Java Full stack with AI

Fayaz - Trainer

→ What is programming?

Def:- Programming is writing instructions for a computer to perform specific task.

→ These programs are written in diff languages that computers can understand.

→ The purpose of writing these codes is solve problems, create systems & applications.

Types of Programming Lang:-

⇒ High-level languages:

- closer to human lang. (eg: Java, Python)
- Easy to read & write.

⇒ Low-level languages:

- closer to machine lang. (eg:- Assembly, machine code)
- Harder to understand.

→ High-level code needs to be translated into low-level for execution.

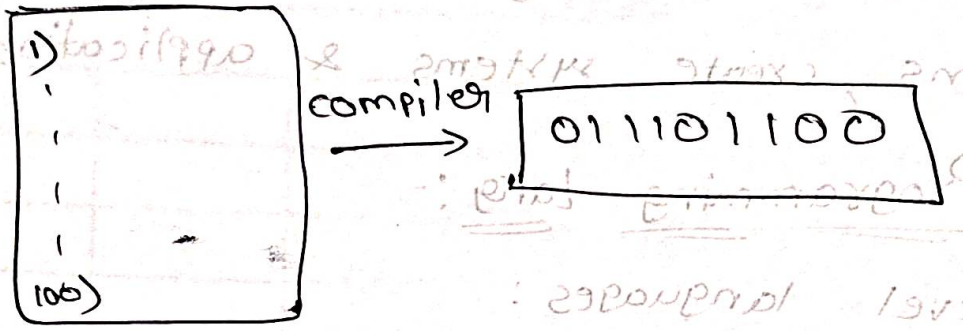
→ What is Binary code?

The language of computers. Everything will be in 0 & 1.

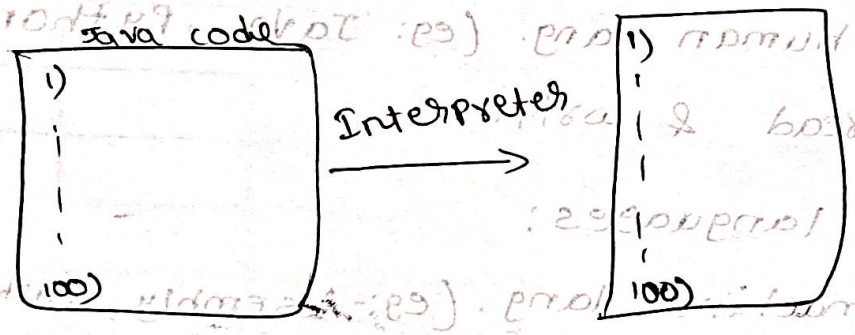
⇒ Translating code :-

compiler :- Translates the entire program into machine code before execution.

Interpreter :- Translates code line-by-line during execution.



eg:- C, C++, golang.



eg:- Python, Java script, Ruby.

- ✱ compiler is faster than Interpreter.
- ✱ Java is a Hybrid language. Java is a combination of both compiler & Interpreted language.
- ✱ so, it's a fast language.
- ✱ Java first compiles to byte code then interpreted by JVM (Java virtual Machine).

→ compiler generates .exe files.

☆ History Of Java :-

- 1991 - Project started :- Java was initiated by Sun Microsystems as Green Project and initial name was "OAK". by Games gosling.
- 1995 - Java Released : Renamed it to Java and Java 1.0 launced with write-once-run-anywhere Philosophy.
- 2010 - oracle Acquisition : oracle acquired sun microsystems and took over Java.
- 2014 - Java 8 Released with some major change
- 2025 - Java 24 Released on march 18 2025

☆ Features of Java :-

⇒ Platform Independent :- WORA (write once run anywhe

• Java → Byte code $\xrightarrow{JVM}$ Binary code.

• class file

↳ This file makes Platform independen

→ Any OS (operating system) can understand byte cod

→ Java bytecode can run on any device with Jvm.

⇒ object-oriented :- supports oops concepts like inheritance, encapsulation and polymorphism.

⇒ strictly typed language :- In java we cannot name same variable for diff. data types.

⇒ Automatic Garbage collection :- It will collect all the unused variables Automatically. To free memory.

⇒ Robust :- Java can handle errors gracefully and keeps running without crashing, bcz of exception Handling.

⇒ Secure :- Java is a very much secured lang than any other languages.

Java does not have pointers (gives direct memory address). So, it does not have direct access to memory address.

⇒ Simple, Portable, Architecture Neutral, Interpreted, Dynamic, High performance, Multi threaded, Distributed, are other features of Java.

⇒ Backward compatibility :- The code supports in any version. Easily I can upgrade to new version.

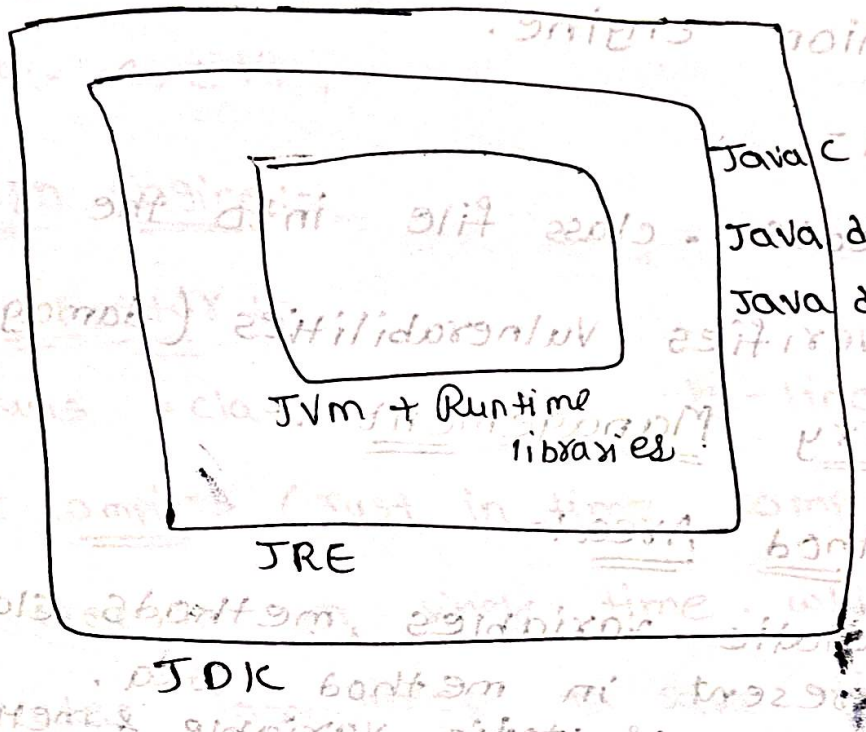
JDK, JRE, JVM

⇒ JDK - Java Development Kit.

⇒ JRE - Java Runtime Environment.

⇒ JVM - Java Virtual Machine.

⇒ JDK :-



→ JDK is a combination of JRE + Development Tools (Java compiler, Java debugger)

→ JRE is a combination of JVM + Run time Libraries (Java.util, Java.io) .class files.

☆ JVM :-

→ It converts Byte code (.class file) to Binary code (low level lang).

• Java → compile → • class → JVM → Binary code (Run)
(HLL) (Byte code) (Interpreter)

→ Interpretor works on .class file in Java.

→ JVM is Responsible for Platform Independent

→ JVM has 3 parts.

(i) CLSS (Class Loader sub system).

(ii) Memory Management.

(iii) Execution engine.

⇒ CLSS :-

- It loads .class file into the JVM.
- It verifies vulnerabilities (damage/virus)

⇒ Memory Management :-

→ Method Area :-

Static variables, methods, class info present in method area.

→ memory of static variable & methods will be in method Area.

→ Heap Area :-

Instance variables, objects are present in Heap Area. (Non-static variables)

→ Instance variables are also called global variables.

→ Stack :-

- Local variables are stored in stack. (Variables present in method)
- Threads are also created in stack.

→ Program counters :-

- It stores the current & next instructions for execution.

→ Native Method Area :-

- we can use other languages codes, (e.g:- C, C++,

⟹ Execution Engine :-

→ Interpretor :-

Runs .class file line-by-line.

→ JIT compiler (Just in time compiler).

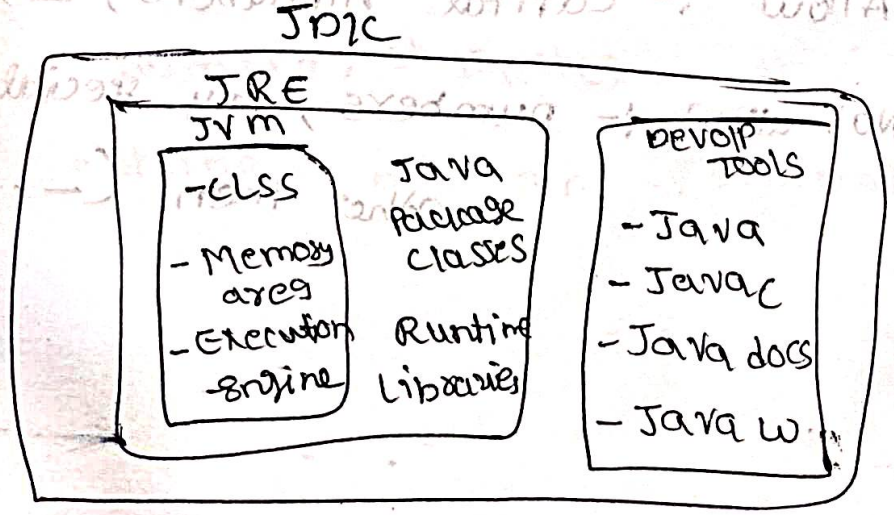
It executes in single time. which interpret executes 1000 times to save time.

- It compiles when there is repeated executions.

→ Garbage collector :-

- stores all the unused memory.
- Trashing all the unused memory.

∴ Architecture of JDK, JRE & JVM.



☆ Naming conventions :-

⇒ Class Names.

Pascal case → start with capital letter. Every word. (eg:- Chennai Super Kings).

Alphabets, numbers, special characters. (_, $)

eg:- Chennai 1 ✓ It allows only two

1Chennai X ; S.characters.

eg:- _chennai ✓

$ Chennai ✓

Starting Allow :- Capital Alphabets, _, $.

starting Not allow :- numbers, any special charcters other than (_, $)

✯ Variables :-

camelCase :- First letter should be Small of first word and second and soon words first letter should be capital.

eg:- chennaiSuperKings

Allowed & Not allowed are same as class names.

✯ Method Names :-

→ same as variables.

→ camelCase.

✯ Packages :-

→ Lower case

✯ Project Names :-

→ Pascal case.

✯ Data Types :-

→ There are two types (i) primitive datatypes.

(ii) Non-Primitive "

→ we should tell java what type of variable we are creating.

- number, charaters
- string, boolean, decimal.

⇒ Primitive Data types :-

1 Gb = 1024 mB
1 mB = 1024 kB
1 kB = 1024 Bytes
1 Byte = 8 bits.

There are 8 types.

int, byte, short, long

Default values

0 0 0 0

Byte → 0-1 byte = 8 → Allows $(-2^{n7}$ to $2^{n7-1})$ -128 to 127

short → 0-2 bytes = 16 → $(-2^{15}$ to $2^{15}-1)$

int → 0-4 bytes = 32 → $(-2^{31}$ to $2^{31}-1)$

long → 0-8 bytes = 64 → $(-2^{63}$ to $2^{63}-1)$ of memory.

0.0 Float - 0.0 4 bytes. (precision → upto 7 after point).

0.0 double - 8 bytes (→ upto 16 after point).

Empty space char - characters. eg:- ('A', 'B') ('@') ('4').
- 2 bytes

boolean - true/false. - 1 bit.

⇒ Non-primitive Data types :-

⇒ user defined data types / classes.

eg:- String, Array.

→ String accepts group of characters.

eg:- String company = "FLM".

☆ Operators :-

→ Arthimetic operators.
→ Relational operators.
→ Assignment operators.
→ unary operators.
→ Logical operators.

⇒ Arthimetic operators :-

+ - sum ∵ If we add two strings it is called concatination.
- - minus
* - into (multiply).
÷ - divison
% - modulus.

⇒ Assignment operators.

= - Equals to.
+= eg:- a = 2 ; a += 2 ⇒ a = 4
-= eg:- a = 2 ; a -= 1 ⇒ a = 1
*= eg:- a = 2 ; a *= 2 ⇒ a = 4
/= eg:- a = 2 ; a /= 2 ⇒ a = 1
%= eg:- a = 2 ; a %= 2 ⇒ a = 1

⇒ Relational operators :-

== → Equals eg:- a = 10, b = 10.
a == b → true.
> → greater than. eg:- a = 11, b = 10.
a > b → true
a = 9, b = 10
a > b → false.

eg:- a = 9, b = 10

< → less than

a < b → true

a = 11, b = 10

>= → greater than equal to

a < b → false.

eg:- a = 10, b = 10

a >= b → True

<= → less than equal to eg:- a = 10, b = 10

a <= b → True.

!= → Not equal to.

a = 10, b = 11

a != b → true.

a = 10, b = 10

a != b → False.

⇒ unary operators :-

These are performed on a single operand.

a + b operators.

operands

- +a (5, +5)
- -a (5, -5) → This converts into opp sign
- ! (opposite) → True - false

false - True.

eg:- boolean a = true;

!a → false.

- Increment operator :- (++) → They will increase the value by 1.
  - → post - a++
  - → pre - ++a

∴ a = 5;
++a = 6;

before crossing the operation :

eg:- int a = 20;
syso(a); → 20
syso(++a); → 21
syso(a); → 21

∴ a = 5;
a++ = 6;

After crossing the operation.

eg:- int a = 20;
syso(a); → 20
syso(a++); → 20
syso(a); → 21

- Decrement operator :- (--) → They will decrease the value by 1
  - ⇒ post → a--
  - ⇒ pre → --a.

∴ a = 5;
--a = 4;

eg:- int a = 20;
syso(a); → 20
syso(--a); → 19

∴ a = 5;
a-- = 4;

int a = 20;
syso(a); → 20
syso(a--); → 20
syso(a); → 19.

⇒ Logical operators :-

→ && (And)

→ || (OR)

* AND (&&) :- works on booleans only.

true && true = true.

f && f = false

t && f = false

f && t = false

* OR (||) :- works on boolean only.

t || t = true

f || f = false.

t || f = true

f || t = true.

* ! (NOt) :- works on boolean

!true → false

!false → true

⇒ Ternary operators :-

? , :

eg:- a = 10;

b = 20;

c = (a>b)? true : false

"

false

o/p :- false.

condition ? statement1 : statement2

If true statement1 prints

If false statement2 prints

☆ Dynamic input :- To give input by user.

Syntax

```
Scanner a = new Scanner(System.in);
```

Conditional statements :- (Yes/no statements)

→ if else.

Syntax

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

Diff blw print & println

∴ In print o/p will be printed in same line.

∴ In println o/p will be printed one after the other.

→ if

→ if - else.

→ if - elseif - else

⇒ if - elseif

} else if can be so many.

⇒ Nested If :- If we write another if & else if conditions in it, it is called Nested if.

eg:-

```
If ( ) {
    If ( ) {
    }
    else if ( ) {
    }
    else {}
}
```

⇒ Switch Statements :-

Syntax :- Switch (expression) {

case exp1 :

statement ;

break ;

case 2 :

statement ;

break ;

case 3 ;

statement ;

break ;

default :

statement ;

✡ Arguements :-

→ Command Line Arguments :-

We can pass few arguements while running the code. [String[] args] in main method.

To run this right click, goto run configurations in run as and give arguements.

★ If we want to print single character from user input.

```
syso ("Enter a char");
Scanner sc = new Scanner(System.in);
char input = sc.next().charAt(0);

syso (input);
```

★ LOOPS :-

→ For LOOP
→ While LOOP
→ DO while LOOP
→ For each LOOP.

Why LOOPS?

→ FOR LOOP :-

When you know how many times you have to do a task. ie, Range.
→ It is also called entry control LOOP.

→ For loop works with 3 things.

1 → Intialisation (variable)
→ int a = 0;

2 → condition → a<5;

3 → update the value.

syntax:- for (1 ; 2 ; 3) {
code
}

eg:- for (int a=1; a<=10; a++) {
code
}

→ While LOOP :-

When you don't know how many times you have to do task.

syntax:- while (condition) {

}.

→ For while Loop we have to initialise outside.

→ we have to update inside LOOP.

eg:- int i=1;

```
while (i<=3){
    i++;
}
```

→ It is called entry control loop.

⇒ Do while Loop :-

⇒ Do while executes atleast once irrespective of condition.

syntax:-

```
int a=1; → initialise
do {
    // logic.
    a++; → update
} while (condition);
```

a<=5

→ It will check the logic once that is in do and checks for while condition.

If it is true it will again goes to do or else it will exit.

→ It is called exit control Loop.

☆ Nested for loop :-

-> one for loop inside another for loop.

syntax:-

```
for (int a=1; a<=5; a++){
    for (int b=1; b<=5 ; b++){
    }
}
```

☆ Patterns :-

i, value is for rows. -> 1st for LOOP

j value is for column. -> 2nd for LOOP

☆ Break & continue statements :-

=> Break :- exit from all cases.

In loops if you use break, it will exit from the LOOP.

eg:- for (int i=1; i<=5; i++) {

```
    if (i==4) {
        break;
    }
    print(i);
}
```

o/p
i → 1 - print
i → 2 - print
i → 3 - print
i → 4 - break (exit) For loop.

⇒ continue :- It skips the current itteration.

eg:- for (int i=1; i<=100; i++) {

```
    if (i==4) {
        continue;
    }
    print(i);
}
```

o/p
i → 1 - print
i → 2 - print
i → 3 - print
i → 4 - Do not print.
i → 5 - print.

Prime Number :-

When a num is divisible by 1 & itself.

Methods :-

It is used to execute a block of code

That block of code is called Method.

⇒ 3 points for every method. → Return type.

syntax :- Void name() {

}

Return type :-

Void - there is nothing to return.

8 primitive Data types (eg) NON - primitive data types.

→ we have to return something.

⇒ Return type methodName() → Method declaration.

⇒ Method calling :- For example method name is "sum()";.

syntax :- sum();.

eg :-

```
int sum(){

}
sum();
int value = sum();
```

```
String name(){

}
String name =
name();
```

→ Variables outside static method are called Non-static (or) Instance variables.

→ The variables inside static method (or) any other method are called Local variables.

→ To call Instance type object is required.

→ To call local/static return data types no object is required.

eg:- Test t = new Test();
↳ class name.
↳ object

☆ Arguements / Parameters :-

eg:- void sum (int a, int b);
↳ Parameters / arguements.

☆ Type casting :- converting one data type variable into another data type.

byte → short. } examples
short → long. }

These are two types:-

⇒ Implicit Type casting :-

This is automatically done by Java.

eg:- byte b = 10;
short s = b;
int i = b;
int j = s;
long L = j;

⇒ Explicit Type casting :-

This we have to do manually.

eg:- long L = 10000001L; } ✗ This will not work.
int i = L;

short s = 128; → Greater than byte size
byte b = (byte) s;
sysout(b) = -128

Data loss → circle → -128 to 127.

-128, -127, -126 ... 126, 127, 128
last.

So, the value changes it is called data loss.

☆ String :-

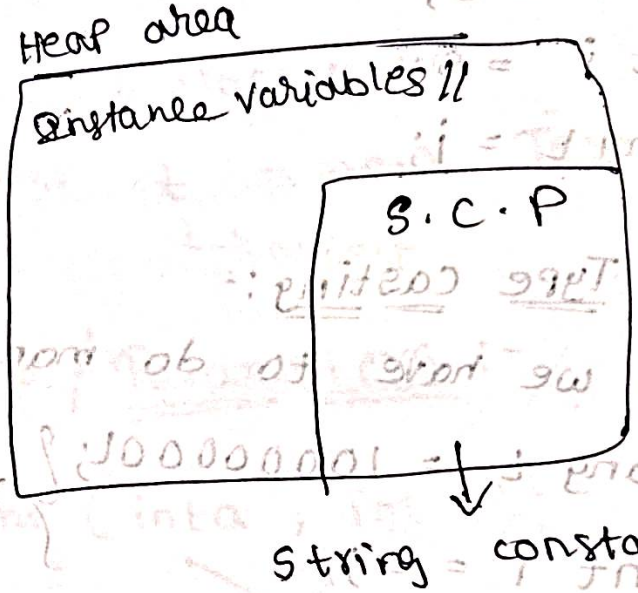
→ Stores group of characters.

→ In strings we can create new object using literals & objects.

eg:- String s = "New string"; // Literal

String s = new String("New string");

// objects.



→ In java all strings are immutable.

→ Immutable → which doesn't change.

eg:- String s = "FLM";

s = "FLM EDUtech"; X

→ All strings created using literals are stored in string constant pool.

→ If we update the value of string it will update but in background it will create new string but will not update.

eg:- String s = "FLM";

s = "FLM FLM";

print (s);

O/P "FLM FLM"

Background

case-1) s → FLM
after updating.
s → FLM FLM

Reference will be changed but there is no updation.

case-2) s1 - FLM
s2 - FLM
s3 - FLM

FLM → s1, s2, s3

String storage (= FLM) = 6 bytes + overhead ( )
↓
2 bytes

→ The above all is for strings created using literals.

→ strings which are created using objects ~~it with exe~~ are stored in Heap area.

eg:- String s1 = new FLM;
String s2 = new FLM;

It stores in two diff objects in Background

eg:- String s1 = "FLM";
String s2 = "FLM";
Print (s1 == s2);

o/p :- true.

Because above strings are created using literals. They store in S.C.P. In this for same string it will have same address. i.e, s1 & s2 refers to same FLM.

→ In java for strings to compare it compares address.

eg:- string s1 = new string ("FLM");
String s2 = new string ("FLM");
Print (s1 == s2);

o/p :- false.

opp to literals.

Print (s1.equals(s2));

→ compares content.

✯ Methods in Strings :-

→ equals → checks the content (compares).

→ Length → To find length of string.

→ charAt → To give the character of string using index.

→ isEmpty → checks the content is there are not.

→ isBlank → checks the string is Blank (or) not.

→ equalsIgnoreCase → It will compare two strings without considering capital (or) small letters.

→ contains → It will check ~~two~~ if one string has another string.

eg:- s1 = "flm"
s2 = "flm edutech".
print (s2.contains(s1));
o/p:- true.

→ startsWith → It checks two strings weather it starts with the one string or not.

eg:- s1 = "flm";
s2 = "flm edutech";
print (s2.startsWith(s1));
o/p:- true

→ endsWith

→ indexOf → Gives the index of the char in string.

→ lastIndexOf → Gives the last index of char in a string if we have 2 times (or) more times char.

→ substring → It gives half string.

eg:- S1 = "programming".
0 1 2 3

Print(S1.substring(1,4)); Print(S1.substring(0,3));

O/P:- (809) O/P:- P80.

In ending it takes one number extra.

→ It also takes only one index. i.e, starting index and print the remaining string.

e.g:- S1 = "programming";
0 1 2 3

print(S1.substring(3));

O/P:- gramming.

☆ Arrays :- It stores multiple values of same datatypes

→ It has a fixed size.

syntax:-

```
int[] arr = new int[5];   → size.
int arr[] = new int[5];   → possible.
```

[data Type[] name = new array]

arr[5]

1	10	5	3	2
Index = 0	1	2	3	4

To keep numbers arr[0] = 1

arr[1] = 10

!

so on.

// 2nd way. in creating array.

```
int[] arr = {1, 10, 5};
```

size = 3

→ Arrays default value based on data type.

☆ continious Memory location :-

→ In Java it allocates continous memory location.

eg:- arr[] = {1, 2, 3}

For index 0 = 100 - 104 bytes (int)

1 = 104 - 108

2 = 108 - 112.

To find memory for int.

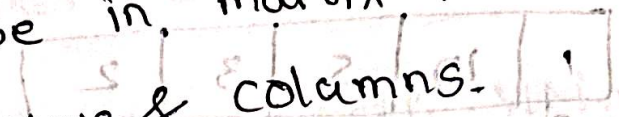
Base + Type size X index.

→ uses this formula.

→ Arrays are very much faster in Java because it has contagious memory locations, so, it can access directly the value using above formula.

✱ 2d Arrays:-

Which will be in matrix format, which works in rows & columns.



syntax:- ~~arr[][]~~

```
int[][] arr = new int[2][2]
```

✱ Jagged Array:-

which have diff rows & coloums.

→ Arrays faster in read operations.
→ Arrays slower in write operations.

✱ Arrays.toString(arr);

prints array values. normally we use for loop to print array.

☆ OOPS (Object-oriented programming)

→ Java is not 100% OOPS language because it has primitive data types. bcz these are not ~~are~~ ~~it~~ classes.

→ To make it 100% OOPS they introduced wrapper classes which contains int class etc

→ In Java everything is objects & classes.

→ class is blueprint in Java.

→ class is combination of 3 things.

1) Variables / fields / properties.
2) Methods
3) constructor.

→ class can have infinite objects.

⇒ constructor:-

→ It is used to create objects.

→ To assign the values to instance variables.

→ constructor will not have return type.

eg:-

```
Public class Bike{

    Bike(){ // Non parameterized constructor
    }

}
```

→ constructor should always have same name as the class name.

→ JVM provides default constructor if we not write.

```
Bike (String colour){
}
```
→ parameterized constructor.

→ one class can have multiple constructors with a condition that we can't write same constructor twice.

→ parameters should not be same.

→ when we write parameterized constructor the JVM will not give default constructor.

→ Then we have to write default constructor which is non parameterized constructor.

Q.

☆ Types of Variables :-

→ There are 3-types of variables.

(i) Instance → which you declare outside all methods and inside a class.

eg:-
int name; → also called as global variable.

→ Stored in Heap area.

→ You need an object to access.

→ Jvm gives default value.

(ii) Static → which you declare out side all methods and inside a class.

eg:-
static int name;

→ Stored in method area.

→ Loaded when a class is loaded.

→ Does not require a object.

(iii) Local → which you declare inside any method.

eg:- void Book()
{
int a = 20;
}

→ Stored in stack area.

→ Does not require object.

→ Jvm will not give default value

★ OOPS principles =4

(i) Inheritance.

(ii) Polymorphism.

(iii) Encapsulation.

(iv) Abstraction.

→ Any programming lang with OOPS works on these 4 principles.

★ Inheritance :-

→ Parent class gives its features to child class.

→ Parent class is also called super / Base class

→ child class is also called sub / derived class.

→ To inherit Parent class to child class there is a keyword "extends".

→ Inheritance is also called as IS-A relationship.

⇒ Single Inheritance :-

→ It has single parent & single child

⇒ Multi - level Inheritance :-

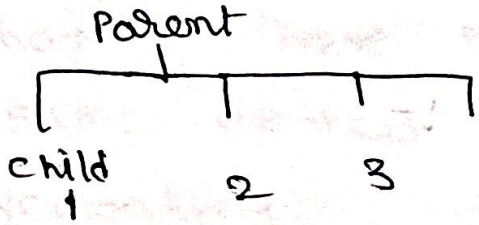
→ It has multi parents & multi child.

eg:- Generations in real life.

⇒ Hierarchial Inheritance :-

→ It has one parent and multiple childs.

eg:-

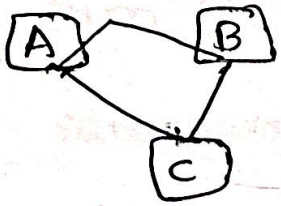


→ In Hierarch Inheritance its child's have no common features.

⇒ Multiple Inheritance :-

→ It takes features of two parent class.

eg:-



→ But it ~~suppo~~ doesn't support in Java.

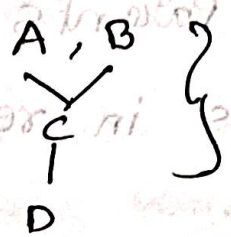
→ It has Ambiguity issue that means A & B have same features so, c class will confuse to take features. This confusion is called Ambiguity.

→ It is also called diamond problem. because it has diamond shape passing.

⇒ Hybrid Inheritance :-

→ It is also not supported in Java.

→ It is a combination of multiple inheritance and single heritance / multi-level inheritance

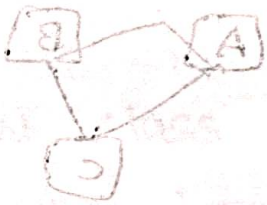


→ Hybrid Inheritance.

☆ this vs super :-

⇒ this :-

- It is a keyword in Java.
- It refers current instance (or) same class Instance (object).



⇒ super :-

- It is a keyword in Java.
- It refers to parent class instance (object).

→ this (or) super cannot access in static method bcz it cannot create memory until the class is called.

✯ Polymorphism :- → Many forms.

Poly - Many.

Morphism - greek - Forms.

→ There are two types :-

(i) compile time Polymorphism / static / Early Binding

(ii) Run time Polymorphism / dynamic / Late Binding.

→ Method overriding.

→ Method overloading.

✯ Method overLoading :-

→ one method can have multiple forms.

→ using same method again & again is called overloading.

→ we can use same method names / multiple time with different parameters.

Conditions :-

→ change the count of Parameters.

Eg:- void sum (int a, int b)

void sum (int a, int b, int c).

→ we can use same method with same no. of parameters by changing data type.

Eg:- void sum (String a, int b).

~~void sum (int b, String a).~~

→ By changing order of parameters we can use same method name.

Eg:- void sum (int b, String a).

→ The Behaviour of this method is known at compile time. so, it is called compile-time polymorphism.

→ Before executing only JVM knows that which method should be executed. so, it is called Early binding.

→ we can change return type also.

→ we can overload final methods.

☆ Method overriding:-

→ This comes into picture only when there is Inheritance.

→ This works with Parent and child relationship.

→ You can overide a Parent class method in child class by giving your own implementation.

conditions:-

→ Always method name should be same.

→ You cannot change the return type.

→ You cannot change Parameters.

→ you can change only implementation inside method.

→ @override annotation is not mandatory.

→ we cannot override static methods in Java.

→ we can write static methods but when we tag @override annotation it shows error and this is called method hiding.

→ This because overriding works at runtime while for static methods memory is assigned while ~~creating~~ Loading the class. so, we cannot override static methods.

→ we cannot override final methods also.

★ Order of acces modifiers :-

Public → Protected → default → Private.

→ While overriding we cannot assign weaker access. that means for public methods we cannot give weaker access modifiers.

→ For private methods we cannot give all other access modifiers. because private is only for that class.

→ For default methods we can give protected and public.

→ For protected we can give public.

→ This order of access modifiers is another rule in overriding.

✱ Final Keyword :-

→ It is a keyword which makes data constant. we cannot update.

→ we cannot declare final variables without initializing.

→ Final Keyword can be used for methods also but we cannot override that method but we can overload that method.

→ we can use final keyword in class. also but for that classes we cannot perform Inheritance.

☆ Access Modifiers :-

→ These decide the access of your variables, methods ---.

☆ public static void main (String[] args) {

- public → access modifier.
- static
- void → return type
- main → method name.
- (String[] args) → String array to store multiple elements
- { → it

→ There are 4 access modifiers.

(i) Public
(ii) Private
(iii) Protected.
(iv) default.

⇒ Public :- (key word).

→ we can use public at variables, methods, class, constructor.

→ when we use public keyword that will be accesible across your project.

⇒ private :- (key word).

→ we can use private at variables, methods & constructors.

→ when we use private we will be able to access in same class.

→ when we make constructor as private we cannot create objects in other classes.

⇒ Protected :- (key word)

→ we can use protected at variables, methods & constructors.

→ when we use protected we will be able to access with same Package.

→ we can access Protected when the class is child class.

⇒ default :-

→ we call default when we do not mention any access modifiers. for var, methods, class & constructors;

→ we can access them in same Package only.

✯ Encapsulation :-

→ In real time we have to follow encapsulation in every class.

→ we will make variables private and in main class to get data we use setters and getters.

→ To hide data we use Encapsulation.

✯ Object class :-

→ For every class there is a Parent class given by JVM that is called Object class.

⇒ 8 imp methods :-

- → equals()
- → hashcode()
- → tostring()
- → clone()
- → finalise()
- → wait()
- → notify()
- → notifyALL().

⇒ equals()

→ compare 2 objects.

obj1, obj2.

obj1.equals(obj2);

obj1.equals(obj2)

→ false.

If we write @override method it gives true then it compares content.

⇒ hashcode() :-

unique value → not memory address.

→ It is a unique value used to search objects (or) anything fastly

→ ~~It~~ A.

☆ Hash code & Equals contract :-

→ When two objects are equal they should have equal hashcode.

→ when two hashcodes are equal they need not be equal objects.

☆ 2 equal objects should always have a equal hashcode but 2 objects having equal hashcode need not to be same.

⇒ tostring() :-

→ It comes from object class().

→ If we print any object JVM calls tostring automatically.

→ If we override it gives custom implementation.

⇒ clone() → explained after Abstraction.

⇒ finalise() → This method is deprecated after Java 9.

⇒ wait(), notify(), notifyAll() → explained in threads

☆ Abstraction :-

→ Hiding the actual implementation.

→ Abstract method are written only body not the implementation.

→ To write a abstract method abstract keyword is mandatory.

Eg:- Public abstract void sythi();

→ This abstract method has only body there is no implementation. This implementation is given by child class.

→ Abstract methods can exist only in an abstract class.

→ Abstract classes are classes which have abstract methods and normal methods (concrete methods).

→ Abstract method should have abstract class but for abstract class there should not be abstract method.

→ If we inherit abstract class we should implement abstract method compulsory.

→ If we have 3 abstract method in abstract class we have to implement all the 3 abstract methods in child class.

→ Abstract methods cannot be static.

→ Abstract methods cannot be final.

→ we cannot create object for abstract Parent class.

★ Interfaces :-

→ It is also same as normal classes.

→ To write Interface classes Interface keyword is required.

→ Interfaces support 100% abstraction.

Eg:- Public Interface Vehicle {

}

→ In interface all methods are by default abstract methods.

→ In interfaces abstract keyword is not mandatory.

→ In interfaces all variables are public, static, final by default.

→ we cannot create constructors for interface.

→ we cannot create objects for interface Parent class.

⇒ class → class inheritance → extends

⇒ Interface → class → implements.

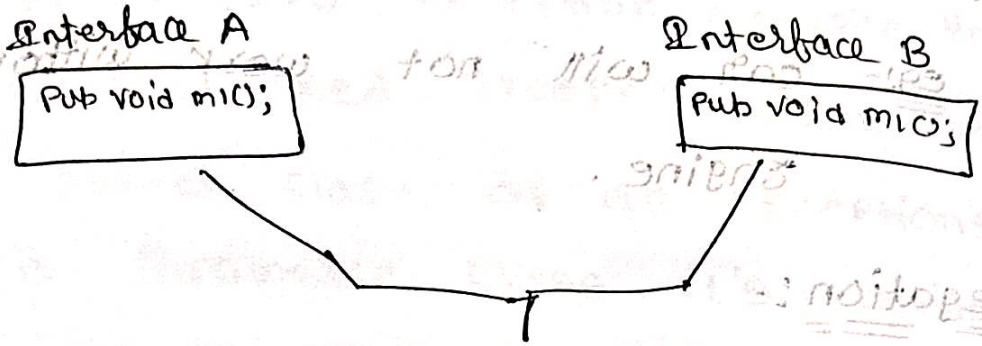
⇒ Interface → Interface → extends.

These are the keywords to we.

→ Method overloading is possible in Interface.

→ Method overriding is not possible.

→ Multiple Inheritance is possible in Interface because the implementation of method is given in child class.



☆ Has-A Relationship :-

→ It has two types.

→ Aggregation

→ composition.

→ one class having a field which is an object of another class.

Eg:-

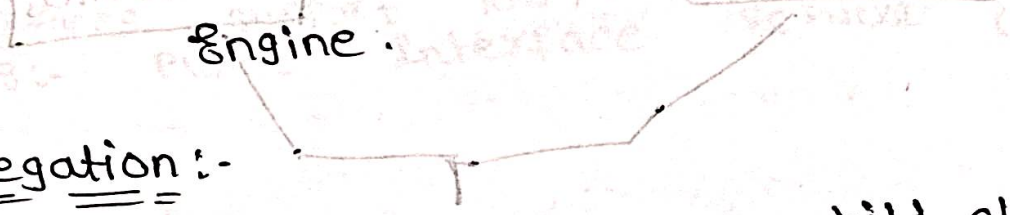
```
class Car {
    Engine engine;
}
Class Engine {
    String model;
    String capacity;
}
```

⇒ Composition :-

When a parent class and child class are very tightly coupled then it is called composition.

→ parent doesn't make sense without child class.

eg:- car, will not work without engine.



⇒ Aggregation :-

When a parent class and child class are very loosely coupled then it is called Aggregation.

→ parent make sense without child class.

Eg:- cake will work without music player.

☆ Wrapper class :-

→ To make Java 100% oops they created wrapper classes.

→ byte - Byte
→ short - Short
→ int - Integer
→ long - Long
→ char - character
→ float - Float
→ double - Double
→ boolean - Boolean.

☆ Auto-Boxing - converting int to Integer.

☆ Auto - unboxing - converting Integer to int.

↳ This is for all data types.

☆ Exception Handling :-

⇒ Exception :-

→ This causes a abnormal termination of your program.

→ There are two types of exceptions.

(i) checked Exception → compile time Exception

(ii) unchecked Exception → Run time Exception.

→ The Parent class of all Exceptions in Java is Throwable class. it's child class is Exception class. For unchecked Exception Exception class has another child that is Run-time Exception class.

→ But for checked exceptions the parent class is Exception class.

⇒ Eg:- File not found → checked.

~~SQL Exception~~

Arithmetic Exception → unchecked.

⇒ Handling :-

→ try :-

→ write all your error prone code inside your try block.

→ when an exception is encountered in try block it immediately goes to catch block.

⇒ Catch :-

→ used to handle the exception.

→ For one try block we can write many catch blocks.

→ In catch we can write again try catch blocks.

⇒ finally :-

→ execute always. irrespective of exception.

→ It is used for clean up purpose/ resource closing.

Eg:- sc.close();

→ To close scanner resource.

→ To stop executing finally block there are two ways. (i) forcefully shutdown JVM by using system.exit(0); but in real time we should not do this.

→ (ii) By getting out of memory error.

⇒ throws :-

→ this is used at method declaration.

→ we cannot handle exception using throws

→ we can only delegate/ cascade the exception to calling method.

⇒ custom Exception:-

→ create a class.
→ Extend with runtime Exception.
→ create a parameterized constructor
→ call parent class using super() keyword.

⇒ throw:-

→ It will not handle your exception.
→ Just used to throw custom exception
→ but still exception should be handled by a catch.

☆ File Handling:-

→ File Handling in Java refers to reading from and writing to files stored on the system. It allows java programs to store data permanently, retrive it, and manipulate files and directories.
→ why file Handling?
→ To store data permanently.
→ To read configuration files.
→ To create, read, write, update (or) delete files (or) directories.

☆ Threads :-

→ Thread is very light weight process in Java.

⇒ Multithreading :-

☆ How to create a thread?

→ there are 2 types.

(i) extends Thread

(ii) implements Runnable.

(i) extends thread (same as custom exception)

run - override.
method

(ii) implements runnable.

run - override.
method

→ we can save time using multithreading.

→ The most preferred type to create thread is implements runnable bcz runnable is interface and multiple Inheritance is supported in interface only & Thread is class so, it is not preferred.

→ The parent class of Thread is Runnable.

→ ~~We have to call~~ We can register a thread only by calling start method.

→ you cannot register a thread by calling run method.

→ Threads are stored in stack bcz they are temporary.

→ In multithreading we run threads parallely and output is not predicted bcz it depends on memory scheduler.

★ Join() :-

→ It stops the execution of main thread until the thread execution is completed.

★ Race condition :-

→ When multiple threads across a resource at a same time some collisions occurs and few iterations are skipped.

★ Synchronization :-

→ It acquires a lock.

→ For example, In banking system for debit process we use synchronization bcz two transactions for same debit card should not be possible.

→ once first transaction happend it locks that using synchronization.

★ Dead lock :-

→ t1 thread is waiting for t2.

→ t2 thread is waiting for t1.

```
t2.t1 = t1;
t1.t2 = t2;
```

★ Thread Lifecycle :-

→ There are 5 stages in thread lifecycle.

⇒ **New** :-

→ when we create new thread.

```
T1 t1 = new T1();
```

⇒ **Runnable** :-

```
t1.start();
```

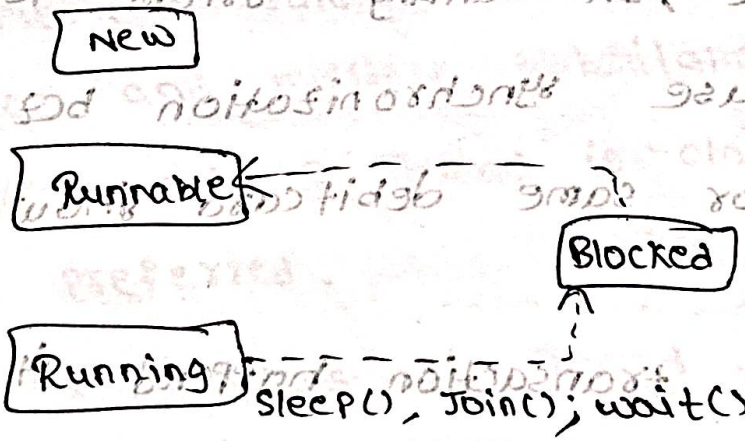
→ Thread is ready to run and the process of scheduling memory is called Runnable.

⇒ **Running** :-

→ Execution is started.

i.e, Run method is called.

⇒ **Blocked / waiting** :-



⇒ **Dead** :-

wait() :-

→ this is called only inside synchronized block.

→ notif();

If wait is called, notify should be called and it is also in synchronized block.

→ It is called when single thread is called.

→ notifyAll();-

eg:- t5 ! lock - wait

t4 → t5 lock - wait

t3 → t5 lock - wait

t5 → notifyAll();

→ It is used when we have multiple threads

☆ Thread Priority :-

→ We can give only 1-10 numbers.

→ It will set the priority of thread which should execute first and next by giving numbers.

☆ Diff b/w final, finally, finalise;-

⇒ final - Keyword

used at

variable - constant

method - cant override

class - cant inherit.

⇒ finally - to close resources.

→ this executes always.

⇒ finalize - This is invoked by Garbage collector.

→ It is deprecated from Java 9.

✡ String Builder, StringBuffer :-

→ It is stored in heap.

→ It is Mutable. i.e, value can be updated.

⇒ Creation :-

```
StringBuffer s = new StringBuffer ("Hii");
StringBuilder s = new StringBuilder ("Hii");
```

✡ String Buffer → Multi Threading.

- → Thread safe
- → synchronized
- → slower.
- → 1.0 version.

✡ StringBuilder.

- → Not Thread safe.
- → Not synchronized.
- → Faster.
- → 5.0 version

→ Default capacity of StringBuilder & StringBuffer is 16.

→ if we call String.capacity it will give 16 + Initial string length.

→ To convert StringBuffer (or) StringBuilder to String we use toString method.

☆ System.out.println("Hi");

↓ Class ↓ variable. ↓ Instance method.

↓ Print stream (class).

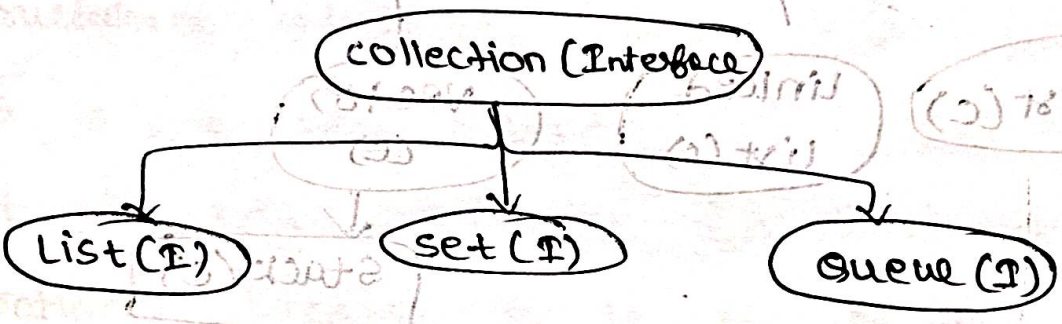
☆ Collections :-

→ collection framework.

→ In java collection is interface.

→ To overcome drawbacks in array collections is introduced.

→ Iterable is the Parent class of collection.



☆ List :-

→ It accepts duplicates.

eg:- 1, 2, 1, 2, 1, 2

→ It preserves insertion order. (indexing)

☆ Set :-

→ It does not accept duplicates.

eg:- 1, 2 ✓ eg:- 1, 2, 1 X

→ This does not follow insertion order.

(Not indexing).

☆ Queue :-

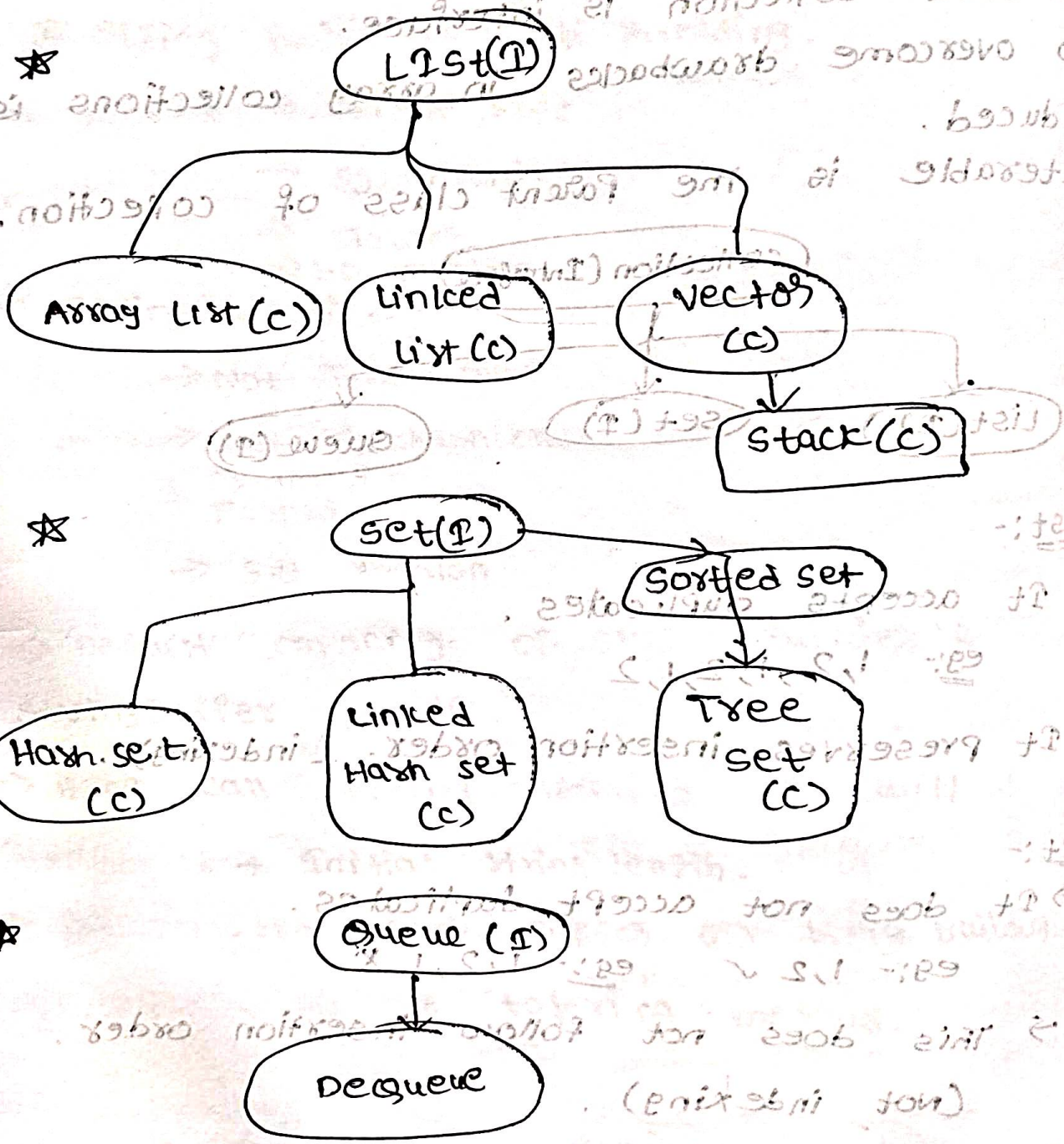
FIFO.

→ collections accept Hetrogenous data types.
i.e, we can store diff data types.

→ collection framework does not work with primitive data types.

→ It works only with classes.

→ It don't have fixed size.



⇒ Array List (Java 2.0) :-

→ It accepts Heterogeneous data.

→ It does not work with data types it works with classes.

→ It don't have fixed size.

→ It also have indexing.

→ when we create an array list it internally creates an array with default size 10.

→ If we enter 11th element the array will be restructured internally with size 10 X 1.5 = 15.

→ 15 is a new size.

⇒ It is not preferred for frequent write operations because it is slower that it has to restructure if we increase size.

⇒ It works faster for read operations i.e, bcz of indexing.

Syntax to create Array List :-

```
ArrayList a = new Arraylist();
```

→ Arraylist has diff inbuit methods.

→ It accepts infinite nulls.

☆ Generics :-

→ This is used to define the data type used for collection.

eg:- < > used with this symbol.

→ By using this it will not accept duplicates & diff datatypes

ArrayList<String> - only strings.

→ In real time we use generics bcz it accepts Homogeneous data (same data type)

☆ Iterator :-

→ hasnext() → it is used to iterate the elements of Arraylist

→ It works with all collections

→ It works only on forward operation.

☆ LIST Iterator :-

→ It works with lists.

→ It can operate in forward & backward direction.

→ hasNext, hasPrevious.

↓ To go next

↓ To come front.

☆ Set :-

→ It does not allow duplicates.

→ It does not maintain order.

⇒ Hash set :-

→ same as normal set. → accepts 1null

→ NO order.

→ NO duplicates

⇒ Tree set :-

→ sorts the values in ascending order.

→ NO duplicates. → ~~null accepts~~.

→ NO order → Null not allowed.

⇒ Linked Hash set :-

→ Maintains order.

→ NO duplicates.

→ accepts 1null

☆ Map :- combination of key-value pair.

→ key is unique in a map.

→ if you repeat the key value, then the value will be overided.

→ values can have duplicates

→ can use Generics.

Types :-

→ Hash Map.

→ Tree Map

→ Linked Hash Map.

```
Map = {key1 : value1 ;
       key2 : value2} .
```

☆ Hash Map :-

→ Key is unique in Map.

→ if u repeat the key value will be overrided.

→ values can have duplicates.

→ Have Generics.

→ NO order & NO indexes.

→ It can have multiple null values & 1 null key.

☆ Tree Map :-

→ Key is unique.

→ if u repeat the key value will be overrided.

→ Values can be duplicates.

→ sorted in ascending order for key.

→ NO indexes

→ It can have multiple null values but no keys.

☆ Linked Hash Map :-

→ same as above two but

→ It follows insertion order.

→ It can have multiple null values but 1 null key.

☆ Internal implementation of Map :- (For eg:-)

→ Map is combination of key-value pair.

→ Map internally has 16 Buckets. by default.

→ Each bucket acts as Linked List. (Tree) Later versions

→ whenever we insert a key-value pair, a Hash code is generated for key.

→ It calculates the index using a formula.

→ index = (Hash & size -1)

→ Here '&' is Bit operator works with 0's & 1's

→ eg:- (Hash & 15) ⇒ gets 0 to 15 value betw
(16-1)

→ The above value decides in which bucket should the key-value pair enter.

→ Load factor = 0.75 (75%)

→ when the 75% of buckets (i.e 12 buckets) are filled then the map is resizes as default size x2 = 32.

→ Now, the map buckets will be restructured bcz. the formula & size changes.

i.e, Hash & 32-1

= Hash & 31.

→ Set uses map internally. so, the value sent acts as key. so, it don't allow duplicates

★ Collections :-

→ It is class in Java.

→ It is helper class. (or) utility class.

→ It helps us with direct methods like sort, reverse order etc.

★ Variable Arguements :-

→ we can pass any no. of data from 0 to any with specific data type.

→ Syntax

```
public void sum(int... in){
    logic.
}
```

→ we can give any parameters before (int... in) var args. - 1st limitation.

→ we cannot give two var args at a time. - 2nd limitation.

★ clone (from object class) :-

→ duplicating an object.

→ when we duplicate an object it converts its type to object data type.

→ to get original datatype we have to type cast it.

→ It works when we implements cloneable interface and when we override it bcz it's

protected method.

→ If we update after cloning it will not change the value. it is called shallow copy.

→ For all primitive types and string classes it just copies the values.

→ For all user defined object types it copies the reference.

→ Itis called shallow copy.

→ In shallow copy references will be updated.

→ To avoid shallow copy we have to use deep copy i.e, we have to override to new address.

☆ Blocks in Java :-

→ There are two types.

(i) Static Block

↓

works at class level

```
static {

}
```

→ this will execute before main method.

→ this executes when class is loaded by CLSS.

(ii) Instance Block.

↓

works at object level.

```
{

}
```

→ this will executes when we create an object.

→ executes whenever we create an object

→ only once per class

→ From here we can access static methods and static variables.

→ this is executed before the constructor.

→ From here we can access both static & instance variables & methods.

☆ enum :- (ENumeration)

→ It is used to declare constants.

→ Type safe.

→ It is a another type of class.

☆ Linked List :-

→ It has nodes that contains reference values.

→ (i) single Linked list. 

(ii) Double Linked list: 

→ All the methods in LinkedList are same as List.

☆ JAVA 8 :-

→ There are major breaking changes from Java8.

→ After Java 8 coding is in less no. of lines this made readable by making functional programming.

☆ Features of Java 8 :-

1) Functional Interfaces.
2) default and static methods in Inteface.
3) Lambda EXPressions.
4) Method references.
5) Optional class.
6) Stream APIs.
7) LocalDatetime class.
8) collectors.

☆ Functional Interfaces :-

→ It has only one public abstract method.

→ we can mark ~~method~~ /interface as functional inteface by using @functional interface annotation

→ It accepts any no. of default & static methods.

→ But this annotation is not mandatory.

☆ Marker Interface :-

→ It has 0 methods.

→ This is an indication to Jvm to give some behaviour.

☆ Default and static Methods in interfaces:-

→ In Interface we normally use abstract methods which do not have implementation.

→ To give implementation we use default methods and in this default is not access modifier it's a keyword here.

→ default implementation is used to give same method ie with same implementation to all childs.

→ And to avoid problem with multiple inheritance we can use super keyword.

→ For static methods we cannot update in childs.

→ They can be directly called by class name in main method in interfaces but in normal methods we can call them by creating object also.

→ we can override default methods.

→ we cannot override static methods.

★ Anonymus Inner class :-

→ This is a implementation without any name.

★ Lambda Expressions :-

→ This works only on functional Interfaces.

→ while writing a method it doesn't have access modifier, return type, method name.

Syntax :- (;) → {} code.

Case-1 → I have created new class file. (Bike)
→ override start method inside
→ 10 lines of code.

case-2 → I did not create a seperate class.
→ override start method inside anonymys inner class
→ 7 lines of code.

case-3 :- → I did not create class
→ I did not override.
→ 2 lines of code.

→ Based on cases by using lambda expression in third case decreased.
no. of lines of code.

☆ Comparable And Comparator :-

⇒ Comparable :- → It is a functional Interface.

→ It is used to compare.

→ Comparable has one method i.e, compareTo.

→ It understood ~~accepts~~ Negative, Positive, 0.

→ It allows only natural ordering that means by hardcoding like compare with id.

→ It allowed only at class level. because it has one parameter.

⇒ Comparator :-

→ It allows custom ordering. that means we can write a class and pass them.

→ Major difference is parameters.

→ comparator takes two parameters.

☆ Stream APIs :-

→ A sequence of operation happening in a Pipeline.

→ we use streams on collections.

→ In Streams we have two types of operations

→ Intermediate operations. → Done on the go by using methods. eg:- map(), filter().

→ Terminal operation.

→ streams do not modify actual data they will just update and store in pipelines.

→ To get updated list we have "collect" and inthat we have collectors.

collect (collectors.tolist()) → This is called Terminal operation.
toset()
tomap()

→ There should be only one Terminal operation.

⇒ For Hashmap stream will not get directly because it is not a collection by using entryset we can enter.

→ entryset has key & value same as map.

→ when we want to stream on map we have to use entry set to get both key & value.

→ To get only keys we use "keyset" and then stream.

→ To get only values we use "values" and then stream.foreach(value → sysos(value));

→ map() → To do some modification.

→ flatmap() → merges multiple collections into a single collection.

→ stream → It creates a pipeline on that we can perform Intermediate & Terminal operations

→ Parallel stream → multi threading

→ In this order is not guranteed.

→ stream works in order but parallel stream does not works in order.

→ Parallel stream saves time.

★ Grouping function in stream :-

→ It return a map.

eg:- frequency. program

[apple, banana, banana, mango].

apple = 1

banana = 2

mango = 1.

eg:- apple.

program in eclipse.

using chars() → gives Ascii

map ToObj(c → (char)c) → converts to char.

a = 1

p = 2

e = 1

l = 1

★ Method references :-

→ :: → uses this.

foreach.

eg:- ^ (System.out :: println) prints data.

⇒ reduce() in terminal operations :-

→ Reduce works as a loop ! → reduces the list/array into single value.

eg:- nums= [1, 2, 3] → nums.stream().reduce

sum = 6.

multiply = 6.

(0, (a,b) → a+b);

↓ Initial value

↓ logic

→ For product initial value "1".

★ Optional class :-

→ doesn't provide value directly.

→ To store value in optional.

Optional.of()

→ To avoid nulls we use optional.

→ All stream operations are present in optional. so, no need of calling streams

→ To avoid nullPointer exception Optional class introduced.

✱ Local Date & time class :-

→ used to change, format and get local Date & time.

→ we can format date & time also.

→ we can get individual objects (i.e. only date, Hour, minutes etc.,)