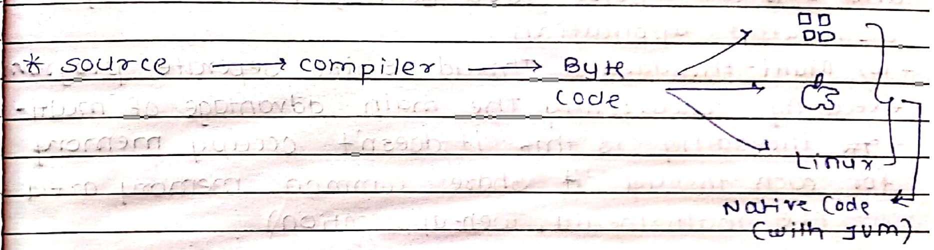
Java EE (advanced)
Java ME (for mobile, not android).

/* Java is platform Independent language
(can work on various OS)

*1



* feature of Java :- 1) simple
2) object-oriented.
- 3) Platform Independent
- 4) secured
  - ⓐ No explict pointer
  - ⓑ Program run inside virtual machine sandbox.
- 5) Robust (means ⟶ strong). ⟶ ⓐ It use strong memory management.
  - ⓑ There is lack of pointer that avoid security problem.
  - ⓒ Java provide automatic garbage collection which run on JVM to get rid of object which are not being used by java application anymore.
  - ⓓ There are exception handling (run time error) and type checking mechanism
- 6) Architecture - Neutral - There is no implementation dependent feature.
  - e.g. size of ~~int~~ datatype in 32x → 2(int) 4(float)
    64x → 4(int) 4(float)

    But in java int=4byte for both architecure
- 7) Portable - It make possible to carry java byte code to any platform. It doesn't require any implementation.
- 8) High performance.

-9) Distributed - We can create distributed application in any platform. It doesn't require any implementation in java. It use RMI (Remote Method Invocation) and EJB (Enterprise Java Beans) for creating distributed application.

-10) Multi-threaded :- Thread is like seperate program executing concurrently. The main advantage of multi-threading is that it doesn't occupy memory for each thread, it share common memory area (ie multi-media, web-application)

#2

- JVM - Java Virtual Machine.
  :. use to convert byte code → Native code.

- JRE - Java Runtime Environment.
  JRE = JVM + Tool's + Java core libraries
  Ilpn compile karu shakat nahi

- JDK - Java Development Kit

  JDK = java compiler + JRE + Tool's + Java API & libraries.

#4

* Everything in Java goes inside a class.
* class name should be same a file-name (work as main fun)

```
i.e class first {
      public static void main (string args[])
      {
        sout ("Hello");
      }
    }
```


#5

- Java file can have multiple classes but only one class can be defined as public class inside java file. (Save pn typeings en Revang)

#6 DATA TYPE

1) Integer.
   - Byte (1 Byte) = -128 to 127
   - short (2 Byte) = -32768 to 32767
   - Int (4 Byte) = (- 2147483648 to —)
   - long (8 Byte) = (- 9.22e18 to 9.22e18)

2) floating Type
   - float - 4 byte
   - double - 8 byte

3) character Type
   - char - 2 byte.

4) Boolean Type.

· Data Declration :- same as previous language

#7 Literals :- constant / fixed value
   - Whatever value you type into a code.

Type - int, char, boolean, string, floating point (Float, Double

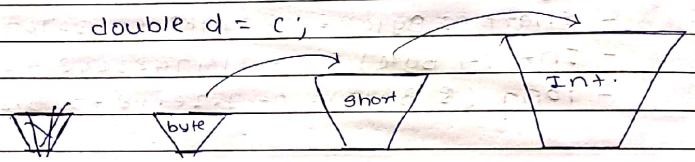
#8 Casting
   - converting value of one data type to value of another data type
   - Types : Implict
             Explict

(1) Implict Type Casting :-

या मध्ये बाकी काही नाही फक्त small data type big मध्ये fit होतात

for e.g

```
int a = 6;
long b = a;
float c = 6.011f;
double d = c;
```



(2) Explicit :- या मध्ये जरी value बरोबर असली तरी पण error येतो

for e.g

```
int a = 20;
byte b = a;  // error return properties mule.
// compiler think a is bigger than b.
```

Solution :-

```
int a = 20;
byte b = (byte) a;
```

✓ explict mahnje apan convert karto
✓ implict mahnje compiler convert karto

ASCII value are subset of unicode.

Unicode value.

```
Sout(" (int) 'ऱ' ");
```


operators.

- , + , * , / , % , ++x , --x , x++ , x-- , > , < , >= , <= , == , !=

- , + , * , / → Arithmatic
% → modulus
++x , --x , x++ , x-- → Unary
> , < , >= , <= , == , != → Relational.

&& , || , ! → logical

condition , true ? false → Ternary

* short Circuit →

```
int i = 9 ; int j = 10 ;
Boolean k = (i == 9 || ++j = 11)
```

++j होणारच नाही evaluate नाही होत

/* आता या मध्ये OR मध्ये 1 जरी true असेल तर सगळं true होतं Java first check kalya nantr second check karat नाही */ (Short Circuit).

#11 IF ELSE STATEMENT

#12 SWITCH CASE, BREAK KEYWORD)

If else madle saglya condition evaluate hot

Switch case without break statement evaluate hotana sagle khalchya statement evaluate होतात

13 For loop.

14 while, do while.

Buffered Reader.

#15 break, continue.

#16 Scanner Class:

* import java.util.Scanner;
* System.in --> mahnje cmd/terminal kadun input ghene

⇨ Input stream Reader and Buffered Reader.

→

Input stream Reader read input from user and convert to character.

Buffered Reader, buffer character and we use readLine() method.

for e.g.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args)
throw IOException {
    InputStreamReader tc = new InputStreamReader(System.in)
    BufferedReader sc = new BufferedReader(tc);

    String msg = sc.readline();
    }
}
```

#20 ARRAY

syntax:-

```
int[] arr = new int[4];
arr[0] = 1;
arr[1] = 2;
arr[3] = 4;
```


Default value :- // Jr kahi assign nahi kel tr.

1) Int → 0
2) double → 0.0
3) float → 0.0f
4) char → □
5) String → null
6) Boolean → false.

Tip :- // cannot take boolean input from user.

2-D-Array.

syntax :-

```
int[][] arr = new int[3][3];
```

* To check Length.

- arr.length for 1D-array.

- 2D-Array madhe length nahi betat
  Length of array = no. of row it has

- To check length of array (2D) & which we cannot get but can get no. of rows & no. of column)
  i.e
  arr.length → no. of rows.
  arr[0].length → no. of column.

#21 → Methods * // same as function.

- Method is block of statement that perform certain action or functionality
- It can help to reuse code.
- Java also provide predefined method such as System.out.println()

for e.g

```
Return-Type f-name (param1, parm ---)
{
  // code;
  returntype value;
}
```

Code With Harry (start).

Naming Convention!-

1) for class we use Pascal Convention
2) for function / method we use camel Case Convention.

Package:-

- Package is group of similar type of class, interface and sub-package.
- Two Type:- Build-in pacake {io, util, etc}
  :- User-defined package.

• User Defined Package.

// जर मध्ये आपल्या class import karto.

// new class sathi new object create करायला लागतो

// जे access karayche ahe te public madhe pahije.

~~जर मध्ये आपण Hero p~~

for e.g

file1.java

```
package Hero;
public class A-file-1 {
    public void show()
    { sout("Hi@!");
    }
}
public class A-file-2 {
    public void show() {
        sout("# →");
    }
}
```

file 2.java

```
import Hero.*;  ——→ Hero package
public class file 2 {
    public static void main (String[] args)
    {
        A-file-1 obj = new A-file_1();
        obj.show();
        A-file-2 objt = new A-file_2();
        objt.show();
    }}
```

या मध्ये आपण Hero package import करून त्या package मधली सगळे class access करू शकतो (जर public आसतील तर). Hero package import करायला खालील line starting ला write करायची

* mean's all class.

```
import Hero.*;
```

आता आपण class normally operate करतो तसे करायचे, आही object तयार करून methods/property access करतो

```
A-file-1 obj = new A-File-1();
    obj.show();
```

//note:- जी file.java include (ahe package madhe त्याच्या name चा kahi संबंध नाही package include (import) केल्या नंतर class access कर fakt, file-name चा jast vichar karu nko.
pan lakshat ter, jayt tula run करायचं आहे (main/current.java) file ती नीट कर.

```
mean's file-2
    public file-2 {
        public ——
    }
}
```


Package import na karta pn apan package use karu shakto (import Hero.* he na karta)

Ya method la fully-qualified name mahntat

for e.g
file.1 ⇒

```
package Hero;
public class shows {
    public void show()
    {
        sout("Here");
    }
}
```

file.2 ⇒

```
public class file2 {
    public static void ——
    {
        Hero.shows obj = new Hero.show();
        obj.show();
    }
}
```

या मध्ये बाकी काही नाही fakt import न करता use करू शकतो

```
package_name.class_name obj =
    new package_name.class_name();
```

Java is statically type ——— determine at compile time

Page no.: 4
Date: / /

17/07/23

* Anatomy of java (Anatomy mahnje structure)

→

1. Documentation Section
2. Package statement
3. Import statement
4. Interface statement
5. Class Defination

optional.

6. main method class ——— essential

18/07/23

* Literal's :- Literal's mahnje constant value

```
int a=12;
```

Here '12' is Literal.

* boolean k = sc.hasNextInt();

k = he direct print karu shakt.
hasNextInteger ahe ka check kart

* String str = sc.next(); ← only read one word

String str2 = sc.nextLine();

↑ read full line

Page no.: 5
Date: / /

Type of operator :-

- Arithmatic → +, -, *, /, %.
- Logical → &&, ||, !
- Relational → <, >, >=, <=, !=, ==
- unary → ++, --, !, - (unary), + (unary)
- Assignment → =, +=, -=, *=, /=, %=
- Ternary → condition? true : false
- Bitwise → &, |, ^, ~
- shift operator → >>, <<, >>> (unsigned right shift)

Here,

- Bitwise → & (Bitwise AND)
  → | (Bitwise OR)
  → ^ (Bitwise EX-OR)
  → ~ (Bitwise NOT)

- shift operator → 1) >> (या मध्ये MSB la ती value add कराय्चची की original मध्ये आहे (negative का negative (1))

for e.g

(Q) -10 >> 2 = -3

-10 ⟹ 10 = 0000 1010
-10 = 1111 0110 (2's complement)
>>2 1111 1101 (2's complement)
0000 0011
= -3

(Q) 10 >> 2 = 2

10 = 0000 1010
>>2 = 0000 0010

2) << left shift operation मध्ये MLB मध्ये only '0' add करायचं

3) >>> (unsigned right shift)
positive value sathi '>>' sarkh kaam karat.
Negative value sati compicated आहे

20/07/23

* Associativity of operator :-
(Associative mahnje kont operator pahill work karnar.

Java doesn't use BODMAS rule, java work on associativity and procedence.

/* Tip- Associativity madhe jr '*' and '/' ekator asel tr divide adhi nahi hongr; associ-ativity pramane honar.
&
procedence means sagle level-wise astat.
for eg.

Procedence	operator	Associativity
14	( ) [ ] . ->	L to R
13	! ~ - ++ -- & (type) sizeof	R to L
12	* / %	L to R
11	+ -	L to R
10	<< >>	L to R
9	<<= >>=	L to R
8	== !=	L to R
7	&	L to R
6	^	L to R
5	|	L to R

**

IMP

4	&&	L to R
3	||	L to R
2	? :	R to L
1	= , +=, *=, etc.	R to L
lowest	,	L to R

// BODMAS
Bracket
Order of power/root
Division
Multiplication
Addition
Substraction
// आहेत procedence मध्ये Associativity.

** Resulting Data Type in expression.
(expression = series of operator, variable and methods.

R = b + s → int
R = s + i → int
R = l + f → float
R = i + F → float
R = l + d → double
R = c + i → int
R = c + s → int
R = f + d → double

b-byte
s-short
i-int
f-float
d-double
l-long
c-char

/* या मध्ये कोणत्या type ला assign करतोय हे imp नाही; कसं करतोय हे imp आहे */

for e.g

float a = 714 * 912 → a = 4
* हे int मध्ये करतोय हे लक्षात ठेव
⇒ 714 * 912 = 7.8T5 आहे (actually)

for eg.

(float) 7/4 * 9/4	7/4 * 9/4	7/4 * (float) 9/4
1.75 * 9/4	1 * 9/4	1 * (float) 9/4
15.75/4	9/4	(float) 9/4
= 3.9375	= 2	= 2.25

→ Difference व लक्षात घेऊन expression तयार करायचे बघा!

7f/4*9/4 हे बरोबर केलं तरी चालेल

21/07/23

String :- In java string is basically an object of string class that represent sequence of 'char value'

i.e.

```
char [] ch = {'s','H','A','N','T','O'};
string man= new string(ch);
        or
string man = new string("SHANTU");
        or
String man = "SHANTU";
```

Ekda string define kel ki parat ti change karu shakto pn purn stringch change karavi lagnar, mean's 1,2 char change nahi karu shakat.

i.e

```
String name = "shantu";
name.charAt(0) = 'P';   X
name = "Phantu";        ✓
sout(name.charAt(0));
```

→ string are immutable.

argument are actual parameter is copy.

* printf("%d", 5);

C language madhil printf pn we karu shakto.

- printformat("%d", 5); → same printf.

* string method:

// original string change hott nahi fakt modified value return hote.

name = "Hello_shantu"
0 1 2 3 4 5 6 7 8 9 10

- name.length();
- name.toUpperCase();
- name.toLowerCase();
- name.substring(4); // 4 index pasun end paryant ⇓ o-shantu select hornar.
- name.trim(); // fakt starting चा space cleare karnar.
- name.substring(4,8); → Return a substring where start index is included & end index is excluded.
  ⇓ o-sh
- name.substring(6,12)
  ⇓
  Shantu
- name.replace('l','k') → Replace 'k' with every 'l' char in string (case sensitive).
  ⇓ Hekko-shantu
- name.replace("ello","kyky");
  ⇓
  Hkyky-Shantu

// first index = target
// second index = replacement

- name.startsWith("He") → return true/false according to index start.
  ⇓
  true

- name.endsWith("k"); ⟶ // " " is mandatory
⇓
false.

- name.charAt(3) ⟶ same as name[3]
⇓
'l'

- name.indexOf("sh") ⟶ Return first occurance of index(sh) in name(string)
⇓
6

- name.indexOf("sh", 6) ⟶ search from index 6 for sh and return original index number of sting
⇓
6

- name.indexOf("sh", 7) ⟶ if no matching found return -1.
⇓
-1

- name.lastIndexOf('l') ⟶ same index of sarkh pn last occurance return
अनुसार
- name.lastIndexOf("s", 4) ⟶ same as indexOF("ll", 2);

- name.equals("Hello") ⟶ return boolean value according to string [case sensitive]
⇓
~~false~~

- name.equalsIgnoreCase("HeLlo sHamTo") ⟶ same ahe fakt upper/lower case ani ignore karte
⇓
~~true~~

* Escape Sequence Character.
In, It.

"/"Hello/"" ⟶ "Hello"

#26 ARRAY

int[] arr = new int[5];
arr.length;

* for each loop.

for(int i : arr) ⟶ Run through every array element,
{
//code
}
// arr मधला प्रत्येक element i मध्ये store होत जाहतात (temporary)

char array print kartana.
i.e
char [] arr = {'S','H','A','N','T','O'}

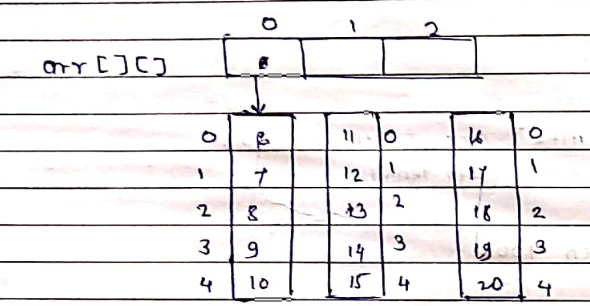
for (char i : arr)
{
sout(i); ✓ //(int i : arr) Kel tr integer (ascii) value print honar.
}

* ONLY ELEMENTS ARE AVAILABLE, COUNT NAHI *

#27 Multidimension Array:

2D

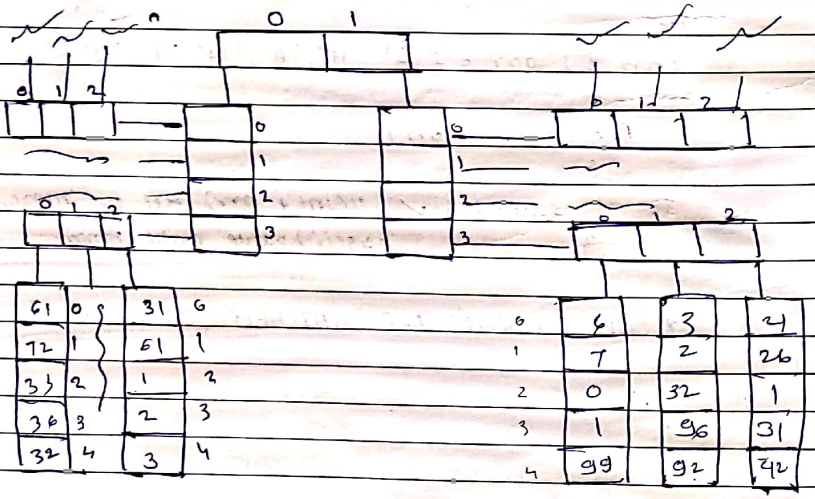
```
int [][] arr = new int [3][5];
                    ↙ row    ↘ column.
```



arr[0][2] = 8
arr[2][4] = 20.

4D

```
int [][][][] arr = new int [2][4][3][5];
```



arr[0][3][0][2] = 83;

arr[0][3][2][0] = 31
arr[1][3][1][3] = 96.

```
for(i;;){
  for(j;;){
    if(condition){
      break;}  → हे केलं की फक्त inner loop break होतो.

    //return;  → return केलं की 2 loop break होतात.
  }
}
```

#31 Method's

```
class Main{ - main(-){
    Main obj = new main();
    obj.show();
}
void show()
{
    sout("Hello");
}
}
```

// Jr static keyword use karayeha nasel tr fyach object ने method ला call करायचा

(static keyword.java)

** static :-

या मध्ये pn static same c++ sarkh आहेत

- static — 1) static function is also bound to class and not to any object. It can be called using the class name rather than an object of class.
  2) The static keyword is used for constant.

IMP → We cannot access non-static variable in static method.

variable or method that is same for every instance of class. The static keyword belong to class rather than an instance of class.

3) The user can apply static keyword with variable, method, block, nested class.

(a) Variable :- static int e=0;

(b) Method :- static void myself()
{
// code
}

(c) block :- हे class मध्ये use करायचे

static {
// code
}

// static block एकदा ch run होत (class assign होताना), constructor प्रत्येक new obj ला call होतो, pn static block जेव्हा class import करतं first time use करतो तेव्हा.

Next example मध्ये check kr (Clear)

(d) Nested class : A class can be declared static only if it is nested class. It does not require any reference of outer class.

e.g of (i) (ii) (iii) :-

```
class newclass {
    static int i = 99;
    static void show () {
        sout ("value = " + i); }
    static {
        sout ("static Block"); }
}
```

IMP → जेव्हा object pass करतो method, मध्ये तेव्हा reference pass होतो (address)


```
public class Main {
    main (String) {
        newclass.show(); → can call show method without creating object of 'newclass'.
```

O/p →

* Static Block.
// जेव्हा constructor काही ही त्यामध्ये आहेही static Block call होतो.

value = 99

** Nested class vs Innerclass google it.

#32 Method overloading in java.

आपण आहेही बघितलं ; jr आपण parameter pass kel method ला तर त्याची तिथं copy (call by value) बनते.

for eg.

```
void change (int a)
    { a = 99;
    }
void main (-) {
    int x=1;
    change (x);
    sout (x) ——→ O/p → 1
}}
```

पण जेव्हा Array Pass करतो त्या वेळी (Java मध्ये array cha object Pass करतो (reference) sarkh होतं.

(MethodOverloading.java)

for eg.

```
void change (int [] a)
{  a[0] = 99; }
void main(——) {
int [] x = {88,22,66,55}
 change (x);
 sout(x[0]);
}
```

O/P → 99

Baki method overloading same ahe.

Note :- Method overloading cannot be performed by changing the return type of method

```
int k(int a, int b) { }
void k (int a, int b) { }
```
✗ हे नाही चालणार

pn

```
int shantu(int a) { return 1; }
void shantu () { }
float shantu (float b) { return 0.0 }
```

#32 VarArgs - Variable Arguments.

```
sum (int a, int b )
sum(int a, int b, int c)
sum(int a, - - - )
```
हे असले करण्या पेक्षा आरंभ करण्या पेक्षा आपल्या कडे आणून Ek option आहे, varArgs चा

i.e

it is available as int[] arr

```
static int sum (int ... arr)
{
for(int i : arr) { sum += i }
```


** Setter & Getter

** #41 Constructor and constructor overloading.

```
class Hello {
    main(——) {
    Employ obj = new Employ();
    Employ obj2 = new Employ("Shantu", 462);
}
class Employ { int k; string name;
  public Employ()
     { this k = 0;
       this.name = "NA" }
  public Employ(String n, int id)
   { this.name = n;
     this.k = id;
   }
}
```

** #45 Inheritance :

Java does not support multiple inheritance. There can't be many superclass to one subclass

We use extends keyword for inherit

** Array of ~~class~~ object.

```
Main obj[] = new Main [2];
obj [0] = new Main();
obj [1] = new Main();
```

#46 Constructor in inheritance.
(ConstructorInInheritance.java).

Suppose there are more the one constructor in Base class (with different parameter

obviously) and you have to access specific constructor. What will you do.

→ sol^n → super – keyword.

if Base class have 2 param constructor and you want to call that constructor

You have to use super(param1, param2) in your derived class constructor.

*** (REMEMBER WE HAVE TO USE SUPER KEYWORD IN FIRST LINE OF OTHER CONSTRUCTOR)***

for eg.

```
Base () {sout ("Base");}
Base (int k) {sout ("Base 2");}
Base (float j) {sout ("Bas3");}

class derived extends Base
    derived (int a) { super (2); sout —}
    derived (float b) {super (1.0); sout —}
```

#47 this.

1) this can be used to refer current class instance variable.
2) this can be used to invoke current class method (implicity).
3) this() can be used to invoke current class constructor
4) this can be passed as argument in method call.
5) this can be passed as argument in constructor call.
6) this can be used to return the current class instance from method.

①
```
print (int k, String name) {
    this.k = k;
    this.name = name; }
```

②
```
void show() {
    this.print (2, "sh"); }
```

③
```
Main () { sout ("hey"); }
main (int x) { this(); sout (x); }
```

in main fun^t.

```
Main obj = new Main (6);
```

o/p →
hey
6

④
```
class Maint { void m (Maint obj) {
    sout ("Hero"); }
    void p() { m (this); }
```

In main fun^t.

```
Maint obj = new Maint();
obj.p();
```

⑤ This can be useful if we have to use one object in multiple classes.

⑥
```
class A {
    A getAC() {
        return this; }
    void msg () {sout ("Hey You"); }

class Test {
    main (—) {
        new A().getAC().msg();
    }
}
```

#48.

@override keyword.

@override चा किंवा pn program चालतो pn hi tumhala bing काही problem च execute करायचे असेल (without any mistake (at compile time)).

#49 Dynamic Method dispath.

also known as runtime polymorphism

1) When overridden method is called through a superclass reference, java determine which version (superclass/subclass) of that method is to be executed based on object being referred to at the time the call occurs.

2) Thus this determination done at run time. At run time, it depend on type of object being referred to, that determine which version of overridden method will be executed.

#49

* Dynamic Method Dispath :-
(Run Time Polymorphism)

- When an overridden method is called through a superclass reference, java determine which version (superclass / subclass) of that method is to be executed ~~before~~ based upon the type of object being referred to at time call occurs (running time)
- At run-time, it depend on the type of object being referred to (not type of reference variable) that determine which version of overridden method will be executed.

- class implement interface
- class extends class

#55 Abstract vs Interface

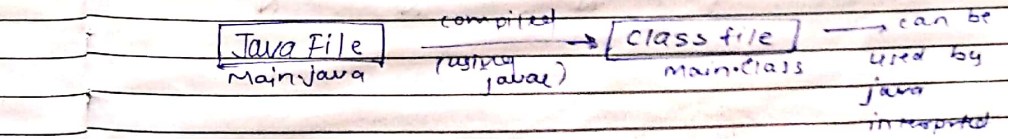
+ 1) Interface properties are final
+ 2) Fields in interface are public, static, final
+ 3) overriding methods should be in public
+ 4) Non-abstract class must override all methods declared in super interface.
+ 5) Abstract class are not forced to override all methods from their super interface. The first concrete (non-abstract) class in inheritance tree must override all methods
+ 6) Interface can be nested inside class and also can be nested inside another interface
+ 7) Method's in interface cannot static and final.

#68

Interpreted vs compiler.

* One statement at a time	* Entire program at a time.
* Interpreted needed everytime.	* Once compiled it is not needed.
* Partial execution if error occurred.	* No execution if error occured
* Easy for programer.	* Not as easy as interpreted

* Java is hybrid language → it has both compiled an as well as interpreted



- JVM can be used to interpreted byte code.
- Hence ~~this~~ java is platform independent.

#70 Multithreading :-

There are two way to create thread in java.

1) By extending thread class.
2) By implementing runnable interface.

```
My thread 1 { extends Thread {
    public void run (){
        //code
    }
}
Mythread 2 extends Thread {
    public void run (){
        //code
    }
}
Main {
    P.S.V.M {
    Mythread 1 obj = new my thread 1 ();
    Mythread 2 obj2 = new My thread 2 ();
    obj.start ();
    obj2.start ();  } }
```

run ()
start ()

*** Access modifiers -

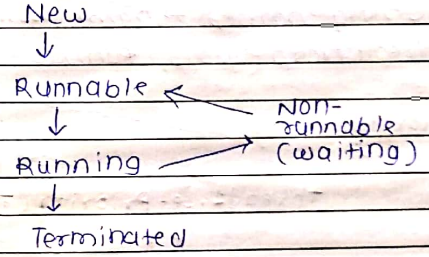
	class	package	subclass	world
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Defaut	Y	Y	N	N
private	Y	N	N	N

#71 Multithreading using runnable interface.

```
class thread 1 { implements Runnable {
    public void run (){
        //code } }
class main {
    P.S.V.M {
    thread 1 obj = new thread 1 ();
    Thread obj2 = new Thread (obj);
    }
}
```

Thread class tayar karaylach lagto interface implement kelela class thread: nahi ~~to~~ run method karu shakto pn te mg multithreading nahi honar

#72 JAVA THREAD LIFE CYCLE



New :- Instance of thread created which is not yet started by invoking start ().

Runnable: After invocation of start () & before it is selected to be run schedular.

Running: After thread schedular has selected it.

Non-runnable: Thread alive, not able to run.

#73 Constructor from Threads.

thread. getId();   thread.
thread getName();

class thread extends thread{

thread. (){
//code
}
thread(string name){
super(name); → // ह name set करतात
}
}

Constructor :-
1) Thread()
2) Thread(Runnable target) → हे Runnable interface मधला use होतो
3) Thread(Runnable target, string name)
(ज्या मध्ये super use karavachi garaj nah.)
4) Thread(string name) → name assingn करतात
5) thread(ThreadGroup group, Runnable target)

#74 Thread's priority

min = 1   thread1.set Priority (Thread 7);
norm = 5   or
max = 10
thread1.set Priority(Thread. MIN-PRIORITY);

(Thread. _) 1] MAX. PRIORITY
2] NORM. PRIORITY
3] MIN. PRIORITY

syntax:-
threadname. setPriority (Thread. MIN/MAX/NORM -PRIORITY);
or
thread-name.set Priority (1...10);

obj ~~thread class~~ ~~obj. thread main method~~

Note :- OS are responsible of thread priority so, don't depent mostly on java thread priorities.

#75 Threads method (need try/catch for exception)

(1) obj.join(); → ज्या thread nantr baki che thread executed honar (baki che means tyachya khalchi lines.) main method cha thread pn yachi (obj) thread nantr execute honar.
// wait for this thread to die

(2) Thread. sleep(millis: 661) → हा method thread ला sleep मध्ये Taknar, dilelya amount paryant
// ही method run() method मध्ये use केली जाऊ शकते
// main method ची pn thread sleep ला taku shakto p.s.v.m madhe.

(3) Thread. currentThread() → show currently running thread.

(4) obj. getPriority() → show priority

(5) obj. toString() → return [thread_id, thread-name, priority, thread-group]

#78 Errors and Exception's in java.
3 Type's :- syntax error - declaration error, etc.
logical error - // not satisfied with result.
Runtime error

#80
Exception in java :-
An exception in java is event that occur during execution of program, that disrupts the normal flow of instruction.

/* Tip → e.printStackTrace() → get line no. where exception occured

Exception class in java.

We can override method's of these class.

normally jevha apan catch block madhe sout(e) karto tevha to call e.toString() la jato.

```
class temp {
    @Override
    public String toString(){
        return super.toString() + "with me";
    }
    @Override
    public String getmessage(){
        return "This is shantu";
    }
public class main{
    p.s.v.m{

    try {
        throw new temp();
    }catch(Exception e){
        sout(e);
        sout(e.getmessage());
    }
}}
```

/* e.printStackTrace(). he method program chya last la jaun execute hote.

Description of error handling.

1) try:- This keyword is used specify a block where we should place an exception code. It mean we can't use try block alone. The try block must be followed by either catch or finally.

2) catch:- This keyword is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. ~~the try block~~ It can be followed by finally block later.

3) finally:- This keyword is used to execute the necessary code of program. It executed whether an exception is handeled or not.

4) throw:- This keyword use to throw exception

5) throws:- This key is used to declare exception- It specifies that there may occur an except--io in method. It is always used with method signature.

ADVANCED JAVA

* Collection framework:- A collection represent a group of object; java collection provide classes and interface for us to able to write code quickly and efficiently.

Commonly used collections in java;

→ ArrayList → For variable size collection
Set → For distinct collection (same parat yenar nahi)
Stack → For A LIFO data structure
Hashmap → for storing key value pairs.

#91 ArrayList. import java.util.ArrayList;

syntax:-

ArrayList <Integer> arr = new ArrayList<>();
or
ArrayList<>(20); ↑ custom capacity.

// ArrayList is not synchronized; 50% Double.
// Vector is synchronized; 100% double

method's:-

→ 1) add (element);
→ 2) add (index:, element:);
→ 3) addAll (collection); end la passed obj add karnar
→ 4) addAll (index:, collection); → specifix index la
→ 5) remove (index); → index varcha remove karnar.
→ 6) remove ((Integer) element); → Index milel nasnna tr:
or
remove (arr.indexOf(element));
→ 7) indexOf (element) → return index of element.
↓
-1 in case of element not found
/* 0,1,2,3 --- */

8) removeAll (obj); → pass kelela obj remove karto. → every element remove karnar. (single element pn);
i.e arr.removeAll(arr2);

best

9) get(index) → return element at index.

10) arr.clear();

*** 8) removeAll (obj); pass kelelya collection madhe jeje element ahet te remove karun taknar.

→ 7.i) lastIndexOf(element) → same as (7)

→ 11) trimToSize() → trim capacity of ArrayList instance to be list's current size.

→ 12) arr.set(index, element) → change index (element) with passed element(new);.

* LinkedList in java.

java.util.LinkedList;

LinkedList <O> obj = new LinkedList <> ();

* nearly every method from ArrayList is present in LinkedList */

* some New methods:-

- peek()
- pop()
- push(element)
- addLast(element)
- addFirst(element)
- removeFirst()
- removeLast()
- removeFirstOccurrence(O)
- removeLastOccurrence(O)

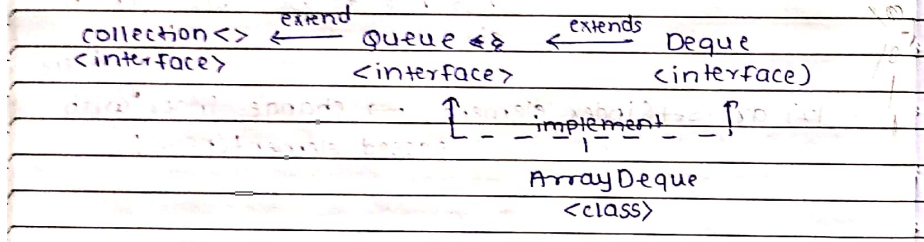
* Array Dequeue.

ADT → DArray Deque class provide constant-time performance for inserting and removing elements from both ends.

a) Resizable - The arraydeque class uses resizable array to store it's element

DADT- 1) Not synchronized :- By default it is not synchronized, which means multiple threads can access it simultaneously, leading to data corruption.

2) Limited Capacity :- Although it is uses resizable array, it still has limited capacity. Need to create new ArrayDeque when old one reach its max size.



* constructor:-

ArrayDeque () → default hold 16 element

ArrayDeque (num) → initial capacity 'num' element

* method's:- '—' is useful, special's.

→ add(E), addLast(E), offerLast(E) → ADD TO LAST OF LIST.

→ push(E), addFirst(E), offerFirst(E) → ''' first of list.

→ ✗ getFirst(), getLast() :- may throw error if empty exception.

↓ instead use

→ ✓ peekFirst(), peekLast() :- return null if empty.

→ ✗ remove(), ~~remove(E)~~, removeFirst(), removeLast()

:- retrive and remove element from list.

↓ instead use

→ ✓ ~~pop~~ pollFirst(), pollLast() :- return null if empty.

→ removeFirstOccurrence(E), removeLastOccurrence(E)

:- remove 'E' from list.

→ other method are common from ArrayList.

* Hashset.

Hashing in java is technique for mapping data to secret key, that can be used a unique identifier for data. It employ a function that generate those key from data. And output of this function (key) is known as Hash-value.

95

* Hashset

```
import java.util.HashSet;

HashSet<>obj = new HashSet<>();
```

constructor:-

1) HashSet(); // Defaut :- (16, 0.75f);
2) HashSet(int capacity); // Defaut (0.75f);
3) HashSet(int capacity, float load factor);

// load factor is measure of how full HashSet is allowed to get before it's capacity automatically increase.

4) HashSet(Collection<? extend E>c);

method's:-

boolean → add(E e); → Add if it is not present.
void → clear();
Object → clone();
boolean → contains(O o);
boolean → isEmpty();
Iterator<E> → iterator(); → return iterator over element in list.
boolean → remove(O o);
int → size()
Spliterator<E> → spliterator()
Object[] → toArray()

#96 Date and Time. (java.time..):

from 8: onward (JDK)
(Generally, java hold from 1 Jan 1970) but

** → Java assumes that 1900 is start year which means it calculate year passed since 1900.

~~System.currentmilli~~
System.currentTimeMillis() / 1000 / 3600 / 24 / 365
(60×60)
↓ sec ↓ hrs ↓ day ↓ year

#97 Date Class
(आजकी use नहीं होता।)

import java.util.Date*;

//Most of method are deorpted.

→ Date obj = new Date();

sout(obj): → return string of full detail
obj.getTime → return long millisecond.
.getDate → return only Date.
.get.Seconds → return current sec.
.get.Year() → return Year - 1900.
.set
↓
? .compareTo(.)

#98- Calendar class
~~#99~~ - ~~Gregorian Calendar~~ & Timezone
import java.util.Calender;

* //Note Calendar class is abstract class but you can get this class instance with Calender c = new calender.getInstance(); method.
Various method are their.

#99- Gregorian calender & timezone.

#98. Calendar c = new ~~Eto~~ Calendar.getInstance();
c.getTime();
c.get(Calendar.DATE);
c.get(Calendar.HOUR);
c.get(Calendar.HOUR_OF_DAY);
c.get(Calendar.Minute);

← import java.util.GregorianCalendar;

GregorianCalendar c = new GregorianCalendar();
c.isLeapYear(YYYY);

– Timezone.

Timezone.getAvailableIDs() [0---] array ans.

#100 - java.time. API.

there are various classes in these package.
//*Note:- need to create reference object (means without new keyword).

1) DayOfWeek
2) LocalDate
3) LocalTime
4) LocalDateTime
5) LocalMonth
6) Year.
7) etc- - ——

Syntax:-
LocalDate obj = localDate.now()

#101 Date Time formatter class

```
import java.time.format.DateTimeFormatter;

(i) LocalDateTime dt= LocalDateTime.(dt);
(ii) DateTimeFormatter obj = DateTimeFormatter.ofPattern(
        "dd-mm-yyyy");
(iii) String Date = dt.format(obj);
      sout(Date);
```

ofPattern("__");

D- day of year. i.e 189.
M- month of year. ie 7.
d- day of math ie 91
E- day of week ie E→Mon. EEEE-Monday
a- am/pm of day ie
h - clock hour of day (1-12)
K - " " " (1-24)
m - minute of hour
S - second of min.
V - timezone ID

#105

*ADVANCED JAVA - 2.

- Creating our own java Documentation.

#108

Annotations in java

@Override
@suppressWarnings("All");
@deprecated.
@functionalInterface. → only one interface

#109 ~~Anonym~~ Anonymus class and Lambda expre.

1) Anonymus class

- it enable you to make your code more concise (clear & short summary).
- They enable you to declare & instantiate a class at some time.
- They are like local class except they do not have name. , use them if you need to use local class only once.

/* interface चा reference घेऊन main method madhec override करायचे. */

eg

```
interface Og{
        method1();
        method2(); }
public class {
    P.S.V.M{
        Og obj = new Og(){
            @Override.
            .
            .
            .
        }; ← semicolon is imp.
        obj.method1();
```

2) Lambda expression.

- It provice clear and consile way to represent one method interface (functional Interface)

// Ex method asell interface main method madhe ghey shakto, jra anonymous class sarkh ahe.

¥1

syntack :

(a) Runnable obj = ()-> { - - -
- - };

@functionalInterface.

(b) interface AS // interface method.
void show(string name, int Roll);
{ - - }
}

P.S.V.M
~~A obj = ()-> {~~

P.S.V.M { // it can be done (int n, int r) also
A obj = (n,r)-> { // formal param.
sout(n+s+r); }

obj.show("shantu", 21); // actual param

#110 Generics.

ArrayList<Integer>

Uses - ① Type-safety
② Type casting is not required
③ compile time checking

class Gen <t1,t2> {
t1 ob1;
t2 ob2;
Gen(t1 o1, t2 o2) {
this.ob1 = o1;
this.ob2 = o2; }

public t1 ~~void~~ gett1() {
return ob1; }

public t2 gett2() {
return ob2; } }
public class Main {
P.S.V.M {
Gen<Integer, String> obj = new Gen(21, "Shantu");
sout(obj.gett1());

// object Tip. // to create array just use normal class array creating tip or use below:

*** var ~~ob~~ arr = new Gen[2];
arr[0] = new Gen<>(23, 79);
arr[1] = new Gen<>("shan", "tanu");
// for fast foreach loop.
*** for(var i : arr) {
sout(i.gett1 + i.gett2);
}

#111 File Handling

import java.io.*;
" java.io.IOException;

① File newfile = new File("name");
② FileWriter fwrite = new File("path");
① newfile.createNewFile();
② fwrite.write(" — ");
② fwrite.close();

Page no.:
Date: / /

③ Reading :-

```
Scanner sc = new Scanner (Path);
while (sc.hasNext()) {
 String l = sc.next();
 sout(l);
 }
sc.close();
```

④ Deleting :-

```
if (newfile.delete()) {
  sout(newfile.getName() + " is deleted ");
 }
else {
 sout("error"); }
```

x x x x

package Project.

- interface → ActionListner = actionPerfomed(ActionEvent e)
- frame.setLocationRelativeTo(null); → ◻
- // frame icon change.
- { ImageIcon img = new ImageIcon("PATH");
  frame.setIconImage(img.getImage())
- frame.setResizable(false) → frame resize nahi honar.
- { Font font = new Font("Time - ", Font.BOLD, size);
  label.setFont(font);
- { passwordfield.setEchoChar('*'); * ही ... replace honar passfield madhe.
  " " " ((char) 0); → show password.
- { Cursor c = new Cursor(CURSOR.HAND.CURSOR);
  button.setCursur(c);