JAVA

→ Core Java
→ Advance Java
→ Android Java

⇒ Basics :-

1. JDK JRE JVM (Intro)
2. History of JAVA.
3. Data type
4. variables.
5. looping statements
6. conditional statements
7. Arrays. (Intro)
8. Strings (intro)
9. methods.

⇒ oops :-

10. Class
11. object.
12. Static keyword.
13. constructor.
14. This keyword.
15. Super keyword.
16. Inheritance.
17. polymorphism (overloading & overriding).
18. Abstraction
19. Encapsulation
20. Abstract classes

(21) Interfaces.

=> <u>Core Java topics</u> :-

(22) Exception Handling

(23) File Handling.

(24) Abstract method.

(25) Interface

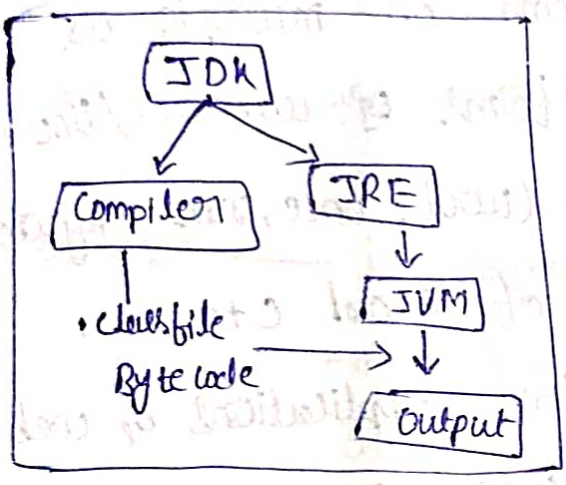
(26) package

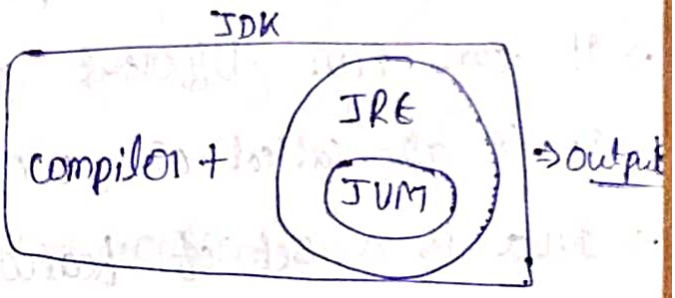
(27) multi-Threading.

— Basics —

① Introduction of JDK - JRE - JVM.

* JDK (JAVA DEVELOPMENT KIT)
* JRE (JAVA RUN-TIME ENVIRONMENT)
* JVM (JAVA VIRTUAL MACHINE). → Interpreter





* for compilation → javac filename.java
* for execution → java filename

→ so finally we are using a software to execute the java programming in JDK

→ In JDK we have two thing, normal java compiler & JRE

→ After writting the program the compiler compiles the program into byte code.

→ Here JAVA consist of JVM.

→ The compiled byte code will be given to the JVM

→ Then finally the JVM Returns the output

→ The JVM is an operating system dependent that is different to different OS.

② History of JAVA :-

→ JAVA is created by james gosling by sun mycrosystem in 1991

→ java is a high level programming language.

→ java is a object oriented programming language

→ java is to write a program on multiple OS.

→ It can Run different platforms Eg:- windows/linux.

→ It is also called as WORA (write once, run anywhere)

→ java is a set of feature of C and C++

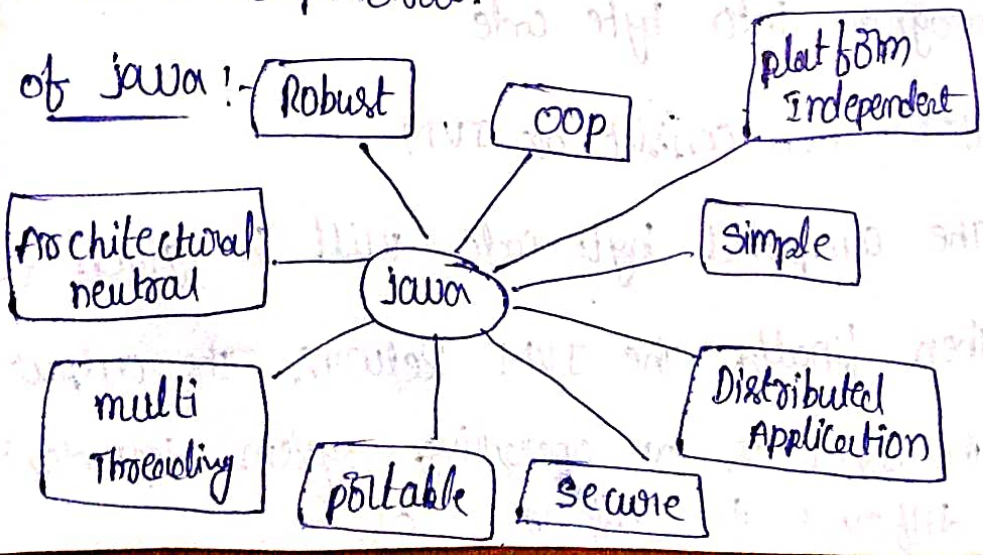
→ java supports both stand alone applications & web applications and also in Enterprise applications middleware applications.

→ java is a general purpose & powerful programming language.

→ used for developing software that run on mobile desktop & servers.

→ It is machine independent.

* features of java :-



↳ platform Independent — Run on any environment. ③

↳ OOP —— The all object oriented concepts.

↳ simple —— no concept of pointers, explicity memory Allocation structures, operator overloading.

↳ secure —— secure for internet application, Boundary Exception in Arrays strong typed language

↳ Robust —— Early checking of errors, Garbage collection Exception Handling.

↳ portable —— java programming written on one environment but it can be executed in another environment also.

↳ multi Threading — concurrent execution of several parts of same program at the same time & Improve CPU Utilization.

↳ Distributed Application — Software that runs on multiple computers Connected to a network at the same time

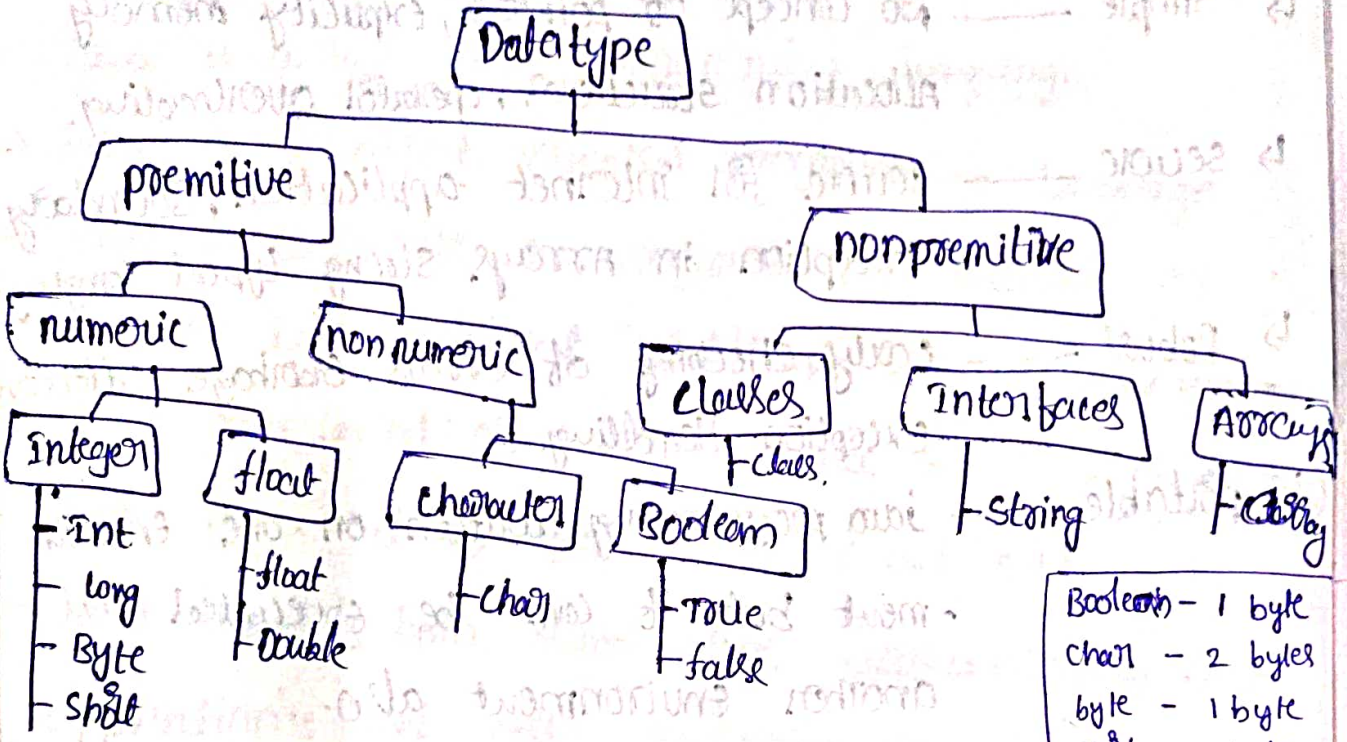
* RMI (Remote method Invocation)

* EJB (Enterprise java Beans)

↳ Architectural neutral — Irrespective of architectural The memory allocated to the variables will not vary.

(3) Data types in JAVA :-

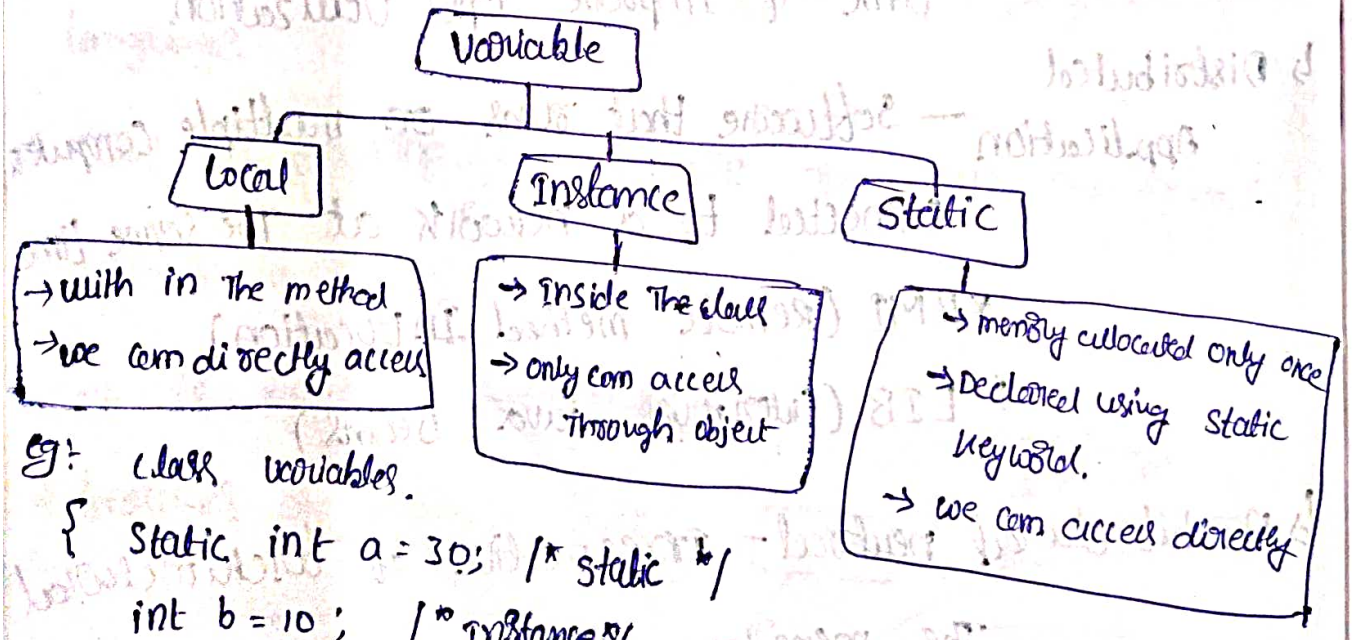
→ Data type specifies type of the Declared variable



Boolean	- 1 byte
char	- 2 bytes
byte	- 1 byte
shot	- 2 bytes
int	- 4 bytes
long	- 8 bytes
float	- 4 bytes
Double	- 8 bytes.

(4) Variables in JAVA :-

→ a variable is some thing that stores values in it.



eg:- class variables.

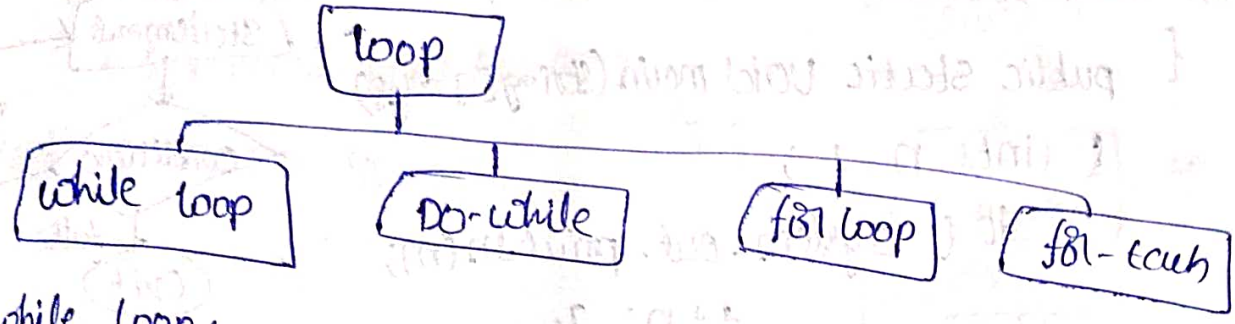
```
{ Static int a = 30; /* static */
  int b = 10 ; /* Instance */
  public static void main (String arg[])
  { int c = 20; /* local */
    System.out.print (a);
    System.out.print (c);
    variable obj = new variable ();
```

```
System.out.print(obj.b);
    }
}
```

output :- 30
20
10

⑤ Looping Statement in java :-

→ whenever we have to repeat certain statement several times Then looping is used.



* while Loop :-

→ It is a pre test loop

→ It is used when we don't know The no.of interations in Advance

→ It is also known as Entry Control loop

Syntax :-

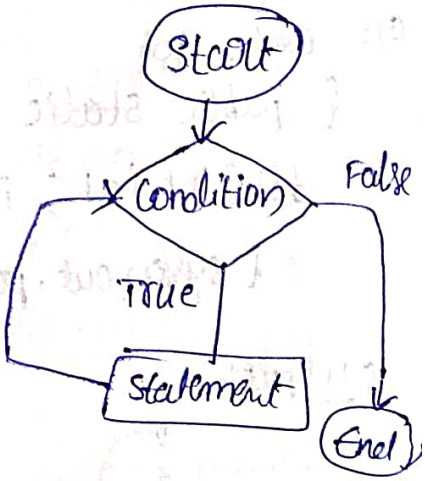
```
while (condition)
{ ..... .
  ..... .
  ..... .
  ..... .
  Inc / Dec
}
```

Eg :-

```
class while
{ public static void main (String[] args)
  { int n = 1;
    while (n<=5)
    { System.out.print("Hi");
      ++n;
    }
  }
}
```

Output :- Hi
Hi
Hi
Hi
Hi

flowchart :-



* DO-while loop :-

syntax :-

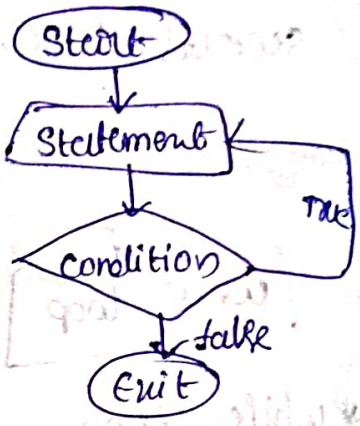
```
do
{ ..... .
  ..... .
  ..... .
  inc/dec ;
while (condition)
```

→ DO-while loop is a post test loop

→ It is used when we want to execute loop at least one time even condition is false.

→ It is also called as exit control loop

flowchart :-



Eg:- class Dowhile

```
{ public static void main (String[] args)
  { int n = 1 ;
    do { system.out.println(n);
         ++n; }
    while (n<=5); } }
```

output :- 1
2
3
4
5

* for loop :-

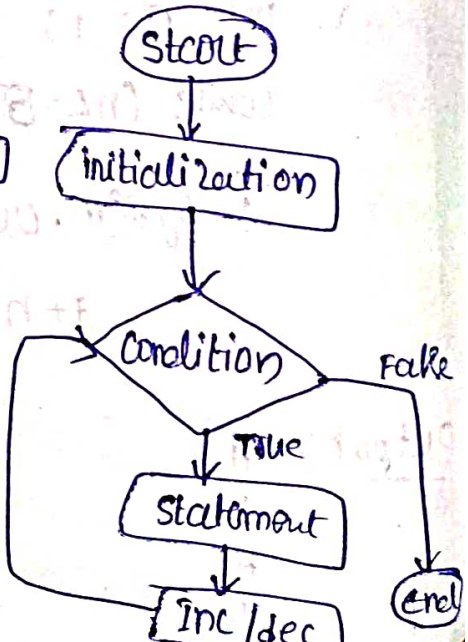
syntax :-

```
for(initial ; condition ; inc/dec)
{ ..... .
  ..... . }
```

→ for loop is the most commonly used loop

→ It is used when we want to perform initialization, condition & increment/decrement operations on single line.

flowchart :-



Eg:- class for

```
{ public static void main(String[] args)
  for(int i=1 ; i<=5 ; i++)
  { system.out.println(i+" "); } }
```

output :- 1
2
3
4
5

* for Each loop :-

→ for each loop mainly used to fetch the values from a collection like array.

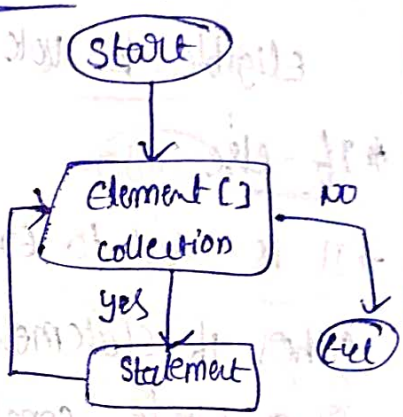
Syntax :-

for(datatype var1 : var2)
{ . . . . . }

Eg:- class foreach
{ public static void main(String[] args)
{ int a[] = {10, 20, 30, 40, 50};
for (int b : a)
{ system.out.print(b + " "); } } }

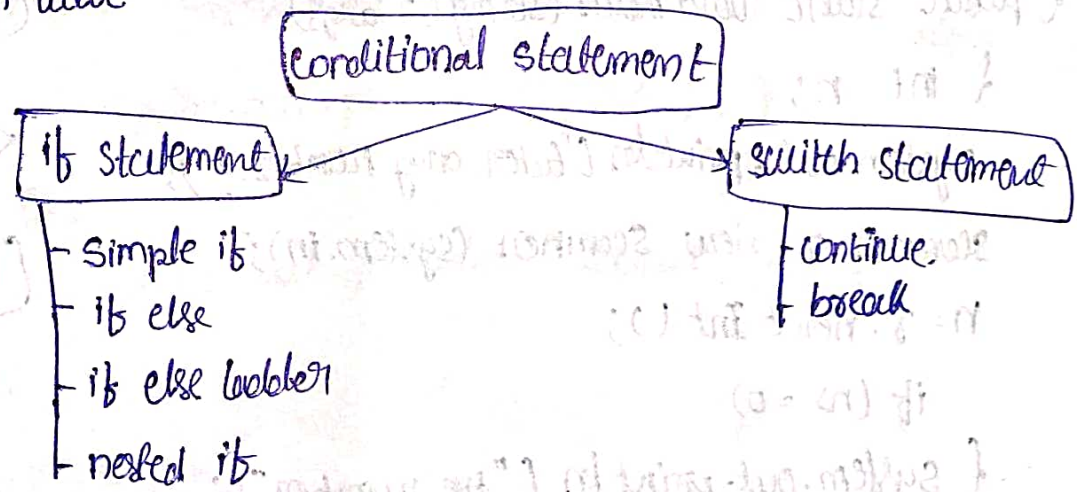
output :- 10 20 30 40 50

flow chart :-



⑥ Conditional statements in java :-

→ conditional statements work on the condition based on truth value



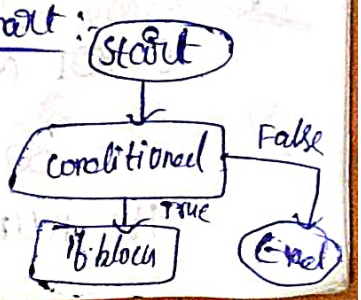
* Simple If statement :-

→ It is used when we want to test a condition.

Eg:- class simple
{ public static void main (String [] args)
{ int age;
system.out.println ("Enter your age...!");

syntax :-
If (condition)
{ . . . . . }

flow chart :-



```
Scanner r = new Scanner (System.in);
    age = r.next Int ();
     if (age >= 18)
{ System.out.print ("Eligible for votes...!"); }
  System.out.print ("thank yo...!"); } }
```

output:-

Enter your age : 20

Eligible for vote... ! Thank you...!

* If - else :-

→ It is used to execute two statement either if statement or else statement for a single condition.

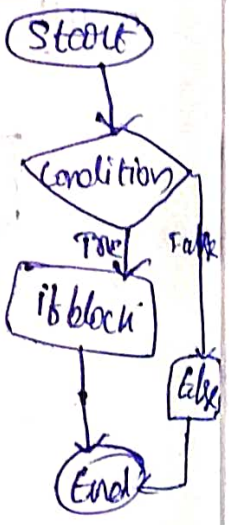
Syntax:-

```
if (condition)
{ ..... }
else
{ ..... }
```

eg:- class ifelse

```
{ public static void main (String [] args)
  { int n;
    System.out.println ("Enter any number..:");
    Scanner r = new Scanner (System.in);
    n = r.next Int ();
      if (n > = 0)
    { System.out.println ("+ve number");
      else
      { System.out.println ("-ve number");
                                    } } }
```

How chart:-



output :

Enter any number...!

30

+ve number

* Else if ladder :-

It is used when we used only one if block, multiple else-if block & at least one else block.

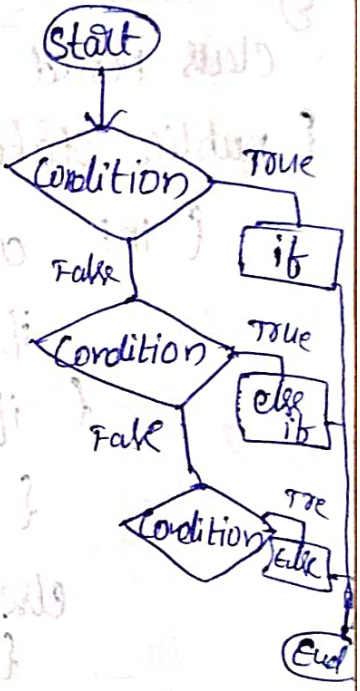
Syntax :-

```
if (condition)
{ . . . . . }
else if (condion)
{ . . . . . }
else
{ . . . . . }
```

Eg :- class else if

```
{ public static void main
                    (String [] args)
  { int marks;
  system.out.println ("Enter marks");
  scanner ref = new scanner (system.in);
   marks = ref.nextInt ();
    if (marks > 80)
     { system.out.println ("Topper"); }
     else if (marks < 80 && marks >= 60)
      { system.out.println ("first"); }
     else
     { system.out.println ("second"); } } }
```

flow chart :-



Output :-

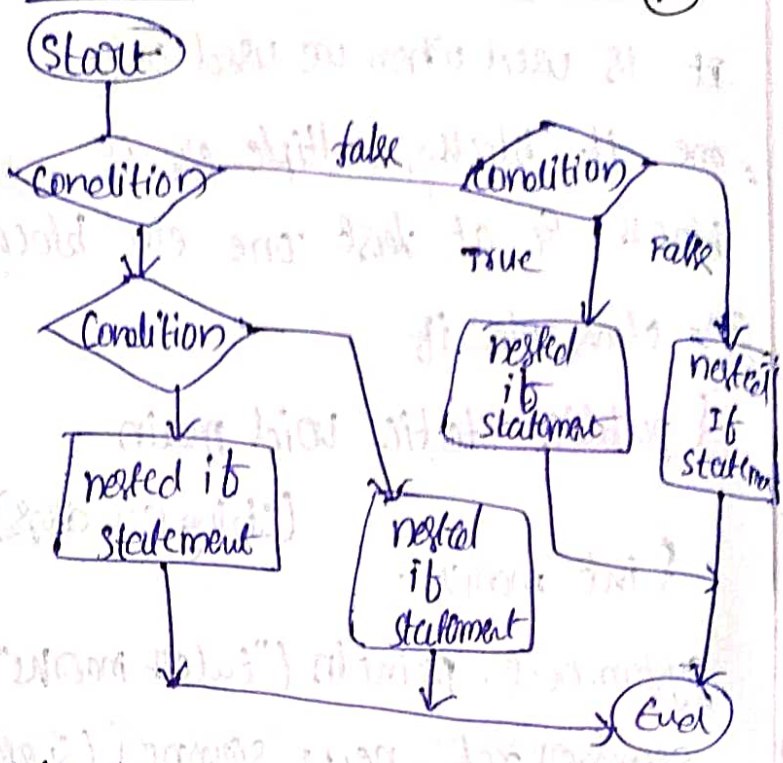
Enter marks.
99.
Topper.

* Nested if :- Whenever we define if else block, inside this is else block we define another if else block is called nested if else statement.

Syntax:-

```
if (condition)
{
  if (condition)
    { . . . . }
  else
    { . . . . }
}
else
{ . . . . . }
```

Flowchart:-



Eg:-

```
class nested
{ public static void main (String [] args)
  { int a=10, b=20, c=30;
    if (a>b)
    { if (a>c)
      { system.out.println(a); }
      else
      { system.out.println(c); }
    }
    else
    { if (b>c)
      { system.out.println(a); }
      else
      { system.out.println(c); } } }
```

Output:-

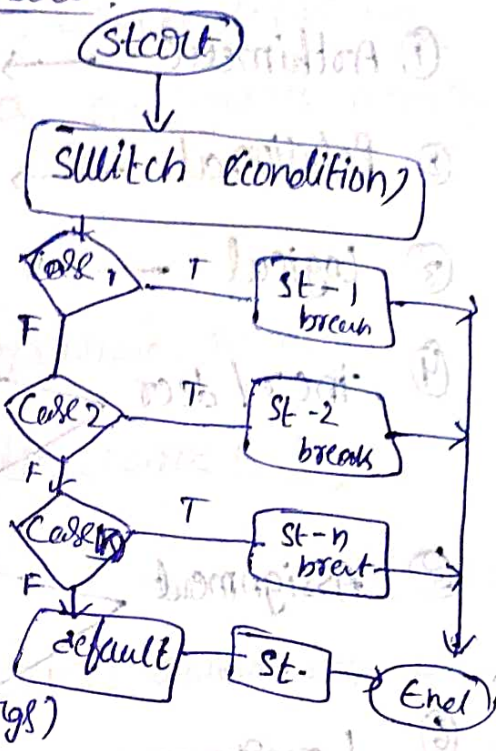
* switch statement :-

switch is a multiple choice decision making selection statement it used when we want to select only one use case out of multiple cases.

Syntax :-

```
switch (exp)
{ case 1 : state-1;
           break;
  case 2 : state-2:
           break;
  :
  case n : state-n;
           break; }
```

flowchart :-



Eg :-

```
public class code
{ public static void main (String[] args)
  { int day number = 2
    switch (day number)
{ case 1:
      system.out.println (x: "sun");
          break;
  case 2:
      system.out.println (x: "mon");
          break;
  case 3:
      system.out.println (x: "Tue");
          break;
  default:
      system.out.println ("enter 1 to 7"); }
```

output: 2
mon

↳ operators in java :-

→ operator is a symbol that is use to perform operation according to user requirement.

→ types of operators :-

① Arthimetic ⟶ +, −, *, %, /

② Relational ⟶ <, >, >=, <=, !=, ==

③ Logical ⟶ &&, ||, !

④ incr / decr ⟶ pre inc() / post inc() (++ exponent / exponent++)
⟶ pre dec() / post dec() (-- exponet / exp--)

⑤ Assignment ⟶ Simple (=)
⟶ compound (+=, -=, *=, etc)

⑥ Ternary ⟶ ? :

⑦ Bitwise ⟶ AND, OR, XOR, complement.

→ The unary operator are, incr / decr & Assignment

→ The remaining are Binary operators

→ The example has mentioned at the end of oop concepts

(7) Arrays in java :-

→ The collection of same datatype is called as Array.

→ Array is an object in java, which contains similar type of data in a contiguous memory location.

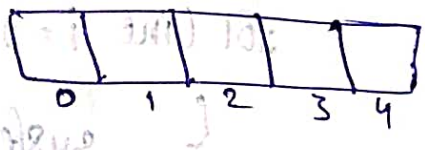
↳ Syntax → (i) datatype [ ] variable_name ;

(ii) datatype variable_name [ ] ;

Eg:- int a[ ]; / int[] a;

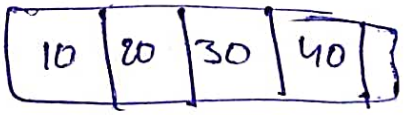
↳ Syntax for giving size. → datatype[] var_name = new datatype[5]

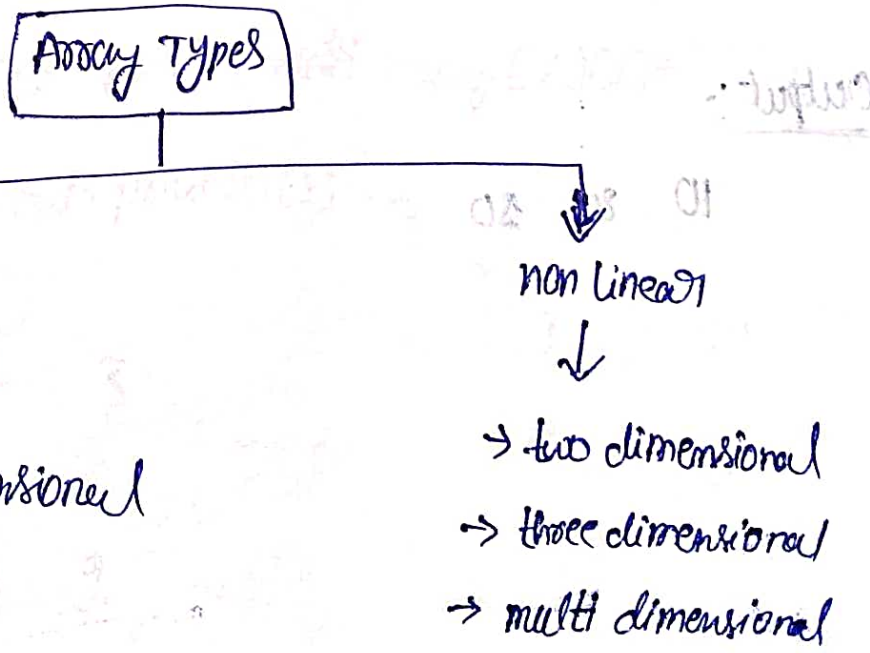
eg:- int [ ] a = new int[5]; ⇒



↳ Entering values into Array :-

int a[ ] = {10, 20, 30, 40}. ⇒





Eg:- class Demo

```
{ public static void main (String [] args)
  {   int a[ ] = {10, 20, 30, 40, 60};
      System.out.print(a[3]); }
}
```

Output:- 40

Eg:- class A

```
{ public static void main (String[] args)
  {  int a[ ] = new int [3];
     a[0] = 10;
     a[1] = 20;
     a[2] = 30;

     for (int i=0; i<3; i++)
     {  System.out.print(a[i]+" ");
     }
  }
}
```

Output:-

10 20 30

multidimensional Array :-

The Array which has more than one dimension Then it is called as multidimensional Array.

Syntax :- datatype name_Array [ ][ ][ ] = new datatype [ ][ ][ ];

Eg :-

```
public class MultiDimensionalArrayExample
{ public static void main (String [] args)
  { int [ ][ ] multiArray = new int [3] [3];
    int value = 1;
  for (int i=0 ; i<3 ; i++)
  { for (int j=0 ; j<3 ; j++)
    { multiArray [i] [j] = value;
      value ++; } }
  for (int i =0 ; i<3 ; i++)
  { for (int j=0 ; j<3 ; j++)
    { System.Out.print (multiArray [i][j]+" "); }
    system.out.println(); }
  }
}
```

Output :- 1 2 3
4 5 6
7 8 9

Array class methods in Java :-

→ sort()
→ equals()
→ copy of()

```
import java.util.Scanner;
import java.util.Arrays;
class A
{ public static void main (String [] args)
  { int a[] = new int [5];
    int a2 [] = new int [5];
    Scanner s = new Scanner (System.in);
    System.out.print ("Enter data in Array: ");
    for (int i=0 ; i<a.length ; i++)
    { a[i] = s.nextInt (); }
    System.out.print ("Enter data in Array:");
    for (int i=0 ; i<a2.length ; i++)
    { a2[i] = s.nextInt (); }
→   boolean b = Arrays.equals (a, a2);
    System.out.print ("They are equal" + b);
→   int a3[] = Arrays.copyOf (a,5);
    System.out.print ("Data in Array2");
    for (int i=0; i<a.length ; i++)
    { System.out.print (a3[i]+" "); }
→   Arrays.sort (a);
    for (int i=0; i<a.length ; i++)
    { System.out.print(a[i]+ " "); }
  }
}
```

(8) Strings in JAVA :-

→ The collection of characters is called as string.

→ It stores as string a = computer

Length – 8
index → 0 – 7.

a =

c	o	m	p	u	t	e	r
0	1	2	3	4	5	6	7

→ Java strings are objects that allows us to store sequence of characters which may contain alpha numeric values enclosed in doubled quotes ("Rama")

→ strings are immutable in java (can not change)

→ It contains methods that can perform certain operations on strings (concat(), equals(), length()... etc.)

Eg:- methods of strings.

length (str) – to get length

charAt (index) – to get value at particular index.

index of (String substring) – returns specified substring.

index of (char ch) – it return the specified char value index.

tolowercase() – converts string into lower case.

to uppercase() – converts string into upper case.

concat (str) – concates the specified strings.

compare to (str str-to-match) – for comparing two strings

replace (char old, char new) – to replace specified character

is empty() – It checks if string is empty.

trim() – It removes beginning & ending spaces of the string.

Creation of String

String literal — new keyword.

Eg: String a = "JAGA";

Eg: String a = new String("hi");
a.concat("Hello");

→

String a = "JAGA";
String a2 = a;
a2.concat("pogiri");

a — jaga
a2 — JAGA pogiri

→ Here the String concept is Immutable so can not modify directly the string so we are creating it.

Eg: Class A
{ public static void main (String [] args)
{ String a = "ankit";
System.out.println(a);
a = a.concat("kumar");
System.out.println(a);
}
}

Output: ankit
ankit kumar

* Constants in Java :-

whenever we use keyword "final" then the declared variable value is final.

Eg :- public ~~stat~~ class Constant

```
{
    public static final int i = 100;
    public static void main (String [] args)
    {
        System.out.println (i);
    }
}
```

output :- 100

* Comments in Java :-

→ The comments are used explain about the written code.

→ The comments can not be compiled by compiler.

Single line → // . . . . . //

multi lined → /* . . . . . */

* Identifiers in Java :-

→ Identifiers are the names given to variables, methods, classes, packages to all the user defined words.

→ must begin with (A-Z, a-z), underscore (_) or dollar ($) symbol only.

→ After first character may contain any thing

→ The first characters of every letters should capital.

→ should not use predefined words like int, float, class

* Inputs & outputs in JAVA :- [IO stream]

java.language
- System.in → Read data from keyboard
- System.out → Display the data on screen
- System.err → Display the error msg.

output → System.out.println("welcome");

System.out.printf → %d — int
%.f — float
%.d — double

Input :→ * Scanner — class → import java.util.Scanner;
* Buffer — class
↳ Input stream Reader → Import.java.IO.*;
↳ file Reader

Scanner class :-

next() → Read String → sc.next();

nextInt() → Read Integer → sc.nextInt()

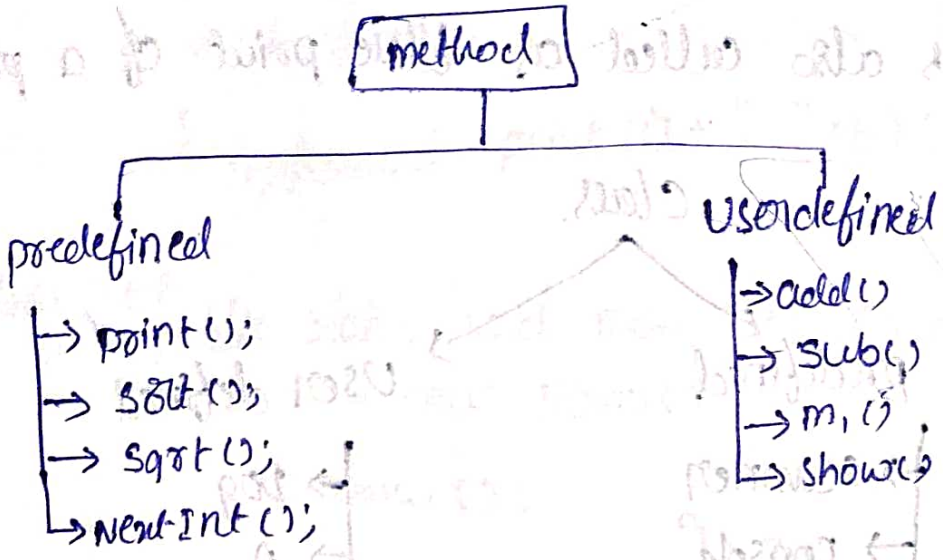
nextFloat() → Read Float → sc.nextFloat()

nextDouble() → Read double → sc.nextDouble()

⑨ methods in JAVA :-

→ method is a group / block of code which take input from the user, processed it & give output

→ method runs only when it called.



Syntax :-

returntype method_name (parameters)

{ // statement }

Eg :-

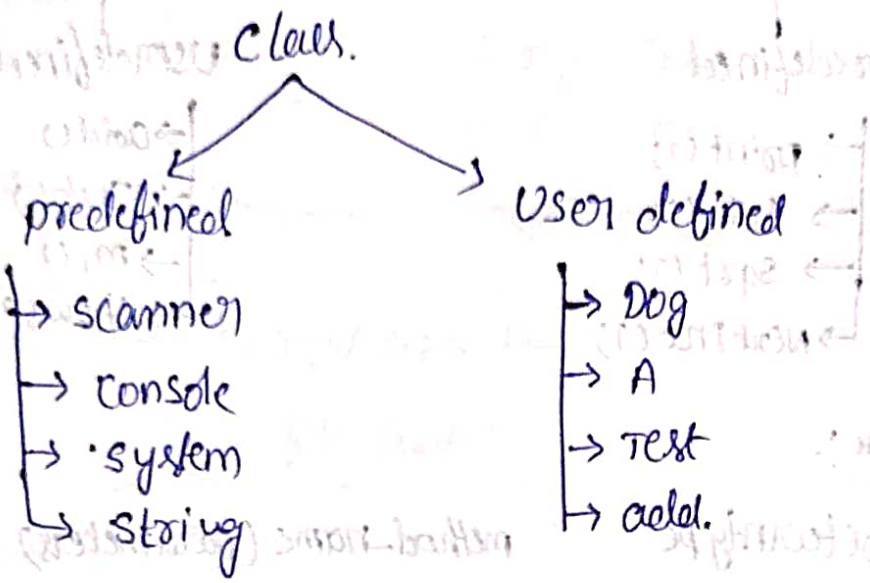
```
class A
{ public static void main (String [] args)
    { A a = new A();
      a.Disp(); }
  void Disp()
    { System.out.print("Learn coading"); }
}
```

O/P :- Learn coading

(i) Class In JAVA :-

→ Class is a collection of objects & it doesn't take any space on memory.

→ Class is also called as blue print of a program



→ The first letter of the class name should be Capital.

User defined Class :-

A class which is created by java programmer is called as user defined class

syntax → class class_name
{
// data
// method
}

(ii) object :- object is an instance of class that

Executes the class. once the object is created it takes up space like other variable in memory.

Syntax :- class_name obj_name = new class name();

Eg :-
```
class Demo
{ int a=10; String b="Jaya";
  void show()
  { System.out.print(a+" "+b); } }

class Test
{ public static void main(String[] args) {
  Demo r = new Demo();
  r.show();
} }
```

O/p :- 10 Jaya.

(12) static keyword :-

→ Here the keyword is static.

→ It only access the static Data.

→ Here we have * static variable
* static method
* static block.

Eg :-
```
class Demo
{ static int a=10;
  static void display()
  { System.out.println("static method"); }
```

```
static
{ system.out.println ("static Block"); } }

class staticDemo
{   public static void main (String args[])
    { system.out.println(Demo.a)
      Demo.display();
    }
}
```

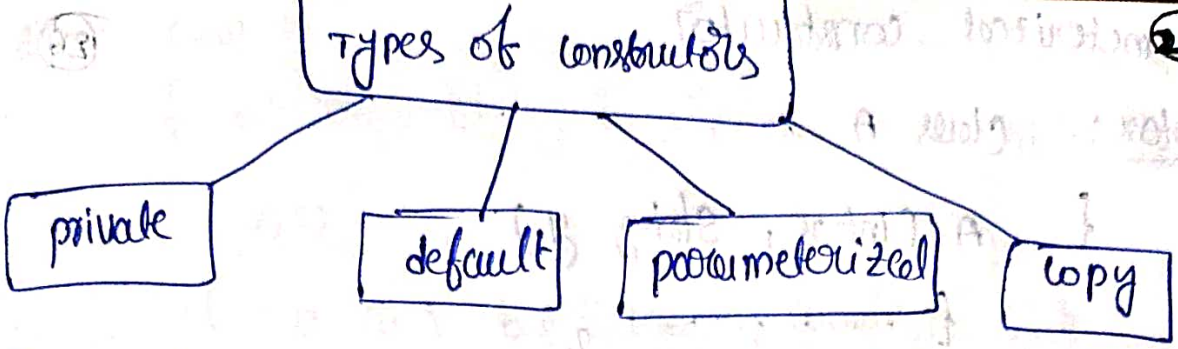
Op :- static Block.
Hi.
static Block.

(13) constructor IN JAVA :-

→ constructor is a special type of method whose name is same as class name.

note :- (i) The main propose of constructor is initialization the object
(ii) Every java class has a constructor by default
(iii) A constructor is automatically called at the time of object creation.
(iv) A constructor never contain any return type.

Syntax :- class class_name()
{
class name();
{ . . . .
. . . . }
}



(i) default constructor :- A constructor which does not have any parameter is called default constructor

syntax :-
```
class A
{   A()
    {.....}
}
```

Eg:-
```
class A
{ int a; String b; boolean c;
   A()
  { a=10; b="JAGA"; c=true; }
    void display();
  { System.out.print(a+" "+b+" "+c); } }
class B
{ A jaga = new A();
    jaga.display();
}
```

O/p:- 10 JAGA true.

(ii) parameterized constructor :- A constructor through which we can pass one or more parameters is called

parameterized constructor.

Syntax :- class A

{ A (int x, String y)

{ . . . . . }

}

Eg :- class A

{ int x, y ;

A (int a, int b)

{ x = a ; y = b ; }

void show ()

{ System.out.print (x + " " + y) ; } }

class B

{ public static void main (String [] args)

{ A j = new A (100, 200) ;

j.show () ; } }

O/P :- 100 200

(iii) copy constructor :-

→ whenever we pass object reference to the constructor then it is called copy constructor.

Syntax :- class class_name

{ class_name (obj ref)

{ . . . . . } }

Eg:- class A

```
{ int a; String b;
  A()
  { a=10; b="learn coading";
    System.out.println(a+" "+b); }
  A(A jaga)
  { a = jaga.a;
    b = jaga.b;
    System.out.println(a+" "+b); } }
class B
{ public static void main(String[] args)
  { A jaga = new A();
    A jaga2 = new A(jaga); } }
```

O/P:- 10 learn coading.

(iv) private constructor :-

In java, it is a possible to write a constructor as a private but according to the rule we can't access private members from out side of the class.

Syntax:- class class_name

```
{ private class_name()
  { ..... } }
```

Eg:- class A

```
{ int a; double b; String c;
    private A()
  { a=10; b=30.56; c="JAGA";
    System.out.println(a+" "+" "+" "+c);}
public static void main (String [] args)
  { A jaga = new A(); } }
```

Output:- 10 30.56 JAGA.

* destructor in java:-

→ A destructor is a special method that is automatically called whenever object goes to end.

→ A Destructor is used for de-allocate & free memory.

→ Their is no Destructor in java.

→ But Here we are using → finalize()

(14) This keyword in java:-

→ The This is a keyword.

→ It is used to refer current state / behaviour of the object / variable.

→ It is only for instance variables & current object.

Eg:- class A
{ int a = 10; ——→ Instance variable
void display()
{ int a = 20; ——→ local variable
system.out.println(a);
system.out.print(this.a); } }

O/P:- 20
10

(15) super keyword in Java :-

→ It is used to Access members of a super class.
→ It is used in inheritance concepts.

Eg:- class A
{ void show()
{ system.out.println("show() in class A"); } }

class B extends A
{ void show()
{ super.show();
system.out.println("show() in class B"); } }

class super2
{ public static void main(String[] args)
{ B b = new B();
b.show(); } }

O/P:- show in class A
show in class B

(16) Inheritance :-

→ when we constructed a new class from existing class in such a way that the new class access all the features & properties of existing class called inheritance.

→ In java 'extends' keyword is used to perform inheritance

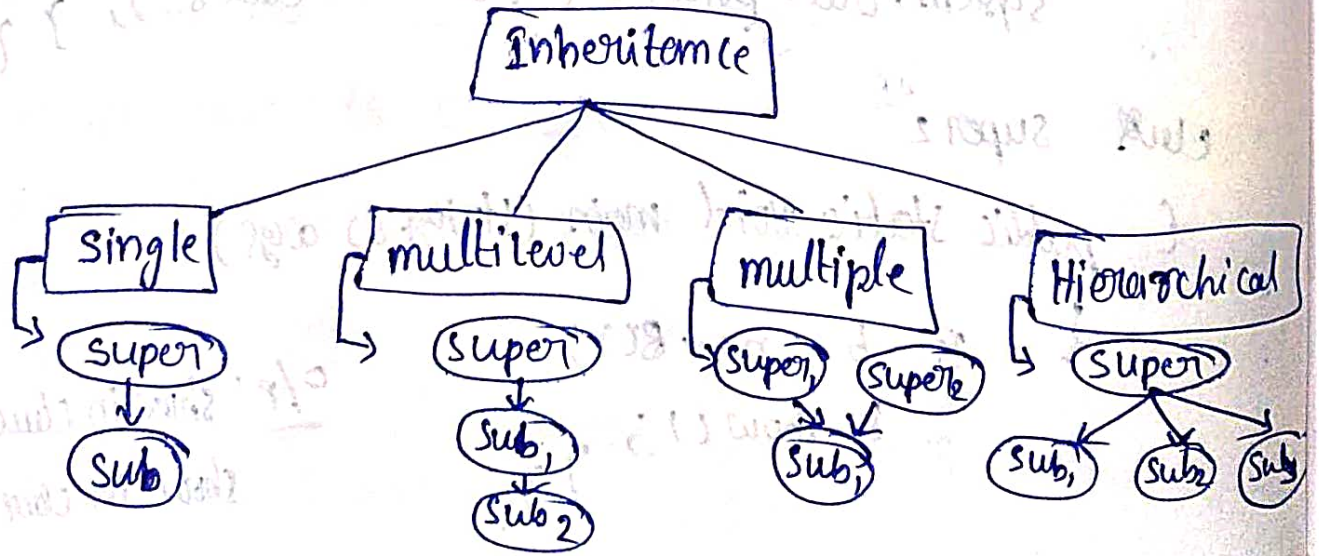
→ It provides code inheritance means reusability.

→ we can not access private members of any class without inheritance

→ A subclass contains all the features of superclass so, we should create the object of subclass.

→ method overriding only possible through inheritance.

syntax :-
```
class A
{ ..... }
class B extends A
{ ..... }
```



(i) single inheritance :- which consists one super class & only one subclass.

Eg :- class A
{ private int a = 10;
};
class B extends A
{ int b = 20;
public static void main (String[] args)
{
sysem.out.println (a+b); } };

syntax :-
class super
{ ..... };
class sub extend super
{ ..... };

o/p :- 30

(ii) multi level inheritance :-
In multi level inheritance we have one super class & multiple sub classes. but each sub class is a super class to next sub class.

Eg :- Class A
{ private int a = 10; };
class B extends A
{ private int b = 20; };
class C extends B
{ int c = a+b;
public static void main (String args[])
system.out.println (c); }

syntax :-
class super
{ ..... };
class sub, extends super
{ ..... };
class sub2 extend sub,
{ ..... };

o/p : 30

(iii) multiple inheritance :-

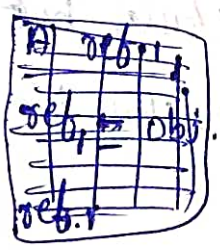
→ Their is nothing like multiple processed concept

→ JAVA doesn't support multiple inheritance.

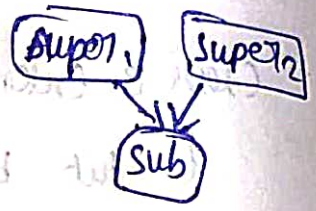
→ TO overcome this introduced. Interfaces concept

Eg:

```
interface A
{ int a = 10; }
interface B
{ int a = 20; }
class C extends A, B.
{ int c = A + B;
  public static void main (String args[])
  {
    C obj = new C ();
    System.out.println (c);
  }
}
```



O/P: 30

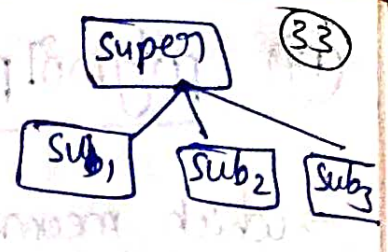


syntax:

```
class super1
{ ..... }
class super2
{ ..... }
class sub extends super1, super2
{ ..... }
```

(iv) Hierarchy inheritance :-

→ A inheritance which contain only one super class & multiple sub class & all sub class directly extends super class called hierarchical inheritance.



Eg:-

```
class A
{ void input()
  { system.out.println("Enter your name:"); }
}
class B extends A
{ void show()
  { system.out.println("my name is pogiri"); }
}
class C extends A
{ void show()
  { system.out.println("my name is jaga"); }
}
class Test
{ public static void main(String[] args)
    B r = new B();
    C r2 = new C();
    r.input();  ——— (1)
    r.show();  ——— (2)
    r2.input();  ——— (3)
    r2.display();  ——— (4)
  }
}
```

Syntax:-

```
class A
{ ..... }
class B extends A
{ ..... }
class C extend A
{ ..... }
```

O/P:

```
enter your name:
my name is pogiri
enter your name:
my name is jaga
```

polymorphism in JAVA :-

→ which means same object having different behaviours.

polymorphism → compile time polymorphism (method overloading)
→ Runtime polymorphism (method overriding)

(i) Compile time polymorphism :-

→ A polymorphism which is exists at the time of compilation is called compile time / early binding / static polymorphism

→ The perfect example to this is method overloading.

* method overloading :-

whenever a class contain more than one method with same name & different types of parameters called method overloading.

syntax :- return_type method_name (para$_1$)
return_type method_name (para$_1$, para$_2$)

Eg :- class A
{ void add ()
{ int a=10, b=20, c;

```
    c = a+b;
    System.out.println (c); }
void add1 (int x, int y)
    { int c;
      c = x+y;
      System.out.println (c);
void add1 (int x, double y)
    { double c;
      c = x+y;
      System.out.println (c);
public static void main (String args)
    { A r = new A ();
      r.add1 ();
      r.add1 (100, 200);
      r.add1 (50, 4532.147); }
```

O/P :- 30
300
9532.147

(ii) Run time polymorphism :-

→ A polymorphism which exists at the time of execution of program is called runtime polymorphism

→ A perfect example to that is method overriding.

* method overriding :-

whenever we writing method in super & sub class in such a way that method name & parameter must be same called method overriding.

Syntax :-
```
class A
{ void show()
  { ..... }
};
class B extends A
{ void show()
  { ..... }
};
```

Eg :-
```
class shape()
{ void draw()
  { system.out.println("can't say shape type"); }
};
class square extends shape
{ void draw()
  { system.out.println("square shape"); }
};
class Demo
{ public static void main(String[] args)
  { shape s = new square();
    s.draw();
  }
};
```

super.draw() — If you want to print both

O/P :- square shape.

(18) & (20) Abstraction & Abstraction classes in JAVA

* Abstraction in JAVA :-

→ A class contains the abstract keyword in its declaration is called abstract class

→ we can't creat object for abstract class.

→ It may or may not contain abstract method

→ It can have abstract & non-abstract method

→ To use an abstract class you have to inherit from sub classes.

→ If a class contain partial implimentation Then we should declare a class as abstract.

Eg :- abstract class shape {
    abstract void draw();}

class circle extends shape {
* @ override
  void draw() {
    system.out.println("Drawing a circle"); } }

class Rectangle extends shape {
  * @ override
  void draw()
    { system.out.println("Drawing a rectangle"); } }

public class main {
public static void main (string [] args) {

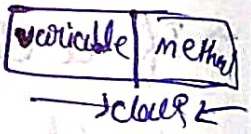
```
circle circle = new circle();
Rectangle rectangle = new Rectangle();
    circle.draw();
    rectangle.draw(); } }
```

(19) Encapsulation in java :-

→ Encapsulation is a mechanism through which we can wrapping the data members & member methods of class in a single unit called encapsulation.

Note :- → Declare the class variables as a private
→ Declare the class methods as a public

→ class is the best example of encapsulation



Eg :-
```
class A
{ private int value;
  public void set value (int x)
    { value = x; }
  public int get value()
    { return value; } }
class B
  { public static void main (String [] args)
    { A r = new A();
      r.set value (100);
      system.out.print (r.get value()); } }
```

O/P :- 100

(21) Interfaces in java :-

→ It is a concept mainly used in inheritance
→ It is especially used in multiple inheritance
To overcome some time of problems.
→ go & check multiple inheritence concept
I have already mentioned it.

* Example for operators in java :-

```
public class operator Example {
    public static void main (String [] args)
    { // Arithmetic operators
        int a = 10;
        int b = 5;
        int sum = a+b;
        int difference = a-b;
        int product = a*b;
        int quotient = a/b;
        int remainder = a%b;
S.O.println(a+b+ sum + difference + product + quotient
                + remainder);
    // Relational operators
boolean is equal = (a==b);
boolean is not equal = (a!=b);
boolean is greater = (a>b);
boolean is less = (a<b);
boolean is greater or equal = (a>=b);
```

bodeam is less or equal = (a <= b);

// logical operators

bodeam logical AND = (true && false);

bodeam logical OR = (true || false);

bodeam logical not = ! true;

* → s.o.p (is equal + is not equal + is greater + is less + is greter or equal + is less or equal

* → s.o.p (logical AND + logical OR + logical NOT);

// increment & decrement operators

int x = 5;

x++;

int y = 10;

y--;

* → s.o.p (x++ + y--);

// Assignment operators

int c = 10;

c += 5; // equivalent to: c = c + 5;

int d = 20;

d -= 5; // equivalent to: d = d - 5;

int e = 3;

e *= 4; // equivalent to; e = e * 4;

int f = 12;

f /= 3; // equivalent to; f = f / 3;

int g = 15;

g %= 4; // equivalent to: g = g % 4;

* → s.o.p (c + d + e + f + g);

// Bitwise operators

int num1 = 5; // 101

int num2 = 3; // 011

int bitwise AND = num1 & num2; // 001

```
int bitwise OR = num1 | num2; // 111
int bitwise XOR = num1 ^ num2; // 110
int bitwise complement = ~num1; // 11111010
```

* → S.O.P(bitwise AND + bitwise OR + bitwise XOR + bitwise complement);

```
// shift operators
int num3 = 8; // 1000
int left shift = num3 << 2; // 10000 (32 in Dec)
int right shift = num3 >> 1; // 100 (4 in decimal)
int zero fill right shift = num3 >>> 1;
                // 100 (4 in decimal) } }
S.O.P(num + left shift + right shift + zero fill right shift);
```

O/P :-

15	true	12
5	false	4
50	false	3
2	true	1
0	false	7
false	6	6
true	9	-6
true	15	32
true	15	4
false		4

— core java topics —

(22) Exception Handling in JAVA :-

→ An exception is unexpected / unwanted / abnormal situation that occured at run time called exception.

→ A perfect example is powercut to an unwanted situation.

→ If the exception takes place at any situation the rest of the program will terminate

→ To overcome the problem exception Handling is introduced.

→ there are some keywords in exception Handling.

(i) TRY
(ii) CATCH
(iii) THROW
(iv) THROWS
(v) FINALLY

(i) TRY → A Block of statements.

→ Here we write the code in which we are excepting the exceptions.

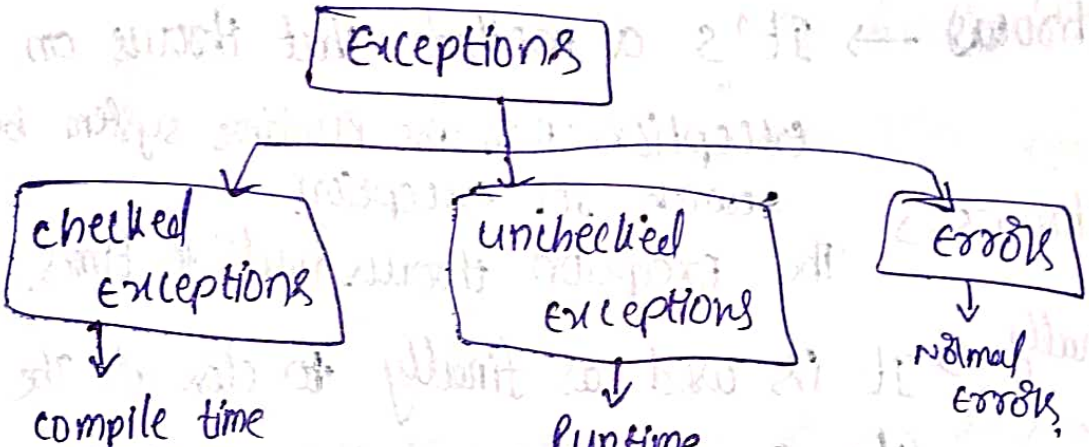
(ii) catch ⟶ A Block of statements.

→ whenever exceptions are occurred in try block then it Handles the exceptions.

Syntax:-

```
try
{ . . . . .
  . . . . . }

catch (exception e)
{ . . . . . .
  . . . . . }
```



Eg:-

```
class Ex
{ void display()
  { try
    { int a = 5/0; }
    catch (Exception e)
    { system.out.println(e); }
    system.out.println("Exception Handled..."); }
}
```

```
class Except
{ public static void main (String args[])
    {  Ex obj = new Ex();
       obj.display();
    }
```

O/p: java.lang.ArithmeticException: / by zero
Exception Handled...

* Throw → It's a method that throws an exception Then the Runtime system begins search for exception.
* Throws → The Exception throws multiple times.
* finally → It is used as finally to close all the files & also used to cleanup.

Example:-

```
class exception Handling Example {
  // method that throws an exception
   public static void check Number (int
   number) throws Exception {
      if (number<0)
   { throw new Exception("number is negative");
   }
   else { system.out.println("number is positive"); }}
```

```
public static void main (String[] args)
{   try { // calling the method that may throw an
    exception.
        checkNumber (-5); }
    catch (Exception e) {
      // Handling the thrown exception
      System.out.println ("caught exception:" +e.getMessage
                                                    );
    }
    finally {
    // code that will always execute
      System.out.println ("This is the finally block");
    }
System.out.println ("program continued after exception
 Handling");  }
}
```

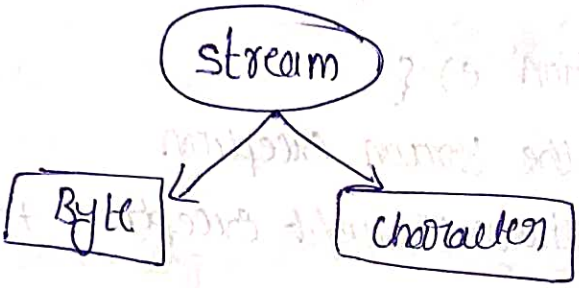
o/p :- caught exception : number must be non-negative
program continued after exception handling.

(23) file Handling in JAVA :-

→ file Handling defines how we can read & write data on a file in Java

→ In Java I/O package contains all the classes Through which we can perform all input & output operations in the file.

* Stream :- stream is a sequence of data on the basics of java.io package all the classes devided into two streams



* File handling methods :-

1. com Read()
2. com write()
3. create NewFile()
4. Delete()
5. exists()
6. length()
7. getName()
8. get Absolutepath()
9. MKdir()
10. List()
11. Read()
12. write()
13. rename TO()

* File handling classes :-

1. File
2. File Reader
3. File Writer
4. File Input stream
5. File output stream
6. Buffered Input stream
7. Buffered output stream

* operations of File :-

1. Create file
2. get file information
3. read
4. write

Eg:
```
import java.io.FileWriter;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
public class FileHandlingExample {
  public static void main(String[] args) {
    String content = "Hello, this is a simple eg of file Handling";
// writing to the file
try (FileWriter writer = new FileWriter("Example.txt"))
  { writer.write(content);
    System.out.println("successfully wrote to the file");
catch (IOException e)
  { e.printStackTrace(); }
// Reading from the file
try (BufferedReader reader = new BufferedReader(new
    FileReader("Example.txt")))
      { String line;
  while ((line = reader.readLine()) != null)
{   System.out.println(line); }
  catch (IOException e) {
    e.printStackTrace();
  }
}
```

O/P: successfully wrote to the file.
Hello. this is a simple eg of file handling.

(24) Abstract methods in JAVA :-

→ Abstract means unfinished.

→ A method which contain abstract modifier at the time of declaration it's called abstract method.

Syntax :- class A

{ abstract void main();

. . . . . .

. . . . . }

(i) abstract method can only be used in abstract class

(ii) it doesn't contain any body " { } " & allways ends with " ; "

(iii) Abstract method must be overriden in sub-classes otherwise it will also become a abstract class.

(iv) whenever the action is common but implementation are diffident then we should use abstract method.

Eg :- // abstract class programmer

abstract class programmer {.

// abstract method

public abstract void write HTML code();

// concrete method

public void debug code() {

System.out.print ln ("Debugging code..."); } }

```
// subclass Frontend Developer
class frontEnd Developer extends programmer {
// implementing the abstract method.
public void write HTML code() {
 system.out.println("writing HTML code for the web
   pag....."); } }
// main class to run the program, public class main
                                        {
 public static void main (String[] args)
  { // create an instance of Frontend Developer.
   programmer programmer = new frontend Developer();
// call the implemented method
   programmer.writeHTMLCode();
// call the concrete method
   programmer debug code(); } }
```

Output:-

writing HTML code for the web page.....

Debugging code.....

(25) Interface In JAVA :-

→ Interface is just like a class, which contains only abstract method.

→ TO achieve interface java provides a keyword called "implement".

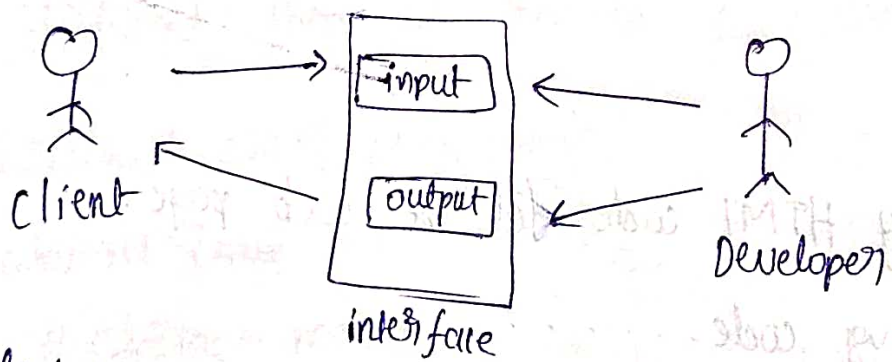
Syntax :- interface client
{
void m2();
}

Note :- ① interface methods are by default public & abstract

② interface variables are by default public + static + final

③ interface method must be overriden inside the implementing classes

④ interface nothing but deals b/w client & developer.



Example :-

```
public class FullTimeEmployee
implements Employee {
    private String name;
    private int EmployeeId;
```

```
private double baseSalary;
public FullTimeEmployee(String name, int employeeId,
    double baseSalary)
{ this.name = name;
  this.employeeId = employeeId;
  this.baseSalary = baseSalary; }
public void displayEmployeeDetails()
{ System.out.println("Employee ID: "+employeeID);
  System.out.println("Base salary: "+baseSalary); }
public double calculateSalary()
{ // for simplicity, just return the base salary.
  // In a real application, you might add bonuses or other
  benefits.
      return baseSalary; }
public static void main(String[] args)
  { FullTimeEmployee employee = new FullTimeEmployee
("John Doe", 12345, 50000.00);
  employee.displayEmployeeDetails();
  System.out.println("Calculated salary:"+employee.
                    calculateSalary()); } }
```

Output:-

```
Employee ID: 12345
Name: John Doe
Base salary: 50000.0
Calculated salary: 50000.0
```

(26) packages in JAVA.

→ A package arrange number of classes, interfaces & sub-package of same type into a particular group.

→ package is nothing but folder in windows

→ There are many types in packages.

Types

pre-defined

- java.lang
- java.util
- java.io
- java.applet
- java.awt
- java.net
- java.sql

Userdefined

- package p1
- package cold
- package mypack
- package jaga.

* Access modifiers in packages.

Access modifier	within class	with in package	outside package by subclass	outside package
private	✓	✗	✗	✗
default	✓	✓	✗	✗
protect	✓	✓	✓	✗
public	✓	✓	✓	✓

Syntax of packages :-

→ package package_name;

eg:- package com.example.mypackage;

→ import package_name;
{ . . . . . .
. . . . . }

eg:- Step1 :- create a package & a class.

File: com/ex/mypackage/Greeting.java

```
package
    com.example.mypackage;
public class Greeting {
    public void sayHello() {
System.out.println("Hello from Greeting class"); } }
```

Step2 :- create a main class to use the package.

File: main.java

```
import com.example.mypackage.Greeting;
public class main {
    public static void main(String[] args)
{ greeting greeting = new greeting();
    greeting.sayHelo(); } }
```

Output :- Hello from greeting class.

(27) Multi Threading :-

→ Multithreading in Java is a proces of executing two or more threads simultaneously to maximize CPU utilization.

→ Each thread runs independently, & multiple threads do not allocate separate memory areas, saving memory & Reducing context switching time.

Types of Threads :-

* User Threads :- The main Thread created when an application starts.

* Daemon Threads :- used for Background tasks, such as cleaning up the application

Creating Thread :-

→ Extending the thread class :-

Eg:-

```
Class multithreadingDemo extends Thread {
    public void run() {
        system.out.println("my thread is in running state");
    } }
public static void main(String args[]) {
    multithreadingDemo obj = new multithreadingDemo();
    obj.start();
} }
```

→ implementing the Runnable interface :-

Eg: class Hello implements Runnable {.

```
    public void run() {
        for (int i=1; i<=200; i++) {.
            system.out.println("Hello"); } } }
public class main {
    public static void main (String [] args) {.
        Thread t1 = new Thread (new Hello());
            t1.start();
    } }
```

O/P :- Hello prints 200 times.

* Thread life cycle :-

↳ state of a Thread :

(i) New : A thread that has not yet started.

(ii) Runnable : A thread executing in the java virtual machine.

(iii) Blocked : A thread waiting for a monitor lock.

(iv) waiting : A thread waiting indefinitely for another Thread.

(v) Timed waiting : A Thread waiting for a specified time

(vi) Terminated : A Thread that has exited.

⇒ multi threading in java allows for efficient utilization of cpu resources by executing multiple threads simultaneously it is achieved by extending the Thread class / implementing the Runnable interface. understanding the life cycle of a thread & the advantages of multithreading is crucial for effective multithreading in JAVA.

Eg:- package demotest;

public class GuruThread1 implements Runnable {

public static void main (String[] args) {

Thread guruThread1 = new Thread ("Guru1");

Thread guruThread2 = new Thread ("Guru2");

guruThread1.start();

guruThread2.start();

system.out.println("Thread names are following :");

system.out.println(guruThread1.getname());

system.out.println(guruThread2.getname()); }

@Override

public void run() {

} }

output:- Thread names are following :

Guru 1

Guru 2