JAVA

→ In the year 1996. James Gosling of Sun micro system has invented java pgming language
→ sun (stanford university n/w)
→ Initially they named it as oak pgming language. let they change to java pgming language.
→ the first project i.e., Green project by using sun micro system
→ Java support 3 extensions :
1). java (java source code extension)
2). class (java byte code extension (compile code extension)
3). jar (java archieve utility)

→ Java support 3 editions :
1) JSE (Java standard edition)
→ By using JSE we can develop standalone applications ← applets, awt, swings
2) JEE/J2E (Java Enterprise edition)
→ It is used for developing web application (enterprise ← servlets, JSP
3) JME (Java mobile edition)
→ By using this we can develop mobile applications

features of Java

1) simple & secure
→ Java is secured pgming lang. by avoiding pointers
there are 2 types of security:
1) implicit → automatically provided by the JVM by using security manager
2) Explicit → By using java.security package & JAAS (Java authorization & authentication services)

2) objected oriented
→ In java data is represented in the form of objects

3) platform independent (byte code)
→ compile in one os execute in any os

4) portable (source code)
→ java is a portable lang. java source code can run in any other os without making any changes in original source code

5) Robust
→ java is strongly typed pgming lang. it is good in two areas :
1) Exception handling 2) Memory allocation

6) Architecture Neutral :
→ we can run java pgm in any architecture (Hardware)

7) Distributed :

8) Multithreading

9) Dynamic

steps to run java pgm :

1) open an editor write the code & save the file with extension (.java)
2) How to compile : 1) go to console > javac filename.java
3) How to execute : 1) go to console > java classname
→ if the class is public then classname & filename should be same
→ if the class is not public then classname & filename can be different
→ javac *java is used to compile all java files at a time

IDE's for Java : MyNetBeans, vscode, Texteditors.

structure of java pgm

1) Documentation section
→ Java support 3 types of comments
1) Single line comment 2) Multiple line comment 3) Documentation comment
2) Import packages
→ package is a collection of similar type of classes & interfaces
→ package is a directory (folder) which consists of .class files
→ java support 2 types of packages :
1) Builtin package 2) User defined package
syntax for declaring a package : import packagename;
import packagename.classname;
import packagename.*;

→ 'java.lang' is the default package
3) → Every java pgm should start with class. Inside the class we can have main method

```
public class Example{
    public static void main(String[] args){
    }
```

→ public static void main(String[] args)
public → access modifier
static → java main method is static, we can call the static method directly without the help of object
void → returntype
main() → method name
args[] → from the console we read multiple values & save in an array
→ from the console whatever the value we are reading java compiler will considered as string datatype
→ Primitive to non-Primitive is boxing
→ Non-primitive to primitive is Unboxing

o/p methods in java

→ java support 3 o/p methods :-
1) print() 2) println() 3) printf() class. these class available in system
→ these methods are available inorder java.lang package. It is the default package.

syntax : system.out.println("text");

class :

→ class is a collection of data members & member functions
→ class is a template (or) blue print.
→ It is a logical representation of data
Java support 2 types of classes :-
1) Executable class 2) Non-Executable class.
↓ ↓
class with main method class with no method
→ class contain 5 elements :
1) variables 2) method 3) Constructor 4) instant blocks
5) static block

object :

→ It is an instance of a class
→ It is a physical presentation of data
→ It holds a value. by using new keyword we can allocate the memory for the object
→ object has 3 properties :-
i) state (it represents data of an object)
ii) Behaviour (represent behaviour of an object)
iii) Identity (name of an object)

syntax for creating obj for a class :-

classname objectReferenceVariable = new className();

Ex : Test t = new Test();
↓ ↓ ↓ ↓
classname obj operator/keyword → Constructor
t → obj

Test t;
t → Nothing

Ex : int i = 10,000;
system.out.printf("%,d", i); → o/p : 10,000

→ for the local variable we don't have default value

Ex : int i;
s.o.p.ln(i);

Identifiers :

→ The name given to the variables, methods, classes, interfaces are called identifiers.

Rules: A-Z, a-z, 0-9, _ $

→ java is not purely oop language
→ java doesn't support operator overloading, multiple inheritance

Rules
→ Java identifier is a combination of alphabets (A-Z), digits (0-9) & two symbols (_, $)
→ identifier name shouldn't start with digits
→ Java identifiers are case sensitive
→ Reserved words shouldn't be declared as identifiers.
→ Max length of the java identifier shouldn't be more than 15 characters.
→ predefined classes & interfaces can be declared as identifier

Ex: int String=10;
s.o.p(String); →10
int Runnable=20;
s.o.p(Runnable); →20
int Integer=30;
s.o.p(Integer); →30.

I/P stmts in Java
→ there are 2 ways to read i/p value from the user:
1) By using console method 2) By using scanner class

Scanner class
→ scanner class is available under java.util package. Under this class we have following methods:
1) nextInt() → converted to integer data type
2) nextFloat() → converted to float data type
3) next() → converted to string data type

Ex: import java.util.*;
public class Examples
{
psvm(String args[])
{
Scanner scan=new Scanner(System.in);
s.o.pln("enter 1st no:"); →10
int i=scan.nextInt();
s.o.pln("entered 1st no:"+i); →20
int j=scan.nextInt();
s.o.pln(i+j); →o/p: 30
}
}

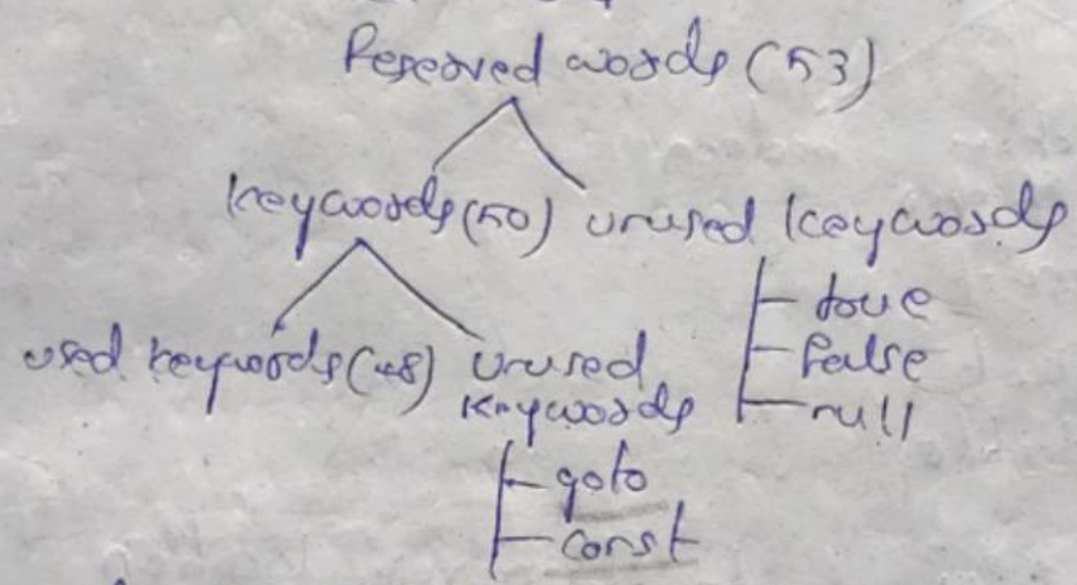
Ex: 2: Scanner scan=new Scanner(System.in);
s.o.pln("enter text");
String s=scan.nextLine();
s.o.pln(s);

ex:3 :- By using consoles()

```
s.o.pln("enter 1st no:");→10
string s1=system.console().readLine();
int i=Integer.parseInt(s1);
s.o.pln("enter 2nd no:")→20
string s2=system.console().readLine();
int j=Integer.parseInt(s2);
int sum=i+j;
sopln(sum);→30
```

Reserved words:-

Some java identifiers have predefined functionality & meaning are called Reserved words



→ keywords for Datatype:-
1) byte 2) short 3) int 4) long 5) float 6) double 7) char 8) boolean

→ keywords for control flow:-
9) if 10) else 11) switch 12) case 13) default 14) for 15) do 16) while 17) break 18) continue 19) return

→ keywords for modifiers:-
20) public 21) private 22) protected 23) static 24) abstract 25) final 26) synchronized 27) native 28) strictfp 29) volatile 30) transient

→ keywords for exception handling:-
31) try 32) catch 33) finally 34) throw 35) throws 36) assert

→ keywords for class relative
37) class, 38) package 39) import 40) extends 41) implements 42) interface

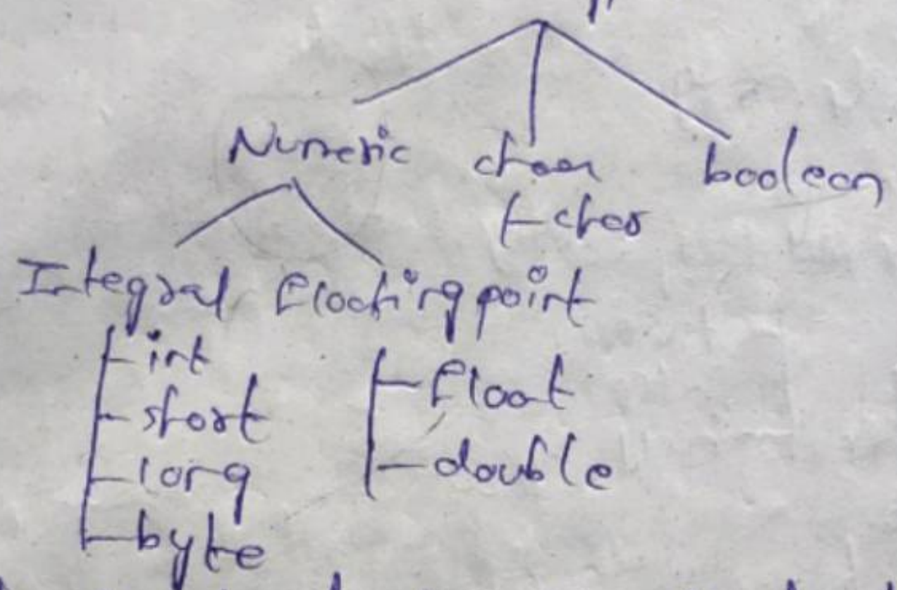
→ keywords for object related:-
43) new 44) super 45) this 46) instanceof

→ keyword related void:-
47) void 48) enum

Complete Syntax for main method:
→ public final/strictfp/synchronized static void main (String[] args)

Basics of Datatypes:
→ It specifies type & size of the data.
→ Java is considered as strongly typed language
→ Java supports two datatypes:
1) primitive data type 2) Non-Primitive Datatype.

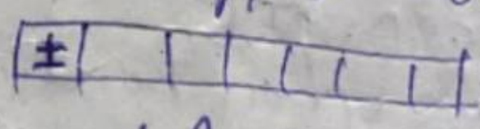
Primitive data type
→ In primitive datatype, we store primitive values



→ Data is stored directly inside the memory location
→ Memory is allocated at Compile time
→ Data is stored in stack memory

Byte:
size = 1
max value = 127
min value = -128
Range = -128 to 127
→ -2^7 to -2^{7-1}

→ Except boolean & char all datatypes are considered as signed (+ & - values)



→ positive values will be represented directly in memory as bits
→ negative values will be represented in 2's complement

Ex: byte b=10;
byte b1=130 → error. Here is chance of loosing
byte b2="java" → incompatibility type error
byte b3=true; → error

short:
size = 2 bytes, 16 bits
Range = -32768 to 32767
maxvalue = 32767
minvalue = -32768
→ short datatype is suitable for 16 bit microprocessors like "8086"

integer:
size = 4 bytes = 32 bits
Range = -2^{31} to $2^{31}-1$
= -2147483648 to 2147483647
minvalue = -2147483648
maxvalue = 2147483647
→ It is the default datatype for integral value type

long:
size = 8 bytes = 64 bits
Range = -2^{63} to $2^{63}-1$ ⇒
ex: long l = 100 → int
= 100326789765

Boolean
size: not applicable it depends upon virtual machine
→ Range: not applicable, allowed values are true & false
ex: boolean b = 0; → int datatype (error: incompatibility type error)
ex: if(1) {
} else
} → error.

char:
size = 2 bytes (unicode system)

Datatype	size	range	corresponding wrapping class	default value
1) Byte	1 byte	-128 to 127	~~Byte~~	0
2) short	2 bytes	-32768 to 32767	short	0
3) int	4 bytes	-2147483648 to 2147483647	Int	0
4) long	8 bytes	$-3.4e^{38}$ to $3.4e^{38}$	Long	0.0
5) float	4 bytes	-2147483648 to 2147483647	float	0.0
6) double	8 bytes	-1.7e308 to 1.7e308	Double	false
7) boolean	NA	NA	Boolean	0 (represent blank space)
8) char	2 bytes	0 to 65535	character	

Literal :
→ A constant value assigned to a variable is called literal

Integer literals :
→ Integer literals can be represented in four forms :-

1) decimal literals :
→ allowed digits :- 0-9 Ex : int i = 10; valid
int j = 010; - not valid (octal)

2) octal literals
→ allowed digits : 0-7 Ex : int x = 010; → 8
→ prefixed with '0' 1x8^1 + 0x8^0 → 8+0

3) Hexa decimal literals
→ allowed digits : 0-9, (A-F) → 10-15) Ex : int x = 0X10; → 16
→ prefixed with : '0X' Ex : int x = 0XFACE;
→ In this case it is not case sensitive.
Ex : char ch = 0XFACE;

Types of Variables
→ Java support 3 types of variables :-
1) instance var. 2) local var. 3) static var.

Instance Variable (object level variables)
→ the value of the variable is different from one object to another object.
→ for every object a separate copy of instance variable is created
→ Instance variable is stored in heap
→ It is also called object level variables
→ these variables are created at the time object creation & destroyed at the time of object destruction.
→ It is declared within the class above all the methods, Constructor, blocks
→ these variable can be access directly in instance area with the help of object in static area.
→ here this variable we have default value depending on datatype.

Ex : class Example {
int i = 10; // instance variable.
public s v main ( ) {
Example obj = new Example(); } → static area
s.o.println(obj.i);
obj.fun();
}
void fun() {
s.o.println(i); } instance area
}

called literal

forms :-

alid (octal)

= 0x10; → 16

are object

tance variable

ject creation &

(1 the methods,

stance area &

depending on

static variable :- (class level variables)

→ the value of the variable is not different from one obj to another object

→ for total class one copy of the static variable is created

→ It can be given reference each & every object

→ this variable is created at the time of class loading

→ It is destroyed at the time of unloading the class

→ scope of static variable is scope of class

→ static variable is called directly inside static & instance area.

→ we can access static variable by using classname / obj.ref.var.

Ex: public class Example {

```
static int i=10;
p s v m (string args[]){
    sopln(i); (sopln(Example.i);
    Example obj=new Example();
    sopln(obj.i); → 10
}
void f1(){
    sopln(i); → 10
}
```

Ex: int x=10;

```
static int y=20;
p s v m ( ){
Example obj=new Example();
 obj1.x=888;
 obj1.y=999;
sopln(obj2.x," ", obj2.y); → 888 999
 }
```

→ the variable which is declared inside the method / constructor / block is called temporary / automatic / local / stack variable. only final keyword is applicable for local var.

→ local variables are created at the time of block execution & destroyed once we leave the block. They are called as method / block level variables

* → increment & decrement operators should be applied to variables not to the constant values & boolean data types &

	final keyword
sop(++10); → error	final int x=10;
x=5 sop(++(++x))	sop(++true); ((sop(++x); ((
6	
++6 → error	

operators:

→ operator is a symbol which is used to perform operations on the operands

→ java support different types of operators:

1) Increment & decrement operators:
2) Arithmetic operators
3) string concatination operator
4) Relational operator
5) Equality operator
6) logical operators
7) Instance of operator
8) Bitwise operators
9) short circuit operators
10) type cast operators (())
11) new operator
12) Assignment operator
13) conditional operator (?:)
14) subscription operator ([])

1) Increment & decrement operators:

→ these operators are example for unary operators

a) Increment:

a) preincrement: y = ++x; ⇒ x = x+1, y = x

b) post increment: y = x++; ⇒ y = x, y = x+1

2) Decrement:

a) predecrement: y = --x ⇒ x = x-1, y = x

b) postdecrement: y = x-- ⇒ y = x, x = x-1

2) Arithmetic operators

→ these operators are used to perform calculations +, -, *, %, /,

→ byte + byte = int
→ char + int = int
→ char + char = int
→ byte + short = int
→ byte + int = int
→ byte + char = int

→ int + long = long
→ float + double = double
→ long + long = long
→ long + float = float
→ string + int = string

ex: 'a' + 1 → 98
'a' + 'b' → 97 + 98
'a' + 10.5 → 97 + 10.5
'a' + 0.5 =

→ By using path environment variable we can set the perticular path.

operations

string concatination operator

```
string a="ida";
int b=10, c=20, d=30;
sopln(a+b+c+d); ida102030
sopln(b+c+d+a); 60ida
```

relational operators

→ these operators can be applied to the the primitive types except object datatype, boolean datatype values

→ these operators are used for comparision.

Ex: sopln(true > false) → error

sopln(10>20>30>40) → error

↓

we can't perform nested relational operator

→ Relational & logical operators will return boolean value

equality operators:

→ these operators can be applied to all primitive data types including boolean data type.

ex. sopln(10==10.0) → true

sopln('a'==97) → true

sopln('a'!='b') → true

instanceof operator:

→ we can use this operator to check whether given object is of type or not

→ Ex: class Complex

```
    psvm(){
        Complex ob1 = new Complex();
        sopln(ob1 instanceof Complex); → true
        Thread th = new Thread();
        sopln(th instanceof Thread); → true
        sopln(ob1 instanceof Thread); → false
```

Bitwise operators

→ these operators perform operations on the bits

1) Bitwise AND (&): if both arguments are true then the result will be true

sopln(4&5) → 0100
0101
100

A=4

2) Bitwise OR (|): if atleast one argument is true then result will be true

3) X-OR (^): If both arguments are different result will be true

4) Bitwise Complement (~):

Ex: soplm(~4) → -(x+1) → -(4+1) → -5

soplr(~true) → error (∵ this operator can't be applicable for boolean datatype)

→ Boolean Complement operator (!) is applicable for only boolean data type.

Ex: !true → false !14 → error

short circuit operators (&&, ||):

→ these operators are exactly same as bitwise operators.

Type casting operator ((c)):

→ By using this operator we can convert one datatype to another datatype. It is 2 types:-

1) Implicit (widening / upcasting): Compiler is responsible for this type casting. Converting lower datatype to higher datatype.

→ byte → short → int → float → double

Ex: int x = 'a';

soplr(x) → 97.

2) Explicit (Narrowing / downcasting): programmer is responsible for converting higher to lower datatype.

a) C++ style: it will convert one datatype to another DT.

Ex int x = 150;

byte b = (byte)x;

b) parse style: It used to convert only string datatype to other datatypes.

Ex: int i = Integer.parseInt(rone);

wrapper class datatype

new operator: this operator is used for creating object of a class

subscription operator ([]): this operator is used for declaring & constructing the arrays

Conditional operator (?:) :- It is alternate to if-else

Syntax: var = condition ? T : res1 : var2

F

Ex: int result = a>b ? a : b;

→ By using printStream class we can display o/p on the webpage.

```
Scanner scan=new Scanner(System.in);
System.out.println("enter 1st no");
int a= scan.nextInt();
System.out.println("enter 2nd no");
int b= scan.nextInt();
```

* in & out variable belongs to printstream class

→ path:- It specifies the java compiler & JVM
→ when we compile java pgm we get .class files. these we put these files in one folder & By using classpath we can set the path for .class files.
→ our java pgm's is in Tomcat web server. By using HOME we can specify the web server.
→ path, classpath, HOME are the environment variables.

```
int result = a>b? a:b;
s.o.println(result);
```

Assignment operator : By using this operator we can assign a value to the variable.

a) simple Assignment operator (=)
b) chaining Assignment operator (a=b=c=30)
c) Compound assignment operator (+=, -=, etc)
→ we can mix other operators with assignment operator

Control flow statements :
→ By using this statements we control the execution of java pgm
→ It is 3 types :
1) Conditional statements → Types → 1) if
2) switch

2) Iterative statements / loops :- → 1) for
1) simple for
2) for each
3) labelled for
4) infinite
2) while
3) do while
4) assert

3) Jump (transfer statements :- 1) Break
2) Continue
3) return
4) throws

1) if

```
if(true)
int x=10 → error ( : bcoz if. if condition true are stmt
```

if should not be declared]

→ To todle this by using curly braces

```
if(true){
int x=10
}
```

2) switch : from the multiple options excepting are return value

syntax:

```
switch(exp){
    case caselabel:
     stmts;
     break;
     - - -
    default:
     stmts;
}
```

→ Curly braces are madatory
→ Both case & default are optional
→ Every stmt should be return inside case/default.
→ Individual stmt is not allowed.

Ex:

```
switch(10){
    s.o.pln("hi");→error
}
```

→ switch exp current can be a float & double. It can be byte, short, int, char, string
→ Every caselabel should be compile time constant

Ex:

```
int x=11;
final int y=22;
    switch(x){
        case 11:
            s.o.pln("hi");
            break;
        case y: ——→ To todle this, we make y variable as final
            sopln("hii");
```

error bcoz variable can be change

```
            break;
    }
```

→ switch exp & case label can be expression

```
int x=30;
switch(10+20){
    case 10:
    case 10+10;
} case 10+20;
```

→ int x=10;
switch(x){
case 10: sopln("10");
case 20: sopln("20");
case 30: sopln("30"); break;
default:
sopln("Invalid");
}

→ case label must be within the range of argument provided datatype.

Ex; byte x=126;
switch(x){
case 125: sopln("10"); break;
case 126: sopln("20"); break; [-128 to 127]
case 127: sopln("30"); break;
case 128: sopln("40"); break; → error [out of range]
default:
sopln("invalid");
}

→ switch argument datatype & case label argument datatype should be same

Ex; String str="aaa";
switch(str){
case "aaa": sopln("hi");
case 'b': sopln("hii"); → error, we are giving characters
default:
sopln("hii");
}

Iterative stmts:

int i=10;
for(int j=1; j<=5; j++){
sopln("hi");
}
sopln(i+j); → error; j is not declared

→ we go with iterative stmts if we want to repeat the multiple times.

→ Java support 3 types of loops.
1) for loop, 2) while loop 3) do while loop

for loop: It is used when we know fixed no. of iterations in advance

→ java support 4 types of for loops :- 1) simple for loop 2) for each (enhanced for loop) 3) labelled for loop 4) Infinity for loop

simple for loop:

```
int i=0;
for(; i<=5; i++){ // we can write individual start in for loop
    sopln("hi");
    sopln("isa");
}
```

→ we can declare the multiple variables but it should be same datatype

for(int i=10, j=20) ✓ for(int i=10, double j=236) → error

Infinite for loop:

```
→ for(int i=0; ; i++){
      sopln("hi");
  }
```

```
for(int i=0; i<5; sopln("hii")){
    sopln("isa");
}
```
o/p: hii

```
for(int i=0; i<5; sopln("isa"), sopln("cse")){
    sopln("hi");
}
```
o/p: hi, isa, cse, hi, isa, cse, hi, isa, cse, hi, isa, cse, hi, isa, cse

for each loop: It is used when we want to read the element directly from a collection like arrays, objects, list, etc. syntax: for(datatype var : collectionname)

for(int i : "naresh")

→ we can use only one variable in s.o.pln().

ex: sopln(x)

→ other sol^ is soprintln("x=" + x + "y=" + y)

soprintf("%d %d", x, y)

→ 2 printf()
10 print()
11 println() } → available in java

→ if we have to use logical / Bitwise operators when we combining multiple conditions. In c lang we can use comma to write like this

ex: i<2 && j<3 && k<2 in java ex: i<2, j<3, k<4 in c lang

- length() → used in strings (method) → return type integer
- length → used in arrays (variable) → return type integer

Enhanced for loop

```
int arr = {11,12,13,14,15};
for(int i : arr){
    sopln(i);
}
```

simple for loop

```
for(int i=0; i<arr.length; i++){
    sopln(arr[i]);
}
```

labelled for loop :- Assigning the name for the for loop.

syntax :- labelname:
for loop

ex :-
```
outer:
for(int i=1; i<=5; i++){
inner:
for(int j=1; j<=5; j++){
    sopln(i + " " + j);
if (i==3 && j==3)
    break outer;
}
```

o/p : 11 21 31
12 22 32
13 23 33
14 24
15 25

Infinite for loop :- the for loop which don't have any condition is called Infinite for loop

ex :
```
for(;;){
    sopln("hi");
}
```

while loop :- this loop is used when we repeat the statements for multiple times.
→ this loop is used when we don't know exact no. of iterations
→ it is a type :-
1) simple while loop :-
syntax :-
```
init;
while(cond){
    statements;
    inc/dec;
}
```
printing tables from 1 to 10
```
int i=1, j=1;
while(i<=10){
    while(j<=10){
        sopln(i + "*" + j + "=" + i*j);
        j++;
    }
    i++;
}
```

2) Infinity while loop :-
syntax :-
```
init;
while(true){
    statements;
}
```

→ for loop & while loop are called entry control loop :-
→ do while loop is called exit control loop.

do while loop:

Syntax: init;
do {
stmts;
incldec;
} while (cond);

Jump / Transfer stmts

1) Break: It is used in two places:
→ inside switch stmt & inside the loop
→ when we are using break inside the loop it should follow the condition.

2) continue: It is only used in loops.

Ex: int sum=0, capacity=15;
for(int i=1; i<=100; i++){
sop(i);
sum=sum+i;
if (sum>=capacity) o/p: 1 2 3 4 5
break;
}

→ It will skip the current iteration & continue next iteration & it should follow the condition

Ex: for (int i=1; i<=20; i++){
if (i==7||(i>=13)) o/p: 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12
continue; 14, 15, 16, 17, 18, 19, 20
sopln(i);
}

Arrays:

→ Arrays are derived datatypes to store multiple values
→ Array is a collection of similar datatype elements
→ Arrays are indexed base & index is starts from 0
→ & size = length - 1
→ Arrays are 3 types: 1) single Dimension 2) Multi Dimension 3) jagged array

single Dimensional array

→ Array elements are arranged in either single row/column

Declaration of array:
Syntax: datatype arrayname[];
Ex: int arr[];

→ at the declaration time we should'nt specify the size

creation:

Syntax: datatype arrname[] = new datatype[size];

Ex: int arr[] = new int[5];

(or)

int arr[];

arr = new int[5];

→ at the time of creation size is mandatory

→ Byte, short, int, char datatype & double.

→ we can't create the size of negative value

exception: NegativeArraySizeException

→ range of array: 0 to length - 1

Declaration, Creation & Initialization in single line:

Syntax: datatype arrname[] = {values};

Ex: int arr[] = {1, 2, 3, 4};

→ ex: int arr[] = new int[10];

sopln(arr) → address of array

sopln(arr[0]) → 0 (1st element)

sopln(arr[1]) → 0

size = 10

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

→ Jvm is going to be filled with default value of integers.

1000 1012

→ | 1 | 2 | 3 | 4 |

0 1 2 3

→ starting address + index * size of datatype

1000 + 0 × 4 → 1000

0

→ 1000 + 3 × 4 → 1012 //

12

Ex: int arr[];

arr = new int[5];

sopln(arr); → 102456

sopln(arr[0]); → 0

arr[0] = 11;

sopln(arr[0]) → 11

for(int i: arr){

sopln(i); → o/p: 11 0 0 0 0

}

```
int arr[] = {1, 2, 3, 4};
double darr[] = {1.1, 1.2, 1.3};
char carr[] = {'a', 'b', 'c'};
for(char ch: carr){
    sopln(ch);
}
```

o/p: a b c

Multi Dimensional Array

→ int a[][], int[][]a; int [][]a; int[][]a;
int[]a[]; int []a[];
→ It is also called as array of arrays.
→ Elements are arranged in format of rows & Columns
→ At the time of creation 1st Dimension is mandatory & 2nd Dimension is optional.
ex: int[10][]x; int[][10]x, int[][]; ✓

```
int arr[][]={{1,2,3},{4,5,6},{7,8,9}};
for(int x=0;x<arr.length;x++){
    for(int c=0;c<arr[x].length;c++){
        sopln(arr[x][c]+" ");
    }
    sopln();
}
```

By using for each:

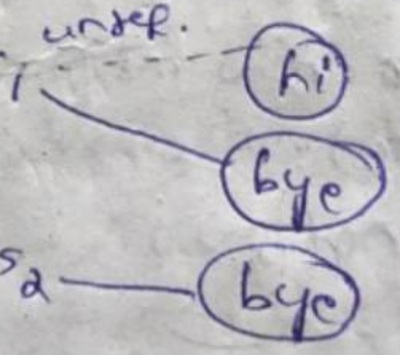
ex: int arr[][]={{1,2,3},{4,5,6},{7,8,9}};

```
for(int x[]:arr){
    for(int a:x){
        sopln(a+" ");
    }
}
```

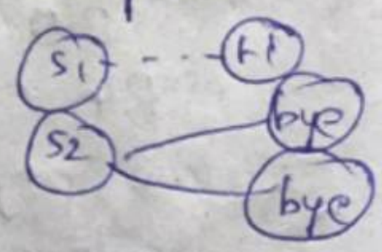
strings

→ string is a predefined class in java which is used to store group of characters
→ we require string object to store group of characters
→ there are two ways to create string object in java
1] By using string literals. Ex: String s1="iba";
2] By using new keyword. Ex: string s1=new String("iba");
→ In java strings are immutable means not modifyable

String s1 new String("hi")
s2=new string("bye")
String s2=new string("bye")



In case of string literals



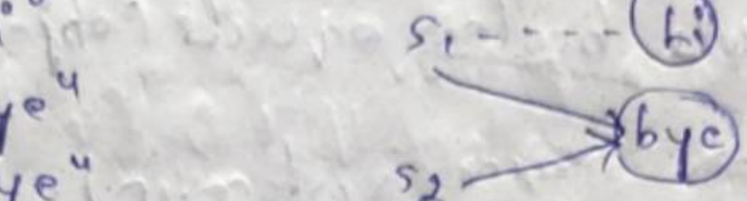
strings with new keyword:

→ It is stored in Heap memory
→ if it is stored in Heap memory it will never verify if content is available or not. Automatically if will create new object to store the content

By using string literals:

→ Group of characters are closed within the double quotes
→ stored in string constant pool
→ It will check whether the content is available or not.
If available it refers to existing object

string s1 = "hi"
s2 = "bye"
s2 = "bye"



Methods under string class

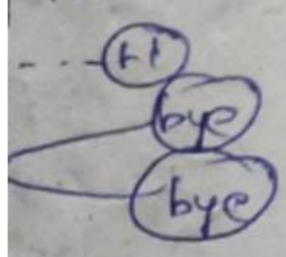
1) int length(): returns the no. of characters of given string
Ex: String s1 = new String("java");
soplr(s1.length()); → o/p: 4

2) char charAt(index): it returns the character at specified index
Ex: String s1 = new String("java");
soplr(s1.charAt(0)) → o/p: j
soplr(s1.charAt(23)) → StringIndexOutOfBound exception

3) Concat(String):
Ex: String s1 = new String("jaa");
s1 = s1.Contact("iaa");
String s2 = "Adv".concat("jaa");
soplr(s2); → Advjava
soplr(s1); → jaaiaa

4) int compareTo(String): it is used to compare by using unicode values based on the cases. this method returns integer value.
+ve ← s1 > s2
-ve ← s1 < s2
0 ← s1 == s2

Ex: String s1 = new String("abcd");
String s2 = new String("ABCD");
int a = s1.compareTo(s2)
if(a > 0)
soplln(s1 + "is greater than" + s2);



```
else if(i<0)
  sopln(s1+" is less than "+s2);
else
  sopln(s1+" is equals to "+s2);
}
```

5) boolean equals() :- This method will compare the content by using these cases.

syntax : boolean equals(object)

```
ex: Scanner scan = new Scanner(System.in);
    sopln("enter username");
    String un = scan.next();
    sopln("enter password");
    String pw = scan.next();
    if(un.equals("saidhar") && pw.equals("isa")){
        sopln("valid user");
    else:
        sopln("invalid user");
}
```

6) boolean startswith(String) } → checks whether the string
   boolean endswith(String) }
starts/ends with character or group of characters.

ex :- String s1 = new String("java programing");

```
sopln(s1.startswith('j'));→ T
sopln(s1.startswith("java"));→ T
sopln(s1.endswith("g"));→ T
sopln(s1.endswith("ming");→ T
```

7) int indexof(char) :- It returns index of 1st occurence of specified character

int lastindexof(char) :- It returns index of last occurence of specified character

ex: String s1 = new String("java.programing");

```
sopln(s1.indexof('a'));→ 1
sopln(s1.lastindexof('a'));→ 7
```

8) string replace (char oldchar new) :- replacing old characters with new characters

9) string substring (int index)

10) string substring (int index, int offset)

11) string toLowerCase()

12) string toUpperCase()

13) string trim()

ex:- String s1 = new String ("java programming");
soln(s1.replace('a','z')); → jzvz programming
soln(s1.substring(4)); → programming
soln(s1.substring(4,9)); → progr
soln(s1); → java programming.
soln(s1.length()); → 23
soln(s1.trim());
soln(s1.trim().length()); → 12

String buffer & builder

→ these are predefined string classes available under java.lang package

→ these strings are mutable & they can be created by only using "new" keyword

ex: StringBuffer sb = new StringBuffer ("java");
sb.append("programing");
sb.append("lang");
soln(sb) → javaprogramminglang
sb.insert(3,"oracle");
soln(sb) → javaoracleprogramminglang
sb.deleteCharAt(3);
soln(sb) → javaoracleprogramminglang

→ StringBuffer is synchronized (thread safe). It can be access by only one thread at a time

→ StringBuilder → it is introduced in java 1.5 version. It is not synchronized (not thread safe). It can be accessible by multiple threads at a time

Methods:

→ Methods are used to perform particular task
→ Inside the method we write logic / functionality

Syntax for creating a method:

```
access specified returntype methodname(parameters){
    statements;
}
```

→ It is divided into 2 parts:
1) Method Declaration & 2) Method Definition

1) Method Declaration / prototype / Header: returntype methodname;

2) Method Definition / Implementation:

```
def body / Implementation {  → sopln(null) → error
    logic;
}
```

cmd prompt:

```
javac B.java          class A{
                      }
java B                class B{
                      }
javac A.java          psvm(){
java A                }
Couldn't find main
    method
```

Eg: String s1 = null;
sopln(s1.length()); → NullPointerException

Constructor:

→ Constructor is a special type of method
→ Constructor means Constructs default values to the variables. based on the datatypes
(int = 0, float = 0.0, string = null, boolean = false)
→ classname & Constructor name should be same
→ constructor will not return any value not even void
→ once we create object for a class automatically the default constructor will be called
→ Automatically java compiler will create a default constructor at the time of object creation
→ for one object one constructor will be called

```
ex: public class Example {
        Example() {
            soln("In constructor");
        }
        void Example() {
            soln("In method");
        }
    psvm(String[] args) {
        Exmple obj = new Exmple(); → In constructor
        obj.Example(); → In method
    }
}
```

```
ex: public class Example {
        int id;
        String name;
        void display() {
            soln(id+" "+name)
        }
    psvm() {
        Exmple obj = new Exmple();
        obj.display() → 0 null
```

[:: the default constructor was created & assign the default values]

Types of Constructor:-

→ Java support 2 types of constructors:
1) Default Constructor 2) parameterized Constructor

1) **Default Constructor**:- the Constructor doesn't have any parameters
→ Java Compiler only generate default constructor at the time of object creation

2) **parameterized constructor**:- the Constructor which have parameters.

```
ex: public class Ex {
        Ex() {
            soln("0 const");
        }
        Ex(int i) {
            soln(" 1 const"+i);
        }
        Ex(String s) {
            soln(" 1 const"+s);
        }
```

```
psvm() {
    Ex obj = new Ex(); → 0 const
    Ex obj1 = new Ex("hi"); →
                1 const hi
    Ex obj2 = new Ex(2); →
                1 const 2
```

→ Constructor can't be abstract, final, synchronised

```
int id;
String name;
int age;
Example(int i, String name){
    id=i;
    name=name;
}
void display(){
    sopln(" id="+id+"name="+name+"age");
}
psvm(){
    Example obj=new Example(2,"Hi");
    obj.display(); → 2 hi 0
}
```

This keyword :

→ this keyword can be applicable for current class variables, methods, constructors

→ In constructor this will be first statement

Ex:

```
public class Ex{
    int a=10;
    int b=20;
    void add(int i, int j){
        sopln(a+b);
        sopln(i+j);
    }
psvm(){
    Example obj=new Ex();
    obj.add(11,22); → 33
}
```

local variables

```
void add(int a, int b){
    sopln(a+b);
    sopln(this.a+this.b);
}
```

→ 30

Converting local variables into instance & static variable by using this keyword

```
public class Ex{
    static int a; → static var
    int b; → instance var
    void values(int a, int b){
        this.a=a;
        this.b=b;
    }
    void display(){
        sopln(a+b);
    }
psvm(){
    Ex ob=new Ex();
    ob.values(5,10,20);
    ob.display();
}
```

Calling one method in other method :

```
ex: public class ex{
      void m1(){
            sopln("hii");
            this.m2(); (or) m2();
      }
      void m2(){
            sopln("hlo");
      }
      psvm(){
            ex ob=new ex();
            ob.m1(); → hii
                       hlo
      }
```

```
int id;
ex(){
  sopln("hii");
}
ex(int i){
  this();
  id=i;
}
psvm(){
  ex ob=new
```

```
public class ex{
      ex(){
            this(10);
            sopln("zero para");-③
      }
      ex(int i){
            this(10,20);
            sopln("one para"+i);-②
      }
      ex(int a, int b){
            sopln("two para"+i+j);-①
      }
      psvm(){
            ex obj=new ex(); → 3 para
                             → one para 10
                             → zero para
      }
```

Inheritance : (variables)

→ Extracting the properties & behaviour (method) from one class (parent/base/super) to another class (child/derived/sub)

→ Inheritance is also known as "is-a" relationship i.e Two classes belongs to one hierarchy

→ By using 'extends' keyword we are achieving inheritance

By using Types of inheritance.

1) single level
2) Multilevel
3) Hierarchical
} → Based on classes

4) Multiple
5) Hybrid (Hierarchical & Multiple)
} → Based on Interfaces.

→ Multiple Inheritance is not possible by using classes but by using Interfaces it is possible

Ex: public class CE{
class pclass{
int i=10;
pclass(){
sopln("pclass const");
}
class cclass extends pclass{
int j=10;
cclass(){
sopln("cclass const");
}
void m1(){
sopln("of i" "+i);
}
}
Revers public class Ex{
psvm(){
cclass obj=new cclass();
obj.m1();
}
}

→ Every class in java is a child class derived from object class (root class)
→ root for all the classes is "object class". It is available in java.lang package.

final ~~keyword~~ modifier

→ final ~~keyword~~ can be applied to variables, methods, & classes
→ final variable value can't be changed
→ final method can't be override
→ final class can't be inheritated
→ Every method in final class always "final" by default
→ Every variable present in final class need not be final
→ For final class we can't create any child class
→ For security purpose we use "final" modifier

Instance block

→ Block means set of stmts
→ Instance block is executed irrespective of any condition
→ In a class no. of instance blocks
→ Instance blocks are executed based on object creation

```
public class Ex {
    {
        sopln("IB-1");
    }
    {
        sopln("IB-2");
    }
    Ex(){
        sopln("constr");
    }
    psvm(){
        Ex obj=new Ex(); o/p: IB-1
    }                              IB-2
                                   constr
```

→ IB in plass & IB in cclass, if we inherit 1st plass IB & 2nd cclass IB will be executed

→ Instance variable & Block having same priority at that situation order is from top to bottom.

ex: public class Ex {
```
    int i=m1();-(or)
    ib m1(){
        sopln("hi");
        return 10;
    }
    {
    sopln("IB");
    }
    psvm(){
        Ex obj=new Ex();
        sopln(obj.i);
    }
```

static block:

→ the block which is declared with static modifier.
→ static block is executed before main method
→ static blocks are executed at the time of class loading
→ static block is executed at only one time for each & every class loading
→ Before Java 1.5 version we can execute java pgm without using main() method

ex:
```
static{
    sopln("hii");
    system.exit(0);
}
psvm(){
    sopln("hii");
}
```
o/p: hii

ex:
```
class PC{
    SB;
    IB;
}
class CC extends PC{
    SB
    IB1
    IB2
}
public class Ex{
    Ex obj=new Ex();
}
```
o/p: PC SB
CC SB
PC IB
CC IB1
CC IB2

```
public class Ex6
    Ex16
    {
    }
psvm(16
{
  Ex obj=new Ex();
  sopln(obj); →address
}
```

Command line arguments:

→ the arguments which are passed from command prompt are called command line args

→ we are passing command line args at the time pgm execution

→ class Ex123

```
{
psvm(String[]args){
    sopln(args.length);
    sopln(args[0]);
    sopln(args[1]);
}
for(int i=0; i<args.length; i++){
    sopln(args[i]);
}
}
```

If cmd prompt:

javac Ex123.java

java Ex123 10 20

java Ex123 10 20 30 "ita"

```
for(String i : args){
    sopln(i);
}
```

Variable argument method:

→ It is introduced in java 1.5 version onwards

→ It allows method to take any no. of args.

Ex: void m1(int...a){

}

m1(int...a, int...b)→x

Ex: public class Ex6

```
{
    void m1(int...a){
        sopln("Hii");  → for(int arr : a){
    }                          sopln(arr);
psvm(16                    }
{
  Ex obj=new Ex();
  obj.m1();
  obj.m1(10,20);
  obj.m1(10,10,30,40);
}
}
```

→ Inside the variable argument method we can declare only one argument otherwise we get compilation error

→ Inside the var.arg. method, we can take void m1(String...s, int...i)-(x)

→ Inside the method, variable argument must be the last

start Ex: m1(int a, String s, int i...i)

```
ex.cree
    void m1(int a, double d, char c, String s){
        sopln(a)
        sopln(d)
        sopln(c)
        for(String str:a){
            sopln(str);
        }
    prvm(){
        ex obj=new ex();
        obj.m1(5, 11.23, 'a', 'c', "Hii", "Hlo");
    }
```

method overloading vs. arg. method:

```
    void m1(int...i){
        for(int str:i){
            print(str);
        }
    void m1(String...s){
        for(String si:s){
            sopln(si);
        }
    }
    psvm(){
        ex obj=new ex();
        obj.m1(10,20,30);
        obj.m1("Hii","Hlo","bye");
    }
```

o/p: 10 20 30 Hii Hlo bye

Abstraction:-

→ Hiding the implementation & showing the functionality is called Abstraction.

→ Abstraction is possible in java by using abstract class & interfaces.

→ If you declare a class with abstract modifier then it is called Abstract class.

→ Abstract class is a collection of abstract & non-abstract methods, constructors & static blocks.

→ Abstract method:-

→ the method which is declared with abstract keyword

→ the method which doesn't have logic/implementation

→ every abstract class should be extended. In the child class we write implementation for abstract method.

→ In that class if you are not done implementation for abstract method declare that class as abstract
→ for abstract class we can't create object
→ for the abstract methods it is possible to provide any type of returntype

Ex:
```
abstract class Test {
    abstract int m1();
    abstract boolean m2();
}
class cclass extends Test {
    int m1() {
        return 10;
    }
    boolean m2() {
        return false;
    }
}
class Example {
    ps vm() {
        cclass obj=new cclass();
int i = obj.m1();
 sopln(i);
boolean j = obj.m2();
  sopln(j);
    }
}
```

calling m1() without obj.
new Test().m1();

→ For abstract method we can have any no of arguments.
→ In an abstract class zero (or) n. no. of abstract methods
→ If we don't have abstract methods also it is not possible to create object for abstract class

Encapsulation:

→ Combining / wrapping / binding the data & code as a single unit is called encapsulation
→ In java encapsulation is possible by classes
→ we are able to provide more encapsulation by taking private data members.
→ To get & set values from private data members we use getters & setters methods

ontation
abstract

ide ony type

without obj.

m1();

gumenbs.
act methods
t is not

as a

by taking

fore we

```
class encapsulation{
    private int sid;
    private String sname;
    public void setsid(int sid){
        this.sid=sid;
    }
    public int getsid(){
        return sid;
    }
    public void setsname(String sname){
        this.sname=sname;
    }
    public String getsname(){
        return sname();
    }
}
class Example{
    psvm(){
        Encapsulation obj=new Encapsulation();
        obj.sid=20
        obj.setsid(20);
        obj.setsname("isa");
    int i=obj.getsid();
    sopln(i);
    int j=obj.getsname();
    sopln(j)
    }
}
```

Super keyword

By using super keyword we can access super class variables/ methods/ even constructors also.

```
class Pclass{
    int i=100;
}
class cclass extends pclass{
    int i=200;
}
    void m1(){
        sopln("pclass m1");
    }
    pclass(){
        sopln("pclass const");
    }
```

```
class cclass extends pclass{
    int i=200;
    void m1(){
        sopln("cclass m1");
        super.m1();
    }
    cclass(){
        super();
        sopln("cclass const");
    }
    void display(){
        sopln(this.super.i);
        sopln(i);
    }
```

```
class ExS psvm(){
    class obj=new class();
    obj.display();
    obj.m();
}
```

block level variables → lowest scope
static var → largest scope

Scope of variables

→ static variables have the largest scope
→ static variables are created once class is created & survival as long as class space in JVM
→ Instance variables are created when new object is created & stay as long as object is removed
→ Instance variables are have the 2nd largest scope
→ Local variables have the lowest scope
→ these variables are stored in stack memory & they survival until method/constructor execution is completed
→ Block level variables are available as long as block is executed
→ there are 3 ways to accept dynamic i/p values:-
1) BufferReader class available in java.io package
2) Scanner class available in java.util package
3) Console class available in java.io package

Interfaces

→ By using Interfaces we can achieve 100% abstraction
→ Interface is one type of a class which contains only abstract methods.
→ By default java compiler will add public & abstract keywords before the method.
→ Interface is an extension of Abstract class
→ Interface can be implemented by using implements keyword
→ By using Interface we can achieve multiple inheritance
→ By using interface keyword we can create interface

ex: interface interface_name{syntax}

```
interface Printable{
    void print();
}
interface Drawable extends Printable{
    void draw();
}
```

ifles-) lowest scope
largest scope

created &

en new object
removed
of scope

emory & they
on is completed
ing as

values:-
package
ackage
ge

abstraction
fairs only
& abstract

ements keyword
e inheritance
face

```
class Example implements Drawable {
    public void print(){
            sopln("printable");
    }
    public void draw(){
          sopln ("drawable");
    }
 p s v m (){
    Example obj=new obj();
      obj.print();
      obj.draw();
  }
}
```

Nested Interfaces

→ The interface which resides inside the class & inside another interface is called nested interface.

```
i) class A {                 ii) interface i1{
   interface i1{} }              interface i2{}
   }                             }
```

```
→ class Test1 {
      interface i1{
          void print();
      }
  }
class Example implements Test1.i1 {
    public void print(){
            sopln("Hlo");
    }
    psvm(){
        example obj=new Example();
        obj.print();
    }
}
```

Adaptor class :

→ It is an intermediate class b/w interface & user-defined class & it contains empty implementation for abstract methods.

```
interface i1{
    void m1();
    void m2();
}
class Adaptor implements i1 {
public void m1(){ }              | public void m2(){ }.
```

Exception Handling:

→ the errors which are occured at run time are called run time errors

→ we get run time errors if we give wrong i/p values or error in the logic

→ there are 2 types of exceptions:
1) user defined exceptions 2) predefined / Built In exceptions

2 types:
1) Asynchronous exceptions (Hw problems)
2) Synchronous exceptions (programmatic problems)

it is a type ←
1) checked exception (compile time exception)
Ex: classNotFoundException
FileNotFoundException
IOException
SQLException
2) Unchecked exception (Run time exceptions)
Ex: ArithmeticException
NullPointerException

→ Asynchronous exceptions are available under java.lang.Error class

→ Synchronous exceptions are available under java.lang.Exception class

JVM → doesn't accept irrelevant data
↓ request for JRE
JRE
↓
Throwable class (it identifies which type of error) — Error / Exception

→ we can handle the exception in java by using try & catch & 2 rd throws keyword.

→ In exception handling we are having five keywords try, catch, finally, throw, throws.

→ we have three blocks: try block, catch block & finally block

→ C, C++ → single threaded Applications & Java, Python are called Multiple threaded applications.

→ C, C++ → single flow & Java, Python → multiple flow

try block: In this block we write exception related code.
→ from this block we get exceptions that exceptions should be handled in catch block
→ catch block is called Exception Handling Block
→ one try can have multiple catch blocks
→ catch block can't be alone it should follow the try block

Syntax: try{
    exception code;
}
catch(ExceptionClassName referenceVar){
    exception handling code;
}

Ex: try{
    Scanner scan = new Scanner(System.in);
    System.out.println("enter 1st no:");
    int a = scan.nextInt();
    sopln("enter 2nd no:");
    int b = scan.nextInt();
    int c = a/b;
    sopln(c);
}
catch(ArithmeticException e){
    sopln(e); → package name, ECN, message
    sopln(e.getMessage()); → only message
    sopln e.printStackTrace(); → PN, ECN, msg
}

Ex:
String s = null;
sopln(s.length) → NullPointerException

Ex:-
→ At a time one Exception will be thrown & at a time one Exception should be handled.
→

Ex: try{
    int arr[] = {10, 20, 30, 40};
    arr[5] = 10;
}
Catch(ArrayIndexOutofBounds, Exception e){
    sopln(e);
}

→ when we are handling multiple catch blocks the order of exception is important
→ we can't write any individual starts b/w try & catch blck but any where we can write
Ex: try{
    sopln("Hello");
  }
  sopln("Hello");
 catch(e){
 }
→ we can write exception code in catch block also but it is not correct way.
→ All the exception classes are derived from Exception Base class
→ Ex: catch(Exception e){ → it can handle all exceptions
        sopln(e);
      }
→ multiple try-catch block
Ex: try{}
    catch(){}
    ------
    try{}
    catch(){}
→ try with multiple catch blocks
Ex: try{}
    catch(){}
    catch(){}
→ Nested try block
Ex:
```
try{
    try{}
    catch(){}
}
catch(){}
```
```
try{}
catch(){
    try{}
    catch(){}
}
```
```
try{
    try{}
    catch(){}
}
catch(){
    try{}
    catch(){}
}
```

finally block:-
→ this block is executed before closing the pgm
→ In this block we write memory related code, closing the file streams and the DataBase connections.
→ this block can't be alone it should follow either try/catch block
→ this block is executed if exception occured/not

→ if exception handled/not finally block is executed

Ex: try{
    sopln("try block");
    sopln(10/0);
}
catch(ArrayIndexOutofBoundsException e){
    sopln(e);
}
finally{
    sopln("Final block");
}

o/p: try block
    final block
    Arithmetic Exception

→ try{
    sopln("Hello");
System.exit(0);
}
finally{
    sopln("Hii");
}

→ In this case only the finally block will not be executed

throw keyword

→ this keyword is used to create an exception object explicitly for pre/user defined.
→ It is like a try block
→ Mainly it is used in user defined exceptions.

Syntax: throw new ExceptionClassName(String msg);

Ex: static void validate(int age){
    if(age<18){
        throw new ArithmeticException("not eligi..");
    }
    else{
        sopln("eligible..");
    }
psvm(){
    try{
        validate(13);
    }
    catch(ArithmeticException e){
        sopln(e.getMessage()); → not eligi.
    }

User defined / programatic exceptions / Custom defined exceptions

steps:

1) create appropriate packagename
2) choose user defined class
3) it should extend either Exception / Runtime class
4) Every sub class should have parameterized constructor by taking string datatype as a parameter
5) this constructor should be called by using super string parameter always

Ex: package isajavaexamples.na;
public class Nage extends Exception {
public Nage(String s){
super(s);
}

Package isajavaexamples;
import Isajavaexamples.na.Nage;
public class isajavaexamples {
public void validate(String s) throws Arithmetic Exception, Nage {
int age = Integer.parseInt(s);
if (age<20){
Nage obj=new Nage("not eligible");
throw(obj);
}
}
psvm(){
String s1=args[0];
Isajavaexamples obj=new Isajavaexamples();

checked Exceptions (compile time):

1) IOException 2) SQLException 3) ClassNotFound Exception
4) CloneNotSupported Exception → when you are unable to clone obj
5) IllegalAccessException → Access denied problem
6) InterruptedException → when multiple threads accessing at time
7) InstantiationException → when you are create obj for abstract classes.
8) NoSuchField Exception
9) NoSuch Method Exception

Unchecked Exceptions

1) Arithmetic Exception
2) Array Index Out of Bounds Exception
3) Input Mismatch Exception
4) Class Cast Exception → we can't invalid type casting
5) Illegal Thread state Exception → we can't start the same thread twice
6) Negative Array size Exception
7) Null Pointer Exception
8) Number Format Exception → sriRam
9) string Index Out of Bound Exception

* Error :-
→ we get this errors due to lack of system Resources
Ex :- Stack overflow Error, out of memory.
Assertion Error

* Multi Threading
→ Executing several tasks or simultaneously at the same time.
→ there are 2 types :-
1) process based & 2) thread based

thread :-
→ thread is a small unit of code which has its own execution part
→ thread is a lightweight process means its consumes less memory & space
Ex :- Bookmyshow, Online Reservation systems., Bank to in account
→ In java thread is a class.
→ there are 2 ways to create a thread :-
1) By extending thread class
2) By implementing Runnable Interface
→ Under the thread class we have following threads :-
1) start() 2) stop() 3) resume() 4) setName(string)
5) getName() 6) setPriority(int)
7) int getPriority(int) 10) Interrupted methods :-
1) sleep(milliseconds)
2) join() 3) Yield()

11) Daemon, Active_count(), is_Alive(), run()
current Thread().

→ Every java/python pgm will have one thread that is run at background is called "Main thread."

life cycle of a thread

1) New born stage
2) Ready stage
3) Running stage
4) waiting stage
5) Halt/Dead stage

→ Interrupted methods will throw the exceptions

1) **New born stage**
→ once we allocated memory thread is in "New born stage."
→ thread will enter into Main Memory

2) **Ready stage**
→ thread is ready to execute but thread scheduler has not schedule the thread.

3) **Running stage**
→ thread schedulers has schedule the thread if & only if the thread is under the control of CPU.

4) **waiting thread :**
→ if we are using sleep() method asking thread to wait for specific period of time it the thread is suspended.

→ 5) **Halt/ Dead stage**
→ once thread execution is completed/ once we release the memory for thread.

Creating the thread

→ Creating the thread directly
Ex: thread th = new thread()
th.start()

→ By factory() method
Ex: thread th = new thread.current_thread();

→ By creating new thread from base class
Ex: class mythread extends thread

Thread priorities

→ priority value ranges from 1 to 10
→ 1 is minimum & 10 is maximum & 5 is default/Normal priority.
→ public/static/final/void setPriority(int)

getName() & setName():

→ For the threads we can provide user-defined & Built-in names.

Syntax: public final void setName(string);
public final string getName();

Ex:
```
Thread th=new Thread()
//before assigning name
sopln(th.getName()); → Thread-0
//after assigning name
th.setName("Hi");
sopln(th.getName()); → Hi.
```

getPriority() & setPriority()

Ex: Thread th=new Thread()
int i = th.getPriority();
sopln(i); → 5
th.setPriority(Thread.MAX_PRIORITY/10);
sopln(th.getPriority()); → 10

MAX_PRIORITY/10
MIN_PRIORITY/1
NORM_PRIORITY/5
} Constants

Creating thread by extending Thread class

Ex:
```
class mythread extends Thread {
    public void run(){
        sopln("Hi");
    }
public class example {
    psvm(){
        mythread obj=new mythread();
        obj.start(); → o/p: Hi
    }
}
```

creating the Thread by using Runnable Interface

ex: class mythread implements Runnable {
    public void run() {
        sop("Hi");
    }
}
public class ex {
    psvm() {
        mythread obj = new mythread();
        Thread th = new Thread(obj);
        th.start(); → o/p : Hi
    }
}

Interrupted Methods :

1) sleep(long milliseconds) 2) join() 3) Yield()

→ this three methods will throw InterruptedExceptions

class mythread extends Thread {
    public void run() {
        for(int i=1; i<=5; i++) {
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException e) {
                sop(e);
            }
        }
    }
}
public class ex {
    psvm() {
        mythread obj = new mythread();
        obj.start();
    }
}

→ we can't start the same Thread twice if we do so
2nd time we get IllegalThreadStateException

class mythread1 extends Thread {
    public void run() {
        for(int i=1; i<=5; i++) {
            try {
                sop(i);

```
        thread.sleep(1000);
      }
      catch(InterruptedException e){
          sopln(e);
      }
    }
}
class mythread2 extends thread{
        some code
}
public class ex6
{
  psvm(){
  mythread1 ob1 = new mythread1();
  mythread2 ob2 = new mythread2();
   ob1.start();} → o/p : 1 3 5 2 4 } without join
   ob2.start();
  {
  ob.start();
  try{
    ob2.join();
  }catch(InterruptedException a){
      sopln(a);
  }
}
}
```

→ with join → 1 2 3 4 5 1 2 3 4 5

Current thread() method;

→ this method is used for obtaining the thread which are running in the main memory of the computer

ex: thread th = ~~new thread();~~ thread.currentThread();
~~system~~ sopln(th.getName()); → main
th.setName("hi");
sopln(th.getName()); → hi

isAlive() method

→ this method returns true if thread is running / in waiting stage

ex: thread th = thread.currentThread();
sopln(th.isAlive()); → true
thread obj = new thread()
sopln(obj.isAlive()); → false

obj.start()
sopln(obj.isAlive());→true

Daemon threads

the threads which are running at the background are called Daemon threads.

eg: class mythr extends thread {
    public void run(){
        sopln("thio");
    }
}
public class ex{
    psvm(){
        mythr th=new mythr();
        sopln(th.isDaemon());→false
        th.setDaemon(true);
        sopln(th.isDaemon());→true.
        th.start();
        th.setDaemon(true);
                ↓
            exception

join

join

→ Synchronization

→ To avoid data inconsistency problem we use synchronization. that means only one thread at a time (if any variable, method, objects classes is shareable then we apply concept of Synchronization)
→ synchronized keyword can be applied to the methods & blocks

eg: class Amount {
    int bal=0;
    synchronized void pt(int amt) {
        bal=bal+amt;
        sopln(bal);
    }

threads which
mputes
atthread();

Interthread Communication

uning / in

);

→ one thread communicating with another thread is called Interthread Communication
→ the o/p of the 1st thread is given as i/p of 2nd thread
→ To develop interthread communication application we use object class, it is available under java.lang package.

→ Wordad this class we have following methods;
1) public void wait(long milliseconds)
2) public void wait(long milliseconds, int nanoseconds)
3) public void wait()
4) public void notify()
5) pubic void notifyAll()

JDBC (Java DataBase Correctivity)

→ It is not an installable software
→ JDBC is an API.
→ It is used to correct Java frontend application with any kind of relational DataBase.
→ JDBC API acts like a mediator/ Interface b/w Java & DataBase.
→ To correct with Relational Databases we require JDBC drivers.
→ Every database got its own JDBC Drivers.
→ We can get the Drivers from sun micro system, DataBase vendors, third Party Vendors.
→ Types of JDBC Drivers:

1) Type-1 Driver (JDBC-ODBC-Bridge Driver)
2) Type-2 Driver (Native API Driver)
3) Type-3 Driver (Network Protocol Driver)
4) Type-4 Driver (Thin Driver)

1) Type-1 Driver

→ ODBC Drivers comes with windows os. This ODBC-JDBC Driver convert JDBC method calls into ODBC function calls
→ It is suitable for small scale applications
→ These are platform dependent Drivers
→ It is used for development & testing not for deploying
→ It is the default Driver
→ It is the universal Driver means it can connect to any relational databases

Java App ⇄ JDBC → ODBC ⇄ DataBase

2) Type 2 Driver :
→ It uses client-side libraries . It converts JDBC method calls into Native calls inorder to interact with different DataBases.

3) Type-3 Driver :
→ It is called Middleware Application server Driver.
→ It converts JDBC method calls into vendor specific protocol Drivers.

4) thin Driver :
→ It is also called Native Protocol Driver.
→ 100% Pure Java Drivers.
→ It can directly interact with DataBases

→ JDBC API comes in the form of two packages:
1) java.sql   2) java.x.sql

1) java.sql
→ Interfaces :
1) Driver
2) Connection
3) statement
4) Prepaidstatement
5) CallableStatement
6) DataBaseMetaData
7) ResultSetMetaData

classes :
1) Types classes
2) DriverManager
3) Blob class
↓ datatype of image in database

→ To see the port no's : ctrl + Alt + L → Task Manager → services

steps to Connect to the Database

1) Register the Driver class
→ By using classforname() method every JDBC Driver should be registered in DriverManager class

```
public class ex{
    psvm () throws Exception{
        Class.forName("oracle.jdbc.driver.OracleDriver");
```

oracle.jdbc.driver → package
OracleDriver → oracleDriver classname
library ← mysql.connector / ojdbc

```
        class.forName("com.mysql.jdbc.Driver");
```

2) Create Correction object by using getCorrection() of DriverManager class.

Ex:- Correction con = DriverManager.getCorrection("jdbc:mysql://localhost/mydb1", "root", "root");
if(con!=null){
    sopln("corr.successful");
}
else{
  sopln("corr not Successful");
}

3) Create statement object by using createStatement() of Correction Interface

→ Under the statement Interface we have 2 methods
1) executeQuery() → when we are using DRL commands/ select stmts → this method returns Resultset.

Ex:- Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from emp");

→ Executeupdate() is used when we are using DML statements like insert, delete, update.
→ It returns integer value
String sql = "insert into tableemp values(?,?,?,?,?);
PreparedStatement ps = con.prepareStatement(sql);
ps.setInt(1,100);
Ps.setString(2,"ira");
Ps.setString(3,"wq");
Ps.execute()
Ps.close()
con.close()

→ Statement st = con.createStatement()
ResultSet rs = st.executeQuery("select * from tableemp");
while(rs.next()){
    sopln(rs.getInt("empno") + " " + rs.getString(
}
con.close();
}

nection() {

ection("jdbc:
ob");

executeQuery() {

methods
QL commands/
ltset.

from emp");

using DML

1,1,1,1,1);
executeQuery(sql);

data");

.getString(2));

servlets & JSP (Java Server Pages)

→ servlet is an API.
→ By using servlets we can develop dynamic web applications
→ servlet API consist of 2 packages:-
1) java.x.servlet.http;
2) javax.servlet.*;
→ there are 3 ways to create a servlet:
1) By using servlet Interface
2) By using Generic servlet class
3) By using HttpServlet class

five life cycle Methods

→ Every servlet will have 5 life cycle methods:-
1) init() → the code to be executed for one time use this method.
Ex: Opening file streams, database connections
2) service() → In this method we write logic that request sent to server & respond back
3) destroy() → this method is executed after we destroy servlet object.
4) getServletInfo() we can get info. about servlet like version of servlet, author name
5) getServletConfig().

- int a = Integer.ParseInt("four") (only integrals values)
sop(a); NumberFormatException

UDP (universal description protocol)
SOAP (simple object access protocol)

→ class isaservlet implements Servlet
→ class isaservlet extends GenericServlet } → creating servlet in 3 ways
→ class isaservlet extends HttpServlet

Content type:

→ It is also known as MIME (Multipurpose Internet Mail Extension). It is the HTTP header that describes what type of info. sending to the browser.

List of content type:

1) text/Html
2) text/plain
3) application/ms-word
4) application/jar
5) application/pdf
6) images/jpeg
7) images/gif
8) images/prg
9) audio/mp3
10) video/mpeg
11) video/quicktime

→ res.setContentType("text/Html");

PrintWriter class:

→ this class available under java.io package.
→ It is used to write o/p data in readable format.
→ It is an abstract class

Eg: PrintWriter ps = res.print();
PrintWriter ps = res.getWriter();

Eg: Html code

```
<form action="addserv">
    <center>
    <H1><b> Add application </b></H1>
    First number: <input type="text" name="n1" />
    <br>
    <br>
    second number: <input type="text" name="n2" />
    <br>
    <br>
    <input type="submit" name="submit" value="ADD" />
    </center>
</form>
```

create "Addserv" file

```
import java.io.*;
import java.servlet.*;
import java.x.servlet.http.*;
public class addserv extends GenericServlet {
  public void service (ServletRequest req, ServletResponse
  res) throws ServletException {
    res.setContentType ("text/html");
    PrintWriter ps = res.getWriter();
    String s1 = req.getParameter ("n1");
    String s2 = req.getParameter ("n2");
    int i = Integer.parseInt (s1);
    int j = Integer.parseInt (s2);
    int sum = i+j;
    //ps.print(sum);
    ps.print ("<html>");
    ps.print ("<body>");
    ps.print ("<h1>");
    ps.print(sum);
    ps.print ("</h1>");
    ps.print ("</body>");
    ps.print("</html>");
  }
```

registration

```
import java.io.*;
import java.sql.*;
import javax.servlet.http.*;
import javax.servlet.*;
public class register extends extends HttpServlet {
public void doPost (HttpServletRequest req, HttpServlet
Response res) {
  try{
  res.setContentType ("text/html");
  PrintWriter out = res.getWriter();
  String n = req.getParameter ("uname");
```

```
String p = req.getParameter("pwd");
String e =      "      ("adal");
String c =      "      ("phno");

    class.forName("com.mysql.getConnection
    Connection con=DriverManager.getConnection("jdbc:my
sql://localhost/jsadbase;"root","root");
PreparedStatement ps = con.prepareStatement("insert into
    registerusers values(?,?,?,?)");
    Ps.setString(1,n);
    Ps.setString(2,p);
            (3,e);
            (4,c);
    int i=ps.executeUpdate();
    if(i>0)
  out.print("you are successfully registered");
  }
  catch(Exception e){
      sop(e);
  }
```

→ Under the HttpServlet class we have following methods:

1) doGet()
2) doPost()
3) doError()
4) doTrace() → Handle finding errors
5) doHead()

> javac -d . ↓ current directory
switch

> javac -d e:/filename

package:

→ It is a directory which consists of similar type of classes & interfaces.
→ It is a directory which consists of ".class" files.
→ Java support 2 types of packages.
1) Built-In packages 2) User-defined packages
* Built-In packages: the packages which are given by java software
→ java.lang is the default package

User-Defined package : the packages which are created by the user.

→ By using "package" keyword we can create a package

syntax:

```
package packagename;
```

→ In one sourcefile it is possible to take only one package

syntax for compiling the package

```
>javac -d . filename.java
```

→ -d (it tells the compiler to create seperate directory structure)

→ . (place the directory structure in the current working directory)

1st: creating package & classes

```
package packdemo;
public class Iclass{
public void m1(){
  sop("m1 method");
}
}
```

2nd: importing this package

```
import packdemo.*;
public class Ex{
 psvm(){
 Iclass obj=new Iclass();
 obj.m1();
   (or)
packdemo.Iclass obj=new
 packdemo.Iclass();
  obj.m1(); → m1 method
}
```

>>> → right shift with fill zeroes

→ Strings are immutable

```
a=true;
b=false;
c=a^b;
sop(c) → false
```

```
byte x=127;
x++;
x++;
sop(x); → -127
```

javac filename → compiles → java classname → interpreted

→ stackOverflow Er. → error exception → java.lang package.

→ JVM (Just in time) compiler → it converts byte code into native code

→ jre (java runtime environment) → to run the pgm

→ JDK (to write the pgm)

finalize → method → utan : destroy the object
finally → block
final → keyword / modifier

Applets :

→ In java, we can develop two types of applications:
1) stand alone applications
2) Distributed applications.
standalone Applications (offline)
→ standalone application runs in the context of local hard disk. It don't require internet, browser, life cycle methods.
Distributed Applications (online)
→ these applications runs in the contex of browser / www (world wide web)
→ it requires life cycle methods

→ Applet is a small java pgm in the context of browser (or) www.
→ To create the Applet we require java.applet * package.
→ this package contains only one class i.e., Applet class
→ Applet class contains following life cycle methods:
1) init()
2) start()
3) stop()
4) destroy()
→ other method which is not life cycle method is
public void paint(String, int rowposition, int colposition)
→ there are 2 ways to run the applet:
1) by using HTML pgm
→ In this we use <applet> tag
Ex: <applet code = "codepp" width = 100px height = 100px>
2) by using applet viewer tool

```
import java.applet.Applet;
import java.awt.*;
public class MyApplet extends Applet {
    public void paint(java.awt.Graphics g) {
        g.drawString("Hello", 30, 20);
    }
}
```

desktopwindows (00) GUI (00)

AWT (Abstract Window Toolkit)

→ It is used for developing standalone applications in java
→ It requires ~~awt~~ java.awt.*; package
→ It extends a predefined class i.e., frame class available under java.lang; package
→ By using AWT we create static components like button, textbox etc.

container

→ It is the GUI component which is used to place all components.
Ex: Frame, Applet.

Event

→ It is an action generated on the components.
Ex: ButtonClick, CheckboxChecked.

Frame:

→ It can be created by using constructor by using Frame class
Ex: Frame f = new Frame("Vijay");

```
class MyFrame extends Frame {
    MyFrame obj = new MyFrame();
}
```

→ Basic Component in AWT is Frame. It is available under java.awt; package
→ It is a container which is used to place the components
→ By default frame is in invisible mode
→ By using setVisible() we
→ By default frame has 0 width & 0 height

```
frame f = new Frame();
f.setVisible(true);
f.setSize(100,100);
f.setTitle("vidya");
f.setBackground(Color.red)
```

corporate (All are the classes):

1) label:

→ this control is used to display label on the frame.

```
Label lb = new Label("welcome");
f.add(lb); → adding label on the frame.
```

2) textbox:

```
Textbox tb = new Textbox();
tb.setText("Hello");
String s = tb.getText();
    sop(s);
f.add(tb);
```

3) choice:

```
Choice ch = new Choice();
  ch.add("c++");
  ch.add("Java");
  ch.add("python");
  ch.insert("html", 2);
  ch.remove("Java");
  ch.removeAll();
  f.add(ch);
```

```
String s = ch.getSelectedItem();
    sop(s);
```

("html", 2);

→ By using AWT we creat the static controls which doesn't have any life. for this we require the events

→ By default the corporate will not listen the event so, we require Event Listeners

→ there are 2 types of listeners:

1) Action Listeners
2) Window Listeners

syntax for adding listeners to the corporate:

public void addxxxxlistener(xxxxListener e)

component	Event source	Listener name	Listener method
1) Frame	windowevent	windowlistener	windowclosed() windowdosing() windowopening() windowactivate() windowopened() windowdeactivate()
2) TextField	ActionEvent	ActionListener	Actionperformed()
3) TextArea	"	"	"
4) Menu	"	"	Actionperformed()
5) Button	"	"	"
6) checkbox	ItemEvent	ItemListener,	itemselectionchaged()
7) radio	"	"	"
8) list	"	"	"
9) choice	"	"	"
10) scrollbar	AdjustmentEvent	AdjustmentListener	AdjustmentValue changed()
11) Mouse	MouseEvent	MouseListener	MouseEntered() MouseExited() Mousepressed() MouseReleased() Mouseclicked()
12) keyboard	keyEvent	keyListener	keyTyped() keyPressed() keyReleased()

Ex: class myFrame extends Frame {

myFrame(){

this.setVisible(true);
this.setSize(100,100);
this.setTitle("Hello");
this.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent we) {
System.exit(0);

}
});

Window Adaptor class :

→ this class is used to recieve / adapt window event

→ smallest integer datatype : byte.