# JAVA.

Introducted in 1992.

- Have OOPS concept.
  - OOPS ( Object Oriented Programming
    - Class
    - object
    - Encapsulation
    - Polymorphism
    - Inheritance
    - Abstraction.

Java is an open source and have all principles of OOPS.

## Class and Objects

- Objects is something that exists, have something properties, and does some action.
- Class is something that does not exist. and it is a collection of objects., variables or methods.
- Class itself doesnot have any memory itself. and its Objects is a way to have the memory.

Java is platform independent.

platform indicates windows, linux, macos. i.e, anything you develop in any platform runs in other platforms.

Java executes a .class file that consists of byte code. For every OS, there is a jvm (Java Virtual Machine) that converts the byte code to machine code.

* To get a jvm, you have to install a JDK (Java Development KIT).

* Rules of Java:

Classname:
- Starting letter must be Capital. Ex: Scanner, Arraylist.

Methodname:
- Starting letter must be small and it have two words, the subsequent word letter start with Capital and then ends with (). Example: get(), getMessage(), printStackTrace().

Variable name:
- A to Z or a to z or _ or $
Example: int 123total; => wrong
int _total123; => Correct.

* Java is case sensitive.
* Reserved Keywords can't be used. (Ex: for, while, break, if, int ... etc.
* Spaces are not allowed between the variable name.
* Every line of code ends with a semi-colon (;).

* IDE (Integrated Development Environment).
  - Eclipse.
  - My Eclipse
  - Netbeans; etc..

* Text Editor:
  - Notepad
  - notepad tt
  - editplus
  - sublime.

* How to compile:

Syntax:

javac Test.java (file_name).

Example:

i) javac Test.java (one file)
ii) javac first.java second.java (multiple file).
iii) javac *.java.
- One we compile .class file will be generated.

* How to run:

Syntax:

java Classname. / Ex: * java Test.
- You can run only one class at a time.

* How to check the java libraries & collections in cmd?

javap java.lang.Object;
javap java.util.Scanner;
} use javap followed by what you want to check.

In a java file, one one class can contain a main method.
i.e. public static void main (String[] orgs)

## Public and Private Keyword.

Private: limited to that class.

Public: Anyone can access.
: Class name and file name must be same.

## Static Keyword:

- It is a predefined memory and doesnot depend on object. one can be accessed with a class name.

Object must be accessed by programmer.

Without .Static, a program cannot be executed by the jvm.

main: Start of program execution.

Static can be applied to:
- variable
- method
- class
- block
- object.

* How to create object:

Syntax:

Classname obj = new Classname();

Ex:

Person rahul = new Person(); // object reference or pointed to an object.

Here, the new keyword allocates the memory. In Java, the memory is allocated at runtime.

* Importing a Package:

Syntax:

i) import java.util.*

ii) import java.util.TreeSet.

To run once in a program:

iii) java.util.TreeSet ts = new java.util.TreeSet();

* Output method in java:

Syntax:

System.out.println() ;

Ex:

```
System.out.println("Welcome");
System.out.println(variable_name);
System.out.println("Variable value =" + variable_name);
```

* Types of variables:

i) primitive variable.

Ex: int a = 10;

↓

primitive variables.

ii) non primitive variable.

Ex. Person rahul = new Person();

↓

non-primitive variable.

* Primitive variables are of three types:
  - i) Instance variable (Inside a class, outside a method, depends on object)
  - ii) Static " (Doesnot depend upon object and is common for object).
  - iii) local " ( Inside a class & inside a method).

* Memory areas of jvm:
  - method (static)
  - heap (object is created inside heap area) (Instance variables).
  - stack
  - pc
  - native.

* Static variable depends on the class name.

* local Variable => declared inside a class & inside a method.
  => Must be initialized.

* Instance Variable => If not initialized, it gives the default values.

---

jvm -> It converts the byte code to machine code.

## Data Types:

- special symbols are known to the machine directly.

## Primitive Data Types:

- numbers
  - integers
    - byte 1B
    - short 2B
    - int 4B
    - long 8B
  - decimals
    - float 4B
    - double 8B.
- characters.
  - Char 2B.
- boolean
  - true/false (1bit)

## Operators & Assignments.

i) Increment / Decrement.

- pre-increment : ++y
- post-increment : y++

- pre-decrement : --y
- post-decrement : y--

Example.

int x=4, y;

y = ++x; x=5 and y=5.

y = x++; x=5 and y=4

# In post, first it will assign and then do the increment / decrement.

* Arithmetic Operators:

( +, -, *, /, % )

Max ( man-type, type of a, type of b).

byte + byte = int
byte + short = int.
char + char = int. (Characters are based off ~~Unicode~~ ASCII values.)
long + float = float
double + char = double.
ASCII - A(65) a(97).

* Relational Operators:

( <, >, >= , <= )

* Equality Operators;

( == , != )

* Bitwise Operators:

AND (&), OR (|), XOR (^).

---

Printf conversion characters:

b = boolean

d: decimal integer (byte, short, int, long)
f: float, double
c: character, Capital C will uppercase the letter.
s: String, Capital S will uppercase all the letters.
h: hashcode
n: newline ( use %n instead of \n for greater compability

* Type Casting:
  - Converting one datatype to another datatype.

i) Implicit Type casting (Done by the compiler).
ii) Explicit Type casting ( Done by the programmer).

* Assignment Operators:

i) Simple Assignment Operators a = 20.
ii) Chained " " a = b = c = d = 20.
iii) Compound " "

Example:
Compound assignment: ( t = , - = , * = , / = ). Common:
a = 20.
a += 30 , means. a = a + 30. => a = 50.

Also there are, ( %= , &= , |= , ^= ).

* Evaluation Order.

* -> / -> + -> -

* logical Operators.

&& , || , ! op = (compound Assignment Operator &= , |= , ^= )

* Ternary Operator.
Syntax: variable_name = (condition) ? value_if_true : value_if_false;

Example: String name = case.equals("uppercase") ? "John" : "john";

* Control Statements.

i) Simple if.
ii) if else.
iii) nested if else.
iv) if else ladder.

Simple if:

```
if (condition)
{
    statement ;
}
```

if - else

```
if (condition)
{
    Statement ;
}
else
 {
    Statement ;
 }
```

Nested if - else:

```
if (condition-1)
{
    if (condition-2)
 {
 }
 }..
```

* Method.

- A set of instructions written to perform a task.

Types:

i) Pre-defined (already developed by developers).
ii) User defined

Every method has two properties:
i) return type
ii) parameters.

Based on these properties. the methods are classified into four types.

i) Method without return type & without parameters.
ii) " " " " " with parameters.
iii) " with " " " without " "
iv) " " " " " with " .

* Switch Case

- Switch case statement is used when we have number of options or choice, and we may need to perform a different task for each choice.

Syntax:

```
Switch (variable or an integer expression)
{
    Case Constant:
        // Java code;
    Case Constant:
        // Java code;
    Default:
        // Java code;
}
```

Example:

```
int num = 2;
Switch (num)
{
    Case 1:
        sout();
    Case 2:
        sout();
    default:
        sout();
}
```

Note: It is important to have break statement after a switch statement because it will execute the following statements after that true case.

* Loop.

- While loop

Syntax.

```
initialization ;
while (condition)
{
    statements ..
    i++ ;
}
```

- do while loop.

```
initialization ;
do
{

    Statement - 1
        :
        :
} while (condition) ;
```

* For Loop

Syntax:

```
for (initialize ; condition ; increment/decrement/logic)
{
    statements;
}
```

* Output in java:

```
java.io.*
java.lang.*    //imported by default
java.util.*
```

(javap java.lang.System

```
java.io.PrintStrem
 class PrintStream
{
    void println();
}
```

```
java.lang.System.
class System
{
    static PrintStream out;
}
System.out.println().
```

# Input in java

java.io.InputStream
class InputStream {
}

java.lang.System
class System
{
    static InputStream in;
}.

System.in : It contains the input object in String format.

Scanner : It converts the string format Object to required format.

java.util.Scanner.
class Scanner
{
    int nextInt();
    float nextFloat(),
    long nextLong();
    double nextDouble(),
    String next();
}

* ## Polymorphism:

Poly = Many
Morphism = forms or states or properties.

Have two concepts:
i) Method Overloading.
ii) Method Overriding.

Method overloading
- Method can be written with ~~same~~ different type of parameters, different number of parameters ; different order of parameters.

Method Overriding
- Method overriding can be written with same (type of parameters, number of parameters and order of parameters). Always in two classes that have a child-parent or IS-A relationship.

Method Overloading.
- May have different return types.
- May have different access modifiers (private, protected, public)
- May throw different exceptions.
- Usually in a single class but may also be used in a child class.

## Array:

- Collection of homogenous elements.
- Collection of similar elements.

How to declare an array:

i) datatype ar[]; or datatype []ar; or datatype []ar;

Syntax:

```
int a[] = new int[5];
    or,
int a;
a = new int[5];
```

Example:

```
int x[];
int[] x;
int [] x;
```

* Two Types:

i) Single Dimensional array : a[]
ii) Two dimensional array: a[][]

* For each loop:

Syntax:

```
for (datatype variable : array/collection) {
}.
```

* Constructor:

```
classname obj = new classname();
                    |
                    ↓
                Constructor();
```

- It takes part in object creation and it initialise the variables.

Properties of a constructor:

i) Constructor name is same as classname.
ii) Constructor has only one property i.e. parameter.
iii) Constructor can't have any return type i.e not even void.
iv) Constructor gets executed when object is created i.e. once per object creation.

Types

i) Zero argument constructor (without parameters)
ii) Parameterized constructor (with parameters)

# * Constructor in Java.

- What is Constructor in Java ?
  - A constructor is a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required number variables.
  - We create a constructor to initialize an object.
  - They have the same name as the class but do not have any return type.
  - It can be used to set initial values for object attributes.
  - At the time of calling a constructor, the memory is allocated for the object.
  - Each class in java has a constructor. Even if you don't create one Java implicitely calls the constructors with all the data values set to 0 or default, which is called as default constructor.

  - A constructor is called when an object or an instance is created. It is used to assign the values to the data members of the same class.

- Rules for using Constructors.
  - The name of the constructor should be the same as that of class name
  - A constructor cannot be declared as final, static, Synchronized or abstract.
  - It cannot have an explicit return type.
  - A constructor can have an access modifier to control the access.

* Types of Constructors.

  i) Default Constructors
  ii) No-args Constructors.
  iii) Parameterized Constructors.

i) Default Constructor.

- A constructor with no arguments.
- If we do not create a constructor of a class, java creates a default constructor with data members with their default value types like 0, null, etc.

Example:

```
pubic class edu {
        String name;
        edu() {
            this.name = "Arjun";
        }
        public static void main(String[] args)
    {
        edu obj = new edu();
        System.out.println(obj.name);
    }
}
```

ii) No-args Constructor.
- More like default constructor.

Example:

```
Class edu
{
    public edu()
    {
        System.out.println("Hello World!");
    }
    public static void main(String args[])
    {
        new edu();
    }
}.
```

iii) Parameterized Constructors.
- Constructor with arguments or parameters.

* To call a constructor inside a constructor, use this( ).
* With constructors, it is not a good practice to call other methods or setters method.
* In Java we have this() and the super() call.
* Use this() to call a constructor from another overlooded constructor in a same class. It must be the first statement in a constructor. It's used with constructor chaining, in other words when one constructor calls another constructor, and helps to reduce duplicate code.
* The only way to call a parent constructor is by calling super(). This calls the parent constructor. Must be the first statement in each constructor.
* The Java Compiler puts a default call to super() if we don't add it, and it is always the no-args super which is inserted by compiler (constructor without arguments).

# SQL : (Structured Query language)

Parts:

DDL : Data Definition language.

Create : To create a table
Alter : To change or manipulate or modify datatypes in table

Parts of alter:

Modify :
Add :
rename :
drop :

} Works on Column.

remame : Renames the table
drop : It drops the table

DML : Data Manipulation language.

insert : Single row insertion & multiple row insertion.
update : Changes the existing column or row value. It can be entire column, particular row and multiple row.

delete : Works on entire table, single row, multiple row.

Select : All rows, particular single row and multiple row.

* Rules of SQL:

i) Duplicate table names are not allowed
ii) Duplicate columns in a table are not allowed
iii) Every command or query ends with a semicolon.
iv) The character values can be given in single quotes itself.
v) SQL is case insensitive. Whatever case you write in, it will take UPPER class case.

* Datatypes: Classification of information.

Numbers:

number (size) :

- It can store without decimal values.
- The size limit is 38. The size indicates the number of digits.

ex: id number (10).

number (precision, scale):

- It can store the decimal values.

ex:

percentage number (4, 3)

7.134

Total four digits and decimal is placed after 3 numbers from the back.

## Characters:

Char (size):
- It can store only characters

ex: name char (30)
- The size is limited to 2000.

Varchar (size) / varchar2 (size): ( varchar2 supports both SQL & PL/SQL
- It can store alpha numeric values also

ex: rollno varchar (20).

- The size is limited to 4000 characters.

* Date: To store the system date.
* Time. To store the system time
* blob: To store the information in binary object ( like files, images).
* Clob: " " " " " character object ( " " , " ).

* DDL

- Create :

  Syntax:

```
create table <table_name>
(
    <column_name1> <data-type1>,
    :
)
```

  Example:

```
Create table jnit ( id number(10), name varchar(20),
                    salary Number(10) );
```

* To check the structure of table in command line.

```
desc table_name
```

- alter (columns)
  - modify)

Syntax.

alter table <table_name> modify ( <column_name> <new data-type

Example:

alter table jnit modify (id varchar(10));

- add

Syntax

alter table <table_name> add (<column_name> <data_type>),

- rename.

Syntax:

alter table <table_name> rename column (<old-column> to <new-column>);

Example:

alter table jnit remone column doj to joining;

- drop.

Syntax:

alter table <table_name> drop column <col_name>;

Example:

alter table jnit drop column joining;

To Drop multiple columns, remove column.

alter table jnit drop (address, email);

- Rename (Table).

Syntax:

rename <old_table_name> to <new_table_name>;

Example:

rename jnit to jnitinc;

- Drop (Table):

Syntax:

drop table <table_name>;

Example:

drop table jnitinc;

Inheritance.
- Aquiring the properties of one class to another class.

extends keyword is used.

There is
parent / base / super class &
child / derived / sub.

- You have to create the object for sub class as object for main class is automatically created.

Types.

i) Single level inheritance.
ii) Multi level inheritance.
iii) Hierachical level inheritance.

Super : super class object or it points to the super class variables, constructors, etc.

Final keyword :
- used to declare variable, class, or methods the does not have to be changed.

The keyword super is used to access / call the parent class members (variables and methods)
The keyword this is used to call the current class members (variables and methods).
This is required when we have a parameter with the same name as the instance variable.
The object class is automatically inheritated in any class along with its various methods.

* DML: (Data Manipulation language):

1) Insert

- All columns Single row:

insert into <table_name> values (<value1>, ....., <valueN> ).

Example:

insert into jnit values (1023, 'bob', 4000);

- All columns multiple rows:

insert into <table_name> values (& <column_name>, ....., <column-N>),

Example:

insert into jnit values (&id, '&name', &salary).

Note: You '/' to call the previous command in command line.

- Particular columns Single row.

insert into <table_name> (<col_name>, .... <col_N>) values (<value1>, ...., <valueN>)

Example:

insert into jnit (id, name) values (1029, 'arjun');

- Particular Columns multiple rows:

Syntax:

insert into <table_name> (<column_name>, ... <columnN>) values (<&column_name> ...
<&columnN);

Example:

insert into jnit (id, nome) values (&id, '&nome');

## * UPDATE.

- Entire Column Update:

Syntax:

update <table_name> set <col_name1> = <value1>, <col_nameN> =
<valueN>;

Example:

update jnit set doj = sysdate; -> provides the system date.

update jnit set doj = '25-aug-2020';

- Particular row ~~update~~ single column update :

Syntax

Update <table_name> set <col1_Name> = <value1> where condition ;

Example:

Update jnit set salary = 2000 where id = 1026 ;

update jnit set salary = 32000 , name = 'ramesh' where id = 1026 ;

↓

- Particular row multiple column update :

Syntax:

update <table_name> set <col1_name> = <value1> , <colN_name> = <value N>
where condition ;

- Multiple rows single column update .

Example :

update jnit set salary = 32000 where id in (1029, 1030) ;

## * Delete.

Entire table:

Syntax

delete from <table_name>

- Particular record delete:

Example:

delete from jnit where id = 1038.

- Multiple record delete.

Example:

delete from jnit where id in (1029, 1026);

## * Select.

Example:

```
Select * from jnit ;
Select * from jnit where id = 2025;
Select id, salary from jnit ;
Select id, salary from jnit where id in (1023, 1032);
```

## * Method Overriding :

- Method name can be same with same type of parameters or some order of parameters or some number of parameters. Must have the same return type. Must not have a lower modifier but may have a higher modifier. Must not throw a new or broader checked exception.

• Rules of Overriding :

i) Separate classes are required for overriding.

ii) Inheritance relationship is mandatory.

iii) separate objects are required for overriding.

- Constructors and private methods cannot be overridden. Methods that are final can't be overridden.
- Use @Override immediately above the method definition.
- We can't override static methods, only instance methods.
- A subclass can use super.methodName() to call the superclass version of an overridden method.

Class Casting :

- Converting one class object to another class.

Types :

Upcasting :

- A super class reference storing the sub class object.

Eg :

```
Car x = new audi();
```

Car → superclass
x → sup class reference.
audi() → sub class.

## * Operators in SQL.

like operator: ( _ , for single characters ) (%, for multiple characters)
in and not in operator.
between and not between operators.
and operator.
or operator.

## * Group By Clause:

- max()
- min()
- sum()
- avg()
- count()

} Column name goes inside ().

- when using group by, you cannot use where by.
- having function or keyword goes only with group by.
- you can select the column name thats used in group by.

## * Order By clause.

select * from jnit order by column-name;

* Contrains.

i) Unique - does not allows duplicate but allows null values.
ii) not null - it doesn't allow null values but it allows duplicate values.
iii) check - based on condition it will allow the value.
iv) primary key - It is the combination of unique and not null. It does not allow null values and duplicates.
v) foreign key.- must have primary key in the parent table.

Example:

```
Create table dept ( . . - - . . . . . - - - ),
foreign key (empid) references employee(empid) ),
        ↓                       ↓
      [redacted]              parent table.
```

* TCL :

- Transaction Control language.

rollback : Goes to the previous state or undo the previous single or multiple commonds.
: Happens inside the primary memory.

commit : It will save the commonds in the secondary memory.

* Memory in SQL

Primary : until we write comit or we close the window, it won't be saved inside the secondary memory.

Secondary :

* Abstract Class in Java :

- Collection of abstract method or concrete methods.
- Works like rules for the subclass.

• Abstract method : the method which doesn't have any implementation or body.

• Concrete methods :
  - It supports hierarchical interface.
- For abstract class, we can't create the object.
- Abstract method in ~~subclass~~ super class must be used in sub class.
- Abstract class exists to be extended, they cannot be instantiated.

* Interface Class in Java.
- It assumes that every methods inside this interface class are abstract. So, you cannot have concrete methods. Also all methods have public access.
- You use the keyword, implements, instead of extends to implement the methods in interface class.
- It supports multiple inheritance.

• Use abstract classes to create the abstract methods that are or will be common to another classes with different values or properties.

• If a class implements an interface then all the abstract method must be overrided.
• If we are overriding the interface abstract method, then all the overrided methods must be declared as public.
• We cannot create object for an interface but we can create the reference.
• Interface variables are by default public, static & final.
• Interface cannot implement another interface, but it can extend another interface.
• We cannot write a constructor in interface class.

* Query within a Query is known as subqueries

Types
- Single Row Subquery ( = ) . ( all )
- Multiple Row subquery. ( any ) ↓
  ↓ < all
  < any > all
  > any = all
  = any

* Inner Join
  - Common Records. between tables.

* left Join. / Right Join
  - Displays common records between tables and then the remaining records on the left table or right table

* HTML.
  - Used to develop web pages.

Web pages are of two types.
i) Static Pages -> Values does not change
ii) Dynamic Page -> values changes.

HTML Tags.

i) Headings : h1, h2, h3, h4, h5, h6.
  - attribute: align = left / center / right

ii) Scrolling : <marquee> </marquee>.
  - attributes: bgcolor = color
  direction = left / right / up / down
  scrollamount = value
  behaviour = scroll / alternate / slide.
  loop = value
  width = value.
  height = value.

iii) Paragraph :  
  attributes - align = justify.

iv) Font : <font> </font>.
  - attribute : size, colur, ....

v) image : <img src = " " />
  - attribute : height, width, align, ....

vi) iframe : copy the embedded command.

vii) table.

<tr> => rows.

<td> => column.

- attribute: border, align, width, height, color, cellspacing, cellpadding.

viii) <a href = "source/url" />

- attribute: target = "__blank" -> goes to new tab.

ix). <form>

- attributes: action, method.

x) <input>

- attribute: type, name, value, placeholder.

xi) <select>

- attribute: name, ....

xii) <option>

- attribute: value, ....

xiii) <textarea>.

- attribute. cols, rows.

* Inner Classes.
  - A class written within a class is known as inner class.

Types
  i) Non Static inner class
  ii) Static inner class

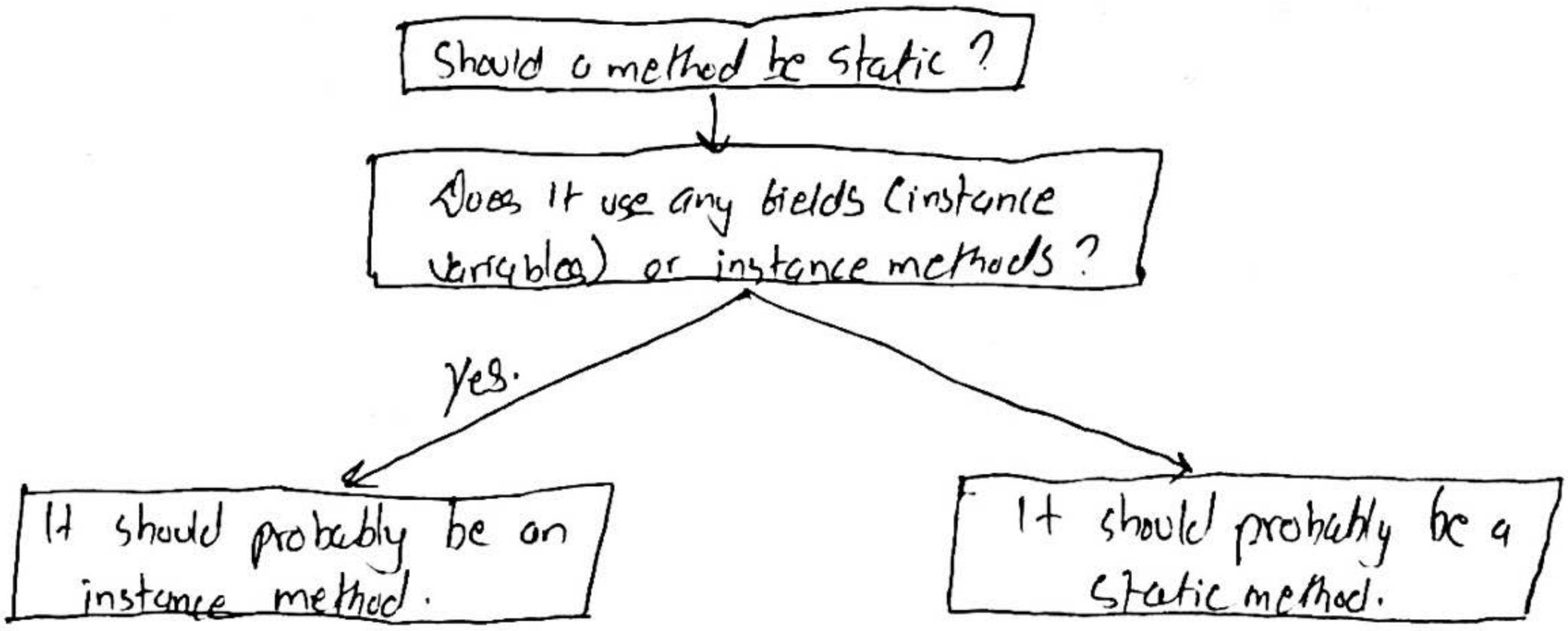
Non-static inner class are classied into three types
i) Member inner class
ii) Local inner class
iii) Anonymous inner class.
  - It do not have any name
  - They are created during object creation followed by opening and closing {};after the constructor.

* Interface Class in Java
i) A abstract class which contains all the abstract methods, instead of using abstract class, we can use interface
ii) It is a pure abstract.
iii) In Interface, all the methods are by default abstract and public.
iv) A class must implement an interface, a class cannot extend an interface.
v) Interface supports multiple inheritance.
vi) If a class implements an interface, then all the abstract methods must be overrided.
vii) If we are overriding the interface abstract methods, then all the overricled methods must be declared as public.
viii) We can't create object for an interface but we can create the reference.
ix) Interface variables are by default public static final.

* Static Methods.
  - Static methods are declared using a static modifier.
  - static methods can't access instance methods and instance variables directly
  - They are usually used for operations that don't require any data from an instance of the class (from 'this') where this keyword is the current instance of a class.
- In static methods, we can't use this keyword.
- Whenever you see a method that does not use instance variables that method should be declared as a static method.
- Can be accessed with class name. Eg. ClassName.methodName();

* Instance Methods.
  - Instance methods belong to an instance of a class.
  - To use an instance method we have to instantiate the class first usually by using a new keyword.
- Instance methods can access instance methods and instance variables directly
- Instance methods can also access static methods and static variables directly.

* Static or Instance method?



Static Variables.

- Declared by using the keyword static
- Static Variables are also known as static member variables.
- Every instance of the class shares the same static variable.
- If changes are made to that variable, all other instances will see the effect of the change.
- Static variables are not used very often but can sometimes be very useful. Useful when methods in the class have some common variables with same values. like in Math class, pi can be a static variable.
- Another example is when reading user input using Scanner, we will declare scanner as a static variable, that way static methods can access it directly.

Instance Variables.

- Instance variables are also known as fields or member variables.
- Instance variables belong to an instance of a class.
- Every instance has it's own copy of an instance variable.
- Every instance can have a different value or state.
- Instance variables represent the state of an instance.

What is instance of a class?

- An object is an instance of a class, and may be called a class instance or class object. Here, instantiation is then known as construction.

- Instantiate in Java means to call a constructor of a class which creates an instance or object, of the type of that class. Instantiation allocates the initial memory for the object and returns a reference.

- A reference is a value that refers to another value.
- A reference variable is declared to be of a specific type and that type can never be changed. The data within the object can be modified, but the reference variable can't be changed.

* Exception Handling:
  - When exception occurs, jvm creates the object for that particular exception and throw it
  - In a normal flow of program execution, if an abnormal situation occurs, then the jvm will terminate the program by throwing the exception class object.
  - It occurs at runtime.

For exception handling we can use

```
try {
    risky code ;
}
catch ( Exception class reference ) {
message related to exception ;
}
```

* In Java, we have two predefined class
  i) Class
  ii) Object : It is the superclass for all the classes.

* Errors:
  i) Syntax error :
     - If we miss any ( ) or { } in the code it will give syntax error.
  ii) Logic error or Semantic Error.
     - Error in the logic of the program.

* Exception handling means, to skip the instruction with the exception and execute the remaining part of the program correctly. Or, to stop the inappropriate termination of program.

Exception is classified into two types.

i) Un-checked Exceptions.
   - The exception where the machine is not aware of the exception.
   Eg: Runtime Exception.

ii) Checked Exceptions.
   - The exceptions which are known to the compiler.
   Eg: IO Exceptions.

Most used runtine exception methods.

```
getMessage();
toString();
printStackTrace();
```

- To handle the checked exceptions, we can use throws and try, catch.
- To handle the unchecked exceptions, we can use try and catch.

Types of try-catch.

i) Single try and catch.

```
try {
   riskycode;
}
catch (ExceptionClass ref){
     msg;
}.
```

ii) Single try and multiple catch. (most frequently used).

```
try{
    risky code;
}
Catch (Exceptionclass ref){
     msg
}
catch (Exceptionclass ref) {
     msg.
}.
```

iii) Nested Try Catch.
iv) Multiple Try and Single Catch.

- When you use try, you must use catch or finally or both.
- Catch for the subclass exceptions must come before the superclass exception.
  Eg: 'ArithmeticException' must be caught before or come before it superclass 'Exception' class.
- At a time only one Exception is occured and at a time only one catch block is executed.
- For each try block, there can be zero or more catch blocks, but only one finally block.

* Finally.

- It is a block which is used to deallocate the memory or close the Objects.

eg:

```
jdbc:
    Connection con;
    con.close();
```

- Finally is a block which gets executed irrespective of the exception.
- Before terminating, it will execute the statements inside the finally and then terminate the program.
- Must be written only with the try catch.
- It must be the last block of code in a class.

* Throw
  - throw is the keyword used to generate the user defined exception.
  - used with try catch or throws.
  - use throws to check the checked exceptions.

* Java Exception Propagation.
- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, if not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propogation.

Ex:

Main Method calls Method1.
Method1 calls Method2
Method2 calls Method3
Method3 calls Method4.

Memory Stack.

Method 4
Method 3
Method 2
Method 1
Main Method.

Here, if an exception occurs at method 4 and it is not handled there, it is then thrown to method 3. If not handled by method 3, it is again thrown to method 2. If exception is not handled by method 2 also, it is thrown to method1. And, it is not handled by method1, it is then thrown to main method for exception handling.

* Java throw keyword.

- The Java throw keyword is used to explicitly throw on exception. By explicitly means, you can throw an exception if you want to. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Syntax:

```
throw exception ;
```

Example:

```
throw new IOException ("Sorry device error");
```
-> parameterized constructor.

- We use throw keyword to generate user-defined exceptions. For example, when creating an application for voting, we can throw an exception if the age of voter is less than 18. Here, no exception will be thrown by the program itself as there is no exception in having age less than 18. So, in these kind of circumstances, we can use throw, to throw an user-defined exception.

- The throw keyword exception will terminate the program if the exception is met. You can overcome this by further using try-catch to run the program by handling the exception.
- Throw is used within the method.
- Throw is followed by an instance.
- You cannot throw multiple exceptions.

* Java throws keyword
  - If a method is capable of causing an exception that it does not handle, it must specify this behaviour so that callers of the method can guard themselves against that exception. You do this by including a throws clause in the method's declaration.
  - A throws clause list the type of exceptions that a method might throw. This is necessary for all exceptions, except those of type Error or RuntimeException or any of their subclasses. All other exceptions that a method can throw, must be declared in the throws clause. If they are not, a compile-time error will result.

  - The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
  - We use throws to check the checked exceptions
  - Primary advantage of throws keyword is it provides information to the caller of the method about the exception.

  General form of method declaration that includes a throw clause:

```
type method_name (parameter_list) throws exception_list
{
        // body of method
}
```

  - Throws is followed by class along with methods.
  - Throws is used with the method signature.
  - You can declare multiple exceptions.
    e.g. public void method() throws IOException, SQLException.

* With both throw & throws, it is alway a good practice to use try catch.

# * Wrapper Class

Is Java Pure OOPS?

=> Because of the primitive data types, it is not pure oops but is fully OOPS

=> Primitive data types are not part of object hierarchy, and they donot inherit Objects.

- For every data type, we have a predefined class.
  Eg: int -> Integer , Character → char
  float -> Float , Boolean -> boolean

- For every data type, the predefined class is the Wrapper Class that consists of many methods.
  Most important : parseInt() , parseFloat(), etc.

Note: Non-Primitive data types are considered as referencial data.

- Primitive type can't pass a reference to a method. To overcome this, Java provides Wrapper Class, which are classes that encapsulate primitive type within an object. These wrapper classes offers a wide array of methods that allow you to fully integrate the primitive types into Java's object Hierarchy.
- All Wrapper Class in Java are Final.

## * Autoboxing and Autounboxing.

- Autoboxing
  - It is the process by which a primitive type is automatically encapsulated or boxed into its equivalent type wrapper whenever an object of that type is needed.
  - There is no need to explicitly construct an object. Auto-boxing is the process by which the value of boxed object is automatically extracted (unboxed) from a type wrapper when its value is needed.
  - With autoboxing, it is not necessary to manually construct an object in order to wrap a primitive type. You need only assign that value to a type-wrapper reference. Java automatically constructs the object for you.

Example: Integer iob = 100; // autoboxing an int primitive type.
Double dob = 15.75; // autoboxing an double primitive type.

- The Java Compiler applies autoboxing when a primitive value is:
  i) Passed as a parameter to a method that expects an object of corresponding wrapper class.
  ii) Assigned to a variable of the corresponding wrapper class.

- Unboxing.
  - Auto-Unboxing takes place whenever an object must be converted into primitive type.
  - Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example, conversion of Integer to int.
  - The Java Compiler applies unboxing when an object of a wrapper class is:
    i) Passed as a parameter to a method that expects a value of corresponding primitive ty
    ii) Assigned to a variable of the corresponding primitive type.

Example:

```
Integer i = new Integer(10);
int j = i;   // unboxing the object.
```

* **Generics** apply

- With generics, you can ~~define~~ an algorithm to a wide variety of data types by defining a algorithm once that is independent of any specific type of data.
- The term generics means parameterized types. Parameterized types are important because they enable you to create classes, interfaces, and methods in which the type of data upon which they operate is specified as parameter. Using generics, it is possible to create a single class, that automatically works with different types of data. A class, interface or method that operates on parameterized type is called generic, as in generic class or generic method.
- With generics, all casts are automatic and implicit.
- Generics can have more than one parameter.

Syntax: ↓

```
Class ClassName <DataTypeName> {

}
```

Example:

```
class Test <T> {   // T is the DataType Here. You can use any other
                   variables name but remember in this case it is
                   the datatype. More than saying datatype,
                   it is actually a placeholder for the datatype
                   that is unknown or can be any.
}
```

- Generics work only with reference types. i.e. when declaring an instance of a generic type, the type argument passed to the type parameter must be a reference type. You cannot use a primitive type, such as int or char.

Example: `Test<int> intOb = new Test<int>(53);` // Error, can't use primitive type int.

- A key point to understand about generics type is that a reference of one specific version of a generic type is not type compatible with another version of the same generic type. Suppose iob is int reference & sob is ~~char~~ String reference.

Example: `iob = sob;` // Error or Wrong!