int[5] → creating obj in heap memory.
- declaration at compile time.
- initialisation at compile time.

⇒ Dynamic Memory Allocation

at runtime/execution time memory is allocated



* Internal Representation of Array.

⇒ In C/C++ array is allocated continuous memory locations

⇒ In Java, it depends on JVM.

⇒ Array objects are in heap ⇒ heap objects aren't continuous

⇒ Dynamic Memory Allocation :- Hence array objects in Java maybe or maynot be continuous, depends on JVM

* Index of an array.

| 3 | 8 | 9 | 10 | 53 |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

⇒ len = 5.

* new keyword

⇒ new keyword is used to create an object.

⇒ Initially if we don't provide values in the array,

i.e. int arr = new int[5] ⇒ {0, 0, 0, 0, 0}
all zeros

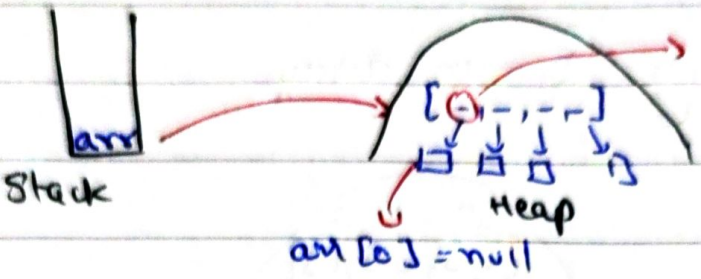
by default it has all values to 0.

* String [] arr = new String [5];

* Internal Working of Objects arrays.

Mo Tu We Th Fr Sa Su  Date / /

* primitives ( int, char, etc ) are stored in stack memory
* All other objects are stored in heap memory.




→ each particular element is an obj and will be stored in diff part of heap memory.

⇒ each element has a reference var ex: arr[0].

⇒ by default ref var points to null.

* for each loop → replacing for (int i=0; i< arr.length; i++) { } with for (int num : arr) { } → here num represents element of array.

```
for (int num : arr) {
    Sout (Array.tostring (num))
}
```

* To string :- Array.toString (arr) ↓ converts to string.

* array are mutable & Strings are immutable.

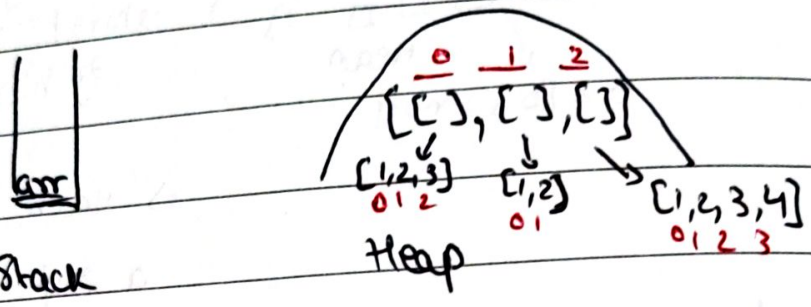
Date / /

Mo Tu We Th Fr Sa Su

## (*) 2D Arrays in JAVA

⇒ can be assumed like a matrix [2×2]

* Internal working of an array.




arr[1] = [1,2]

arr[1][0] = [1]

⇒ column size is not necessary ~~⇒~~ size of the individual row does not matter.

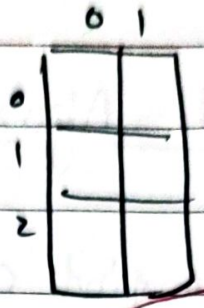
⇒ i.e {{1,2,3}, {4,5}, {6,7,8,9,10}}

this is also accepted.

Mo Tu We Th Fr Sa Su

Date / /

* If we have




or arr.length

```
for (row=0 ; row<3 ; row++) {        or arr[row].length
    for (col=0; col<2 ; col++) {
        arr[row][column] = input ;
    }
}
```

Simplified for loop for printing.

```
for (int[] num : arr) {
    sout (num)
        ↳ Array.toString(num)
}
```

earlier num had each element, here it has first row which is first element in arr.

* Array list : part of collection framework & provides dynamic array slower than std.

Internal working of Array list:

- size is fixed in memory
- if list is finished it automatically creates new & large size copies items and deletes old one.

Syntax:

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

Mo Tu We Th Fr Sa Su Date / /

## ✱ Linear Search Algorithm

In array we can find certain value using linear search by using for or for-each loop.

eg:- {1, 12, 14, 9, 5}

to find 14 we will use for-loop and at index ② we'll get 14.

→ Use for loop if index is of use
→ Use for-each if only element is of use.

---

## ✱ Binary Search Algorithm

→ The data must be sorted for binary search.

Algo: ① find the middle element.
② remove the unnecessary side.
③ if middle ele == target //end.

ascending ⇒ if elem < mid ⇒ end ⇒ mid-1
elem > mid ⇒ start ⇒ mid+1

$\log_{10}(num) + 1$ ⇒ no. of digit in num.



Mo Tu We Th Fr Sa Su

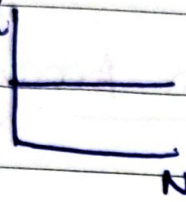
Date / /

# Time Complexity.

## Linear search.

⇒ Best case

O(1)

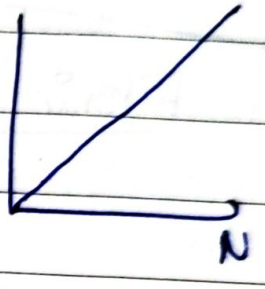



Worst case

O(N)

↓

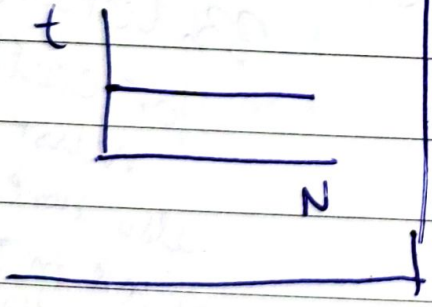
no. of elements in array



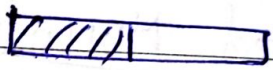

~~time~~ time grows as input grows.

⇒ Best case

O(1)




Worst case (N elements)



$N/2^0 = N$

$\frac{N}{2} = \frac{N}{2}$

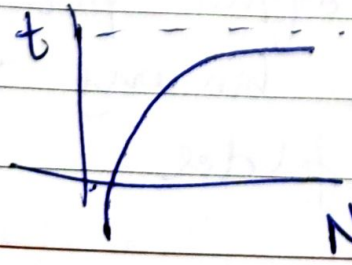
$N/2^2 = \frac{N}{4}$

$N/2^3 = N/8$

$N/2^k = 1$

$N = 2^k$

$k = \log_2(N)$




∴ it is better than linear search as time not grows linearly with N

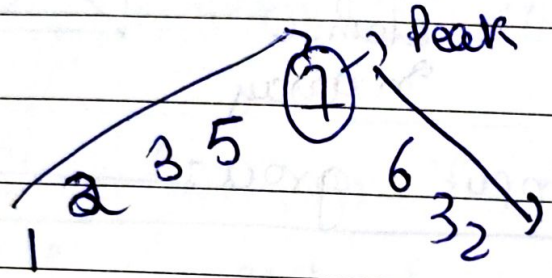
## ✱ Order Agnostic Binary Search.

① find the order of sort (ascend/descend) by comparing first and last element.

② Then apply respective code.

:→ code on pc under 6th Binary Search folder.

## ✱ Bitonic Array & Mountain Array



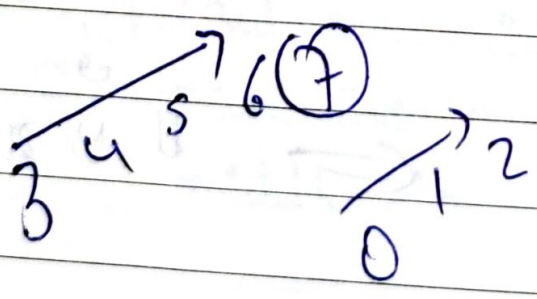

⇒ Bitonic Array.

To find peak code on Binary Search folder

## ✱ Rotated Sorted Mountain Array

33. leetcode for clarity must also done by KK on 2nd & main Binary video

[3, 4, 5, 6, 7, 0, 1, 2]




To find peak, code on binary search folder.

Mo Tu We Th Fr Sa Su

Date / /

## Binary Search on 2-D Array's

- By using normal linear search in it time complexity is $O(N^2)$ for NxN matrix.

⇒ Condition ① :- → Sorted row wise & column wise

| | | | |
|---|---|---|---|
| 10 | 20 | 30 | 40 |
| 15 | 25 | 35 | 45 |
| 28 | 29 | 37 | 49 |
| 33 | 34 | 38 | 50 |

To find 37:

we take 10 & 40
lb upper bound

we conclude that 40 > 37 ∴ whole column can be eliminated. So col --.

Now we move to 30 we say 30 < 40 so all element before 30 are < also, r++, So search space becomes

| | | |
|---|---|---|
| 15 | 25 | 35 |
| 28 | 29 | 37 |
| 33 | 34 | 38 |

Now 35 < 37 again row++

Now 37 == target.

Time complexity ⇒ N+N → 2N → O(N)

Code on Binary folder RowColMatrix.

Mo Tu We Th Fr Sa Su

Date / /

⇒ Condition ②: Matrix is sorted completely

(M-1)

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

~~X~~ Take middle col & perform B.S on 9th.

if we need to find ②
& we get 6 as first in search
6 > 2 ∴ we can say all elements below 6 in all cols & all elements after 6 in that row can be eliminated & we get

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |

Break into parts & apply B.S.

O(log(N) + log(M))

↓ not necessary

case ①
target = ele.
case ②
ele > target
// ignore rows after it
case ③
ele < target
// row above ignored

→ code in binary Sorted Matrix.java

Date / /

M-2

We also do by converting matrix into one array then get indexes by using [array index of one array // col, " % col].

eg:-

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

0 1 2 3 4 5 ← index

∴ 6's index in matrix is [5//4, 5%4]

⇒ [1, 1]