JAVA

JDK → have tools for ~~coding~~ Compiling & runtime environment

Java development kit

JRE → JAVA RUNTIME Environment

for running compiled code

Java class library + JVM (interpreter)

First.Java → Javac (Compiler) → First.class (bytecode) → Java First (Machine code) Execution

interpreter

- path is a global variable (Temporary, Environment variable) [set path = c:/---;%path%]
- everything must be in class
- System.out.println("Hello Mohit");
  - System: class (automatically Java.lang.); out: object (static); println: method
- Signature must be same of main method.

(C++ → library, Java → Package)

Reading from keyboard

```
import java.util.*;
```

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

Scanner: class; s: reference variable; Scanner(System.in): construction, for key board input

```
a = s.nextInt();
```

class scanner :-
- nextInt()
- nextFloat()
- next double()
- next() → string
- nextline() → sentence
- next byte()
- next short()
- next long()
- next boolean()
- hasNextInt() → is it integer.
- hasNext float() →

Javap Java.util.Scanner → all method shown

sc.useRadix(2)

Primitive Data types [Core part of compiler]

Default = 0 (false)

- integral: byte 1, short 2, int 4, long 8
- Floating point: float 4, double 8
- char 2 (Unicode :- all other, ASCII :- lang. Eng)
- boolean ?

Note:- For every data type there is built in class available which contain info. about size, range.

Variable :- are used for storing data when the program run then the memory gets allocated.

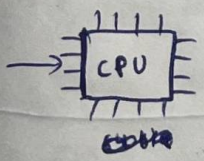
(also have data type) literals :- are Constant value that are used in a program.

5 → integer, 2.5f → Float, 2.5D → Double, ' ' → char

→ otherwise Double literal

Integer.toBinaryString(5)

byte, short, int → int
long → Lorl, Float → Forf
double → Dord, char → ' '



if 64 bit one cycle, big digit more cycle.

→ floating point Number Do not really store decimal in the memory but they represent Decimal number. [mentisa & exponent] [IEEE 754]

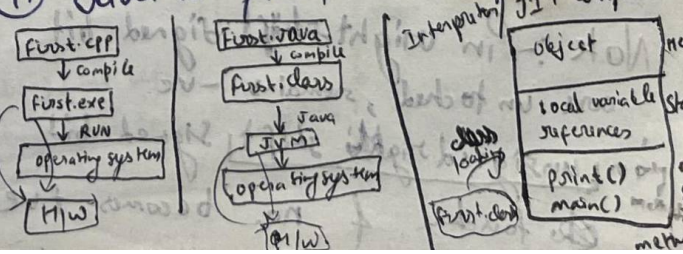
→ upto 6 digit after decimal float else Double. [14-15]

unicode → devnagari [0X0900 — 0X0970]

Interpreter	compiler
• will not generate a separate file	• will generate a separate file
• always required	• require only once
• if there is problem in code it will read untill that error	• if there is error it will not produce any thing until you correct.

Note:- (i) Java is a Hybrid language

(ii) Java is platform independent



Operator and Expression

+, −, *, /

Can be used for (forall)

- int + int
- float + float
- double + double
- char + char

Increment & Decrement

++x, x++

Pre-increment, Post-increment

−−x, x−−

~~Pre-increment~~ Pre-decrement, post decrement

Bitwise operator

AND	∧	&
OR	∨	!
Not	¬	~
XOR	⊕	^
Right shift		>>
Unsigned Right shift		>>>
left shift		<<

Note :- In right shift signed bit are untoched, remains −ve

in unsigned right shift signed bit touche & no. becomes +ve

- % also work for decimal number
  Q → Not in float form
  Divisor, dividend → float form
- For expression to be solve there is precidence you can control by ()
- byte ⊕ byte
  short + short
  byte + byte
  int + int
  short + int

  result be int

  Coercison

- (int) x = b+s | (int) x = c+s
  (int) x = s+i | (int) x = c+i
  (float) x = i + f | (double) x = f + d
  (float) x = l + f | (double) x = l + d

Right shift → $n \times 2^k$ (k → no. of times shift; n → number on which operation is done)

Left shift → $\frac{n}{2^k}$

unsigned right shift

int x = −10;

10 → 00001010

is Comp 11110101

2's Comp 11110110 ← −10

x → 11110110

x>>1 10111011

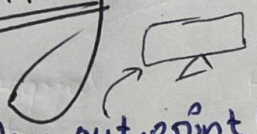
x>>>1 01111011

Merging & Masking :- checking set bit, clear bit, modifying bit, storing a nibble in one byte

Widenning & Narrowing :- Convergen of small size data type to large size (byte type) (short type)

upgrading — Downgrading (typecasting)

Printing



① System.out.print( ) [without new line]

② System.out.println( ) [with new line]

class — static object

```
S.O.P("Hello");
S.O.P(x+y);
S.O.P(x,y); //error
S.O.P("Number is "+x);
S.O.P("Sum "+(x+y));
```

concatination

③ System.Out.printf(string formate, object... args)

```
S.O.P("Hello. %d %f %c \n", x, y, z);
```

Conversion

char	c
int	d, o, x
Float	f, e, g
String	s

%[argument_index$][Flags][width][.precision] conversion

argument index — 1$, 2$, 3$ - -

Flag '-', '+', '0', ' ', '('

```
S.O.P("%3$s %2$f %1$d", x, y, str)
```

String object :- collection of characters

```
String str = "Java Program";
```

class name — reference — literals — string object

Constructors

```
char c[] = {'A', 'B', 'C', 'D'}
String str1 = new String(c);
String str2 = new String("JAVA")
```

Heap ② create two ① objects → Pool

```
String str1 = "Java"
String str2 = "Java"
```

Pool: str1, str2 → Java

```
String str = "Java"
```

str → J a v a (0 1 2 3)

```
int l = str.length(); // 4
String st = str.toUpperCase(); // JAVA
String st = str.toLowerCase(); // java
String st = str.trim()
String st = subString(int begin)
String st = substring(int begin, int end)
String st = replace(char old, char new)
```

heap

```
boolean startsWith(String s)
boolean endsWith(String s)
char charAt(int index)
int indexOf(String s) (5, 4)
int lastIndexOf(String s)
boolean str1.equals(str2)
boolean str1.equalsIgnoreCase(str2)
int str1.compareTo(str2)
```

letters

== → reference

Regular Expressions

R.E	Description
.	Any character
[abc]	Exactly given letters
[abc][vz]	Either first or second set
[^abc]	Except abc
[a-z 1-7]	a-z or 1-7
A|B	A or B
X2	Exactly x2

```
String str = "a"
S.O.P (str.matches (".")); // True
str = "ab"
S.O.P ( str.matches (".")) ; // false
Str = "a", P -> false
SOP (str.matches ("[abc]")); // True
Str = "ab"
SOP (str.matches ("[abc]")); // false
```

Meta characters

R.E	Description
\d	Digit
\D	Not digit
\s	space
\S	Not space
\w	Alphabets / digit
\W	Neither Alphabet or digit

Switch ... Case

```
int day = 1;
switch (day)
{
    case 1 : S.O.P ("MON");
             break;
    case2 : SOP( " TUE");
             break;
    default : SOP( "Invalid day");
             break;
}
```

case → byte, short, int — case 1:

char — case 'a':

string — case "fi":

Loops

① while (Condition)
```
{
  ==
}
```

② do
```
{
  ===
}
while (Condition);
```

③ for (initialization ; Condition ; updation)
```
{
  ===
}
```

④ for-each loop
```
for (int X : A)
{ S.O.P(X)
}
```

int[][] D = int[3]

int[] E, F[] 10 20

Array :- is a collection of similar data elements

$X = \{1, 2, 3, 4\}$ length ← Property

int A[] = new int[5]

reference object

S.O.P(A.length)

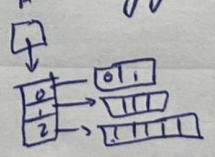
initial value will be zero

int A[][] = new int[3][4];

int A[][] = {{1,2,3}, {4,5,6}}

for(int[] : A){

Jagged array



int A[][] = new int[3][]

A[0] = new int[2]

A[1] = new int[4]

Method

When premitive data type pass copy
when object is passed then reference

```
Class Test
{
    static int max( int x, int y)
    {
    ____________
    ____________
    }
    Public static void main(String[] args)
    {
    ____________
    ____________
    ____________
    s.o.p( max(5, 3)
    }
```

must

if do not want to use static then
Create object of Test & use

Bitwise operator

~ ⟶ Not

& ⟶ AND

| ⟶ OR

>> ⟶ Right shift

<< ⟶ left shift

int b = 17;

int c = ~b ⇒

$b = b_{n-1}\ b_{n-2}\ b_{n-3} \ldots b_1 b_0$

$\sim b = \sim b_{n-1}\ \sim b_{n-2}\ \sim b_{n-3} \ldots \sim b_1 \sim b_0$

$\sim b + 1 = -b$

$\boxed{\sim b = -b - 1}$

left shift

$\boxed{b = 17}$

0 0 1 0 0 0 1

b<<1

0 0 1 0 0 0 1 0

$\boxed{b \Rightarrow 34}$

byte b = 64

byte c = b << 2 [error]

upgrade to int

int c = b << 2

(byte) c ⇒ d = 0

int b = -25; Right shifting

int c = b >> 1

c ⇒ -13

logical

Arithmatic

~~sig~~ Circuler

$\frac{(n-1)}{2}$ ← odd +ve can also work

Right shift preserve sign bit

>>> ⟶ Unsigned right shift

Logical operator

in switch it accept int, char, string

!

&&

||

---

in java array is stored as an object

Heap memory

```
int arr[] = new int[5]; // having zero
int arr[] = {1,2,3,4,5}
int arr[][] = new int[3][5]
```

we can omit it but after at subarray wale time it initialize

```
arr[0] = new int[5];
arr[1] = new int[7];
arr[2] = new int[10];
```

```
For (int x : a)
    sysout (x);
```

```
int a[][] = new int[3][5];
for (int x[] : a)
{   for (int y : x){
        sysout (y);
    }
    sysout ("\n");
}
```

```
labelled break   int a[][] = new int [5][3]

outer: for (int i=0; i<5; i++)
        for (int j=0; j<3; j++)
        {   if (a[i][j] == 10)
                break outer;
        }
```

Further you can use

```
boolean t = true;

first: {
    second: {
        third: {
            sysout ("1");
            if (t)
                break second;
            sysout ("2");
        }
        sysout ("4");
    }
```

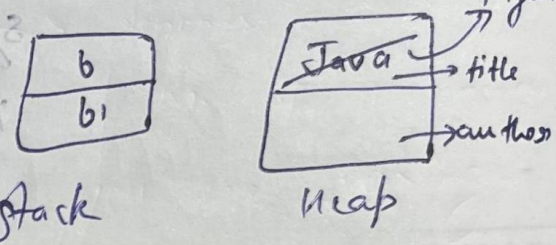
output 1, 4

```
boolean isPrime = true;
int n=23;
for (int i = 2; i*i <= n; i++)
    {
        if(n % i ==0)
            isprime = false;
            break;
    }
}
isprime ? sysout ("prime") : sysout ("Composite");
```

```
class Book {
    String author;
    String title;
}
```

reference variable of type Book

```
Book b = new Book();
```




```
Book b;
b = new Book();
b.title = "Java";

Book b1 = b;
b1.title = "Algorithm;
```

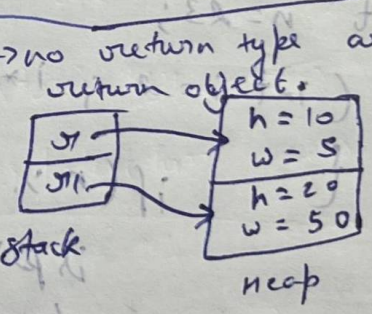
← copy same wali

```
class Rect {
    double h, w;
    double getArea() {
        return h*w;
    }
}

Rect r = new Rect();
    r.h = 10;
    r.w = 5;
    sysout (r.getArea())

Rect r1 = new Rect();
    r1.h = 20, r1.w = 50;
```

→ no return type as it automatically return object.




```
Rect () {
    w = 10;
    h = 5;
}
```

→ Constructor

Or

Parameterized

```
Rect (double h1, double w1) {
    w = w1;
    h = h1;
}

Rect r2 = new Rect(20, 30);
```

~~double getArea () {~~

~~return this.h * this.w;~~

→ when argument or local variable have the same name then this keyword is used for global one

Variable hiding

```
double getArea (double h, double w)
{ ~~return~~ this.h = h;
          this.w = w;

  return h*w;
}
```

```
class Stack {

    int n;
    int data[];
    int top;
    Stack (int n)
    {
      this.n = n;
      data = new int[n]
      top = -1;
    }
};
```

```
Void push (int x) {
    if ( top == n-1)
    { sysout ( " stack overflow"); }
    else {
      data [++top] = x;
```

```
int pop () {
    if (top == -1) { sysout ("Sy stack underflow); }
                return -1; }
    else return data [top--];
```

Method Overloading [Compile time polymorphism]

```
class Test {
    Void mth( ){
        sysout ("Method 1");
    }
    Void mth (int a) {
        sysout ("Method 2");
    }
};
```

```
Test t = new Test()
t. mth(); → Method 1
t. mth (5); → Method 2
```

→ depends on data type & no. of parameter you are passing.

→ from lower to higher type casting is possible

```
class stack{
    int data[];
    int n;
    int top;
    Stack (int n) {
        this.n = n;
        top = -1
        data = new int[n];
    }
```

* if we add my own parameter Constructor then the default Constructor gets removed. for default explicitly you need to add.

★ With this keyword we can call constructor

```
Stack ( ) {
    this (1000);
}
```

should be there

```
Stack (int n)
{
    this. n = n
}
```

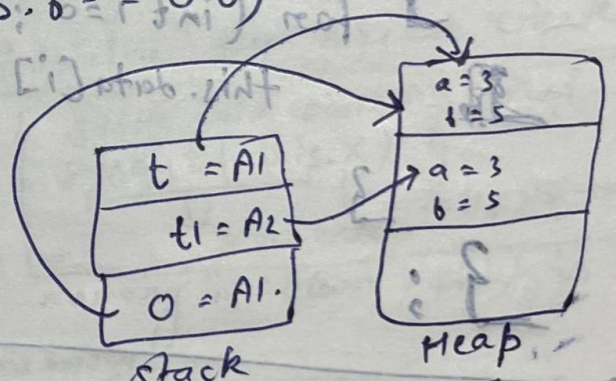
No call by reference in Java

only call by value

- if primitive type copy pass
- if non-primitive type value of reference pass

```
class Test {
  int a, b ;
  Test (int a, int b){
    this. a = a ;
    this. b = b ;
  }
  boolean get equal (Test o) {
      if (this. a == o.a && this. b == o.b)
          return true;
      return false;

  Test t = new Test (3,5);
  Test t1 = new Test (3,5);
  sysout ( t1. get Equal (t));
```



→ true

```
class Test {
    int a, b;
    Test (int a, int b){
        this. a = a ;
        this. b = b;
    }
    Test ( Test o){
        this a = o.a
        this b = o.b
    }
}
Test t = new Test (3,5)
Test t1 = new Test (t);
```

```
class stack {
    int data[];
    int n;
    stack (int n) {
        this.n = n;
        this.data = new int[n];
    }

    stack (stack s) {
        this.n = s.n;
        // this.data = s.data ; this.data = new int[n]
        this.top = s.top;
        for (int i=0 ; i<n ; i++)
            this.data[i] = s.data[i];
    }
}
```

```
class Test {
    static int a, b;
    static Test getTest(int a, int b) {
        Test t = new Test();
        t.a = a
        t.b = b
        return t;
```

Class property

Class method

Test x = new Test(); [redundent ∵ static]
Test y = a.getTest(3,5); [" "]

Test y = Test.getTest(3,5);

object property ~~can~~ cann't be a class propery [each object have its own copy of instance]
but class propery can be a object property [static propery is class property]

9/08/24

1) A Static method can access only static properties.
2) we cann't use this or super keyword. [for this, super keyword we need object but static doesn't need]
reference to current object / parent class

Test t; // Nothing happens. (only memory for storing reference is used)
Test t = new Test() // class inisialize then object created.
Test.meth() // class Inisilize
↳ static method

```
static void staticMethod(){
sysout(this.x) }
```
↳ gives errors → x is Non-static

class Test{
static int a; → it will get memory in class which is special memory.
static void mth(){
a = 5;
}
}

```
Super.mth()
```
→ Parent mth

sysout(t.a); //5

Test.a = 10;
sysout(t.a); //10

* Super keyword refers to the parent object.

→ other object ~~all~~ also access the static property

```
class Test {
  static int Count;
  Test() {
      Count ++;
  }
  static int getCount() {
      return Count;
  }
};

Test t = new Test();
Test t1 = " ";
Sysout(Test.getCount());   // 2
```

* JVM memory ?

+ there is a garbage collector which runs periodically who frees the memory automatically (depends on Jerry)

+ if no one is pointing that memory then it automatically removed.

+ Java support ~~three~~ 3 types of inisializer:

① static inisializer → only for static properties. // invokes when class inisilizer.

② Non static inisializer. // when object inisializer.

③ Constructor :- when an object is inisialized but after Non-static inisializer.

```
class Test {
    static {
        sysout(" from static");
    }
    {
        sysout(" from non-static");
    }
    Test() {
        sysout(" from Constructor");
    }
    static void mth() {
        sysout(" from mth");
    }
}
```

static inisializer

non-static inisializer

Test t; // → space is created to store the address of Test object.

Test.mth() → from static
from mth

Test t = new Test(); → from static
from non-static
from Constructor

Access specifiers

learn from assignment

1. Public → anyone
2. protected → child
3. default [Not keyword] → only accessible ~~only~~ in same pack
4. private → No-one except ~~the~~ within class.

```
class Test {
    public int a;
    protected int b;
    int c;
    private ~~int~~ int d;
}
```

* Outer class can only default or public.
* when all the Java files are in same folder then it call package.

Inheritance 13/08/24

Note:- class can extends only one class

```
class A {
    public int x;
    public int y;
    public void m(){
        sysout("In m of A
    }
}
```

```
class B extend A {
    ~~B b new~~ void m1(){
        x = 10;
    }
    B b = new B();
    b.m();  → In m of A
    b.m1();  → x = 10
}
```

```
A b = new A();
b.m1() x//error
```

method overriding [Run Time polymorphism] ✗ First check for subclass

```
class B extend A {
    void m1() {
        x = 10;
    }
/*  public void m() {
        sysout ("In m of B"); */
```

```
B b = new B();
b.m() ; In m of B

A a = new A()
a.m() ; In m of A

A b = new B()
b.m() → In m of B
to a ~~subc~~ ✗
```

we can assign a subclass object superclass variable

```
A a = new B();
a.m1() ; ✗ [Not in parent]
a.m() ; ✓
```

```
B b = new B() ✓
A a = new A() ✓
A c = new B() ✓
B d = new A() ✗
b.m() ; ✓
b.m1() ; ✓
a.m() ; ✓
a.m1() ✗
```

```
C.m() ✓
C.m1() ✗
```

* before child, parent initiate
* every class in Java, whose parents has not been assigned, has object [Class named object]
* By default the child class invokes non-parametrized constructor of parent.

LAB

```
class Employee {
    String name ;
    String Id ;
    String designation ;
Empolyee (String name, String Id, String d){
    this.name = name ;
    this.Id = ID ;
    d ____ ; }
```

```
class M extend E {
    M( String name, String d){
        super (name, id, "manager")
    }
}
M m1 = new M ("user1", "6,1")
```

Method overriding :- redefining a method of superclass in subclass.

dynamic method dispatch :- Super class reference holding an object of subclass ~~against~~ and overrided method is called then method of object called

[usefull for Runtime polymorphism]

Note :- only have return diff.

```
class A { }
class B extends A { }
class super {
    public A display() { }
    return new A(); }
class sub extends super {
    public B display() {
    return new B() }
}
```

Note : ① override not possible for static & final method
② visibility by access specifier can not reduce

14/08/24

```
class A {
    A() {
        sysout("A's Constructor");
    }
    {
        sysout("A's non-static");
    }
    static {
        sysout("A' static");
    }
}
```

```
class B extend A {
    B() {
        sysout("B's Constructor
    }
    {
        ("B's N-S");
    }
    static {
        ("B's N-S");
    }
    static void m() {
    }
}
```

```
B b = new B();
```

A's static
B's static
A's non-static
A's Constructor
B's non-static

① A's class
③ A's object
② B's class
④ B's object

surely going to inisialized

```
class A {
    public void m() {
        sysout ("Inside m of A");
    }
}
```

```
class B extends A {
    public void m() {
        sysout ("m of B");
    }
}
```

overriding method

```
class C extends A {
    public void m(int a) {
        sysout ("m of C");
    }
}
```

```
A a;
a = new B()
a.m() ---> m of B
a = new C()
a.m() ---> m of C
```

```
class A {
    public void m() {
        sysout ("Inside m of A");
    }
    Static A getfactory (String child) {
        Switch (Child) {
            case "B" : return B();
            case "C" : return new C()
            default : return null;
        }
    }
}
```

* well known designed technique
* there are layeres
  → if lower level designer change then the upper level

```
A a = A.getfactory("B");
a.m();
a = A.getfactory("C");
a.m()
```

→ geter & seter method.

```
class shape {
    public double getArear() {
        return 0.0;
    }
    static shape getshaped (string s) {
    switch (S) {
        case "circle": return new circle();
        case "rectangle": return new retangle();
        default : return new shape();
    }
```

Variable argument

```
public void setDimensions(double... params){ }
```

variable arguments.

Abstract class & Method :- if a class have atleast one abstract method

if a method is not defined just declared.

Note:- can not be static & final.

Purpose :- ① polymorphism (overriding) ② Inherit

```
abstract class shape{
    public abstract double getArea();

    public abstract void setDimensions(double... params);
}
```

* circle has to implement getArea & setDimension

```
class circle extends shape {
    double radius;
    Public double getArea(){ return 2* Math.pi * radius * radius; }
    public void setDimensions(double... params) {
        this.radius = params[0];
```

20/Aug/24

```
Package P1;

interface I1
{
    =
    =
}

Class Doctor implements I1
{
    =
    =
}
```

```
package P1;

Class Doctor
```

```
Class Parents {
    Void m1() {
        sysout("In parents");
    }
}
```

```
Class C1 extends Parents {
    Void m1() {
        sysout("In C1");
    }
}
```

```
class C2 extends Parents {
    Void m1() {
        sysout("In C2");
    }
}
```

whether it is present in or not if not error

```
Parent p = new C1();
p.m1();  ——> In C1

p = new C2();
p.m1();  ——> In C2

Parent p = new C1();
p.m1();
```

Note:- if it is not abstract then it is Concrete

Ref:- F.g. 1.1

Visibility	SC (same class)	SP SC	DP SC (Diff. package sub class)	DP NSC	SP NSC
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	×	✓
Default	✓	✓	×	×	✓
Private	✓	×	×	×	×

Note:- Private Constructor do not allow to make object

for creating package just use package
for importing import packagename.classname

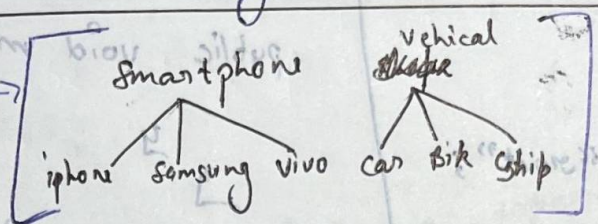
```
abstract class shape {
    Public abstract double getArea();
    Public static shape getshape(string type)
        switch (type)
        Case "C"  return new circle()
        Case "R"  return new Rectangle()
        Default  return new shape()
}
```

```
Class Rectangle extends shape
    Public double getArea() {
        return 10.0;
    }

shape s = shape.getshape("C")
    S.getArea();
    S = shape.getshape("R")
    S.getArea();
```

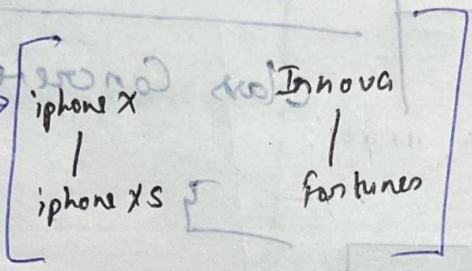
* JVM can make object of abstract class
* Final class means you can not inherit it
* implementing or overriding can not reduce visibility

⊙ Interface
[Generalization] →



⊙ abstract class

⊙ Inheritance
[specialization] →



Interfaces :-

purpose is to achieve only Polymorphism

```
interface Test1
{
    void meth1();
    void meth2();
}
class Test2 implements Test1
{ : public void meth1(){
    }
    public void meth2(){
    }
```

Note: ① class can implement multiple interfaces.
② To achieve runtime polymorphism.

```
class Phone {
    void call(){
    }
    void sms(){
    }
}

interface Camera{
    void click();
    void record();
}
interface MusicPlayer{
    void Play();
    void stop();
}
```

```
class Smartphone extends Phone implements Camera, MusicPlayer{
    void videocall(){
    }
    void click(){
    }
    void click(){
    }
    void record(){
    }
    void Play(){
    }
    void stop(){
    }
}
```

Static

usefull for representing the information of a class not about the objects

MyPack1

```
class P1
{
}
class P2 extends P1
{
}
```

MyPack2

```
class Q1
{
    P1 p = new P1();
}
class Q2 extends P1
{
}
```

```
SmartPhone S = new Smartphone();
Phone p = S (call & sms only)
Camera C = S (click & record only)
MusicPlayer m = S (Play & stop only)
```

f.g. 1.1

☆ interface

1. variable created in interface are static final public (by)
2. private access specifier allowed only.
3. interface object can not make.
4. by defaut public method
5. can have static

reference type

```
interface I1 {
    void m1();
    String DB_PASSWD = "root";
}
```

- private with → public
- signature of mm
- constant
- static method
- default method

```
class C1 implements I1 {
    public void m1() {
    }
}
```

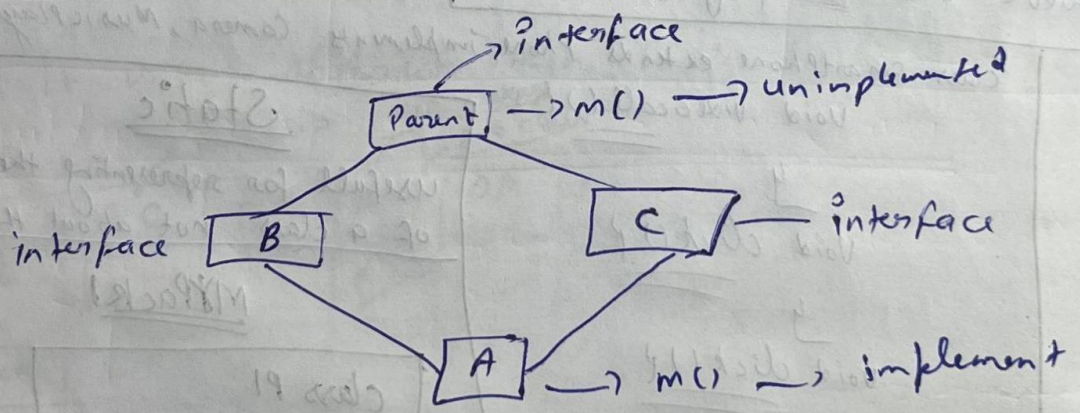
I1.DB_PASSWD

implementation happen by class but now new method is needed then default is used also for private method call inside

```
interface I2 extends I1 {
    void m2();
}
```

```
class Concrete implements I2 {
}
```

Note: interface can only extend interface but can not extend class




```
class A implement B, C
{   Public void m()
    {
    }
}
```

☆ Given example when asked.

* if method is not in interface you cannot invoke by child [if it is in child]

```
interface I1 {
    Public default void m() {
        sysout("Unimplemented");
    }
    void m2();
```

diff. from visibility

implement in the interface body

```
interface DBEntity {
    default void update() {
    }
}
```

```
class Book implement interface {
    =
}
```

```
DBEntity dB = new Book();
dB.update();
```

↓

they are non-static so they need an object to invoke this method.

Package :- is a collection of classes, interfaces or other packages.

⊛ Group the related classes together & organize them in form of package.

```
Package p1;

Public class C1 {
}
Public class C2 {
}
Public class main {
}
```

```
Java/
  ├→ C1.Java
  ├→ C2.Java
  └→ p1
      ├→ C1.class
      └→ C2.class
```

compile file

class path

```
Java -d bin -cp bin
```

current working directory | class path

current directory

search

Herbert Schild

.CPP

Jisa

Exception Handle are runtime errors

Throwable

- Exception
  - Runtime Exception
  - checked exception
- Error

```
int a = 0
int b = 15/a
```

→ arithmatic Exception is an Unchecked Exception.

```
import java.io.FileReader
checked exception {without catch you can not compile}
FileReader file = new FileReader("text.txt");
```

→ error thrown by constructor

```
main(){
m1(){
m2(){
m2(){ /} exception
} } }
```

throw JVM throw

Types of errors

1. Run time errors [Exceptions] → faced by user
2. Syntax
3. logical (Tracing debugger) → faced by programmers

→ { ① Bad input ② Resourse inavailable. } → Handle by Exception Handeling

* We want in case of exception we want our program run smothly except the exception blocks

```
int a = 10
int c = 10/a → Here exception occurs
× Sysout(c);
× Sysout("end of program");
terminates
```

```
try {
int a = 0
int c = 10/a
× Sysout(c);
}
catch (ArithmaticException e){
Sysout(e);
}
Sysout("end of program");
```

- multiple try-catch

```
try {
    ___ ___
    ___ ___
}
catch(..) {
    ___ ___
    ___ ___
}
catch(..) {
    ___ ___
    ___ ___
}
```

nested try catch()

```
try {
    try {
        ___ ___
        ___ ___
    }
    catch(..){
        ___ ___
        ___ ___
    }
    catch(..){
        ___ ___
        ___ ___
    }
}
```

try catch finally

```
try {
    ___ ___
    ___ ___
}
catch (..) {
    ___ ___
    ___ ___
}
catch (...) {
    ___ ___
    ___ ___
}
finally
{
    ___ ___
}
```

clean-up Process

Buildin exception classes in Java

Object → mother class of all Java classes

↑

Throwable

checked exception

Exception

Error {JVM itself handle}

all type of exception handler.

checked

- ClassNotFoundException
- IOException → FileNotFound exception
- InterruptedException → multithreading
- NumberFormatException
- RuntimeException
  - ArithmaticException
  - IndexOutOfBoundException
  - NULLPointerException

} Unchecked

(*) All the Exception class is characterised into two checked & Unchecked.
checked → must handle
Unchecked → optional.

Class Exception have important methods
- string getMessage();
- string toString();
- void printStackTrace();

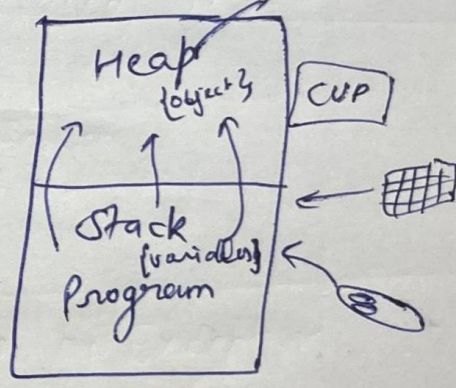
(*) every user defined exception must inherit Exception.

Throw VS Throws

Throw → verb
Throws → adjective, tells that this method throws exception

- Throws :- must ~~use~~ write for checked exceptions.

- All the ~~a~~ things that are outside the ~~heap~~ stack is ~~called~~ resourse for the program.



- whenever a program require a resource require it & when it done with it release it.

- Why we need finally

```
int mth() throw Exception
    FileReader f;
    try{ f = new FileReader("My.text");
        //use file
        ~~f.close()~~
        return result.;
    }
    Catch (-){
    }
    finally{ f.close();
    }
```

```
try (FileReader f = new
        FileReader("my.text"))
{ //use file
  return result
}
```

automatically close

Multithreading

→ extends Thread
→ implement Runnable

- when your program start it always have one thread which is main
- Thread t = Thread.currentThread();
  → Points towards the current Thread

→ sysout(t);
[<name of the thread>, <priority>, <group>]

→ Thread.Sleep(1000) ms → current thread sleep for 1 Sec.

→ Thread t1 = new Thread();
t1.sleep(); → it will effect other thread also.

```
Public Class Main {
    public static void main() {
        Thread t = Thread.currentThread();
        sysout(t);   // Thread [main, 5, main]
        t.setName("OwnThread");
        sysout(t);   // Thread [OwnThread, 5, main]
        for (int i=0; i<5; i++) {
            sysout(i);
            Thread.sleep(1000);
        }
    }
}
```

// 0
1
2
3
4

```
class Thread1 extends Thread {
    private String name;
    public Thread1 (String name) {
        this.SetName (name);
        this.run();
    }

    Public void run() {
        For (int i=0; i < 15; i++) {
            Sysout ("From" + name + " " + i);
            Thread.sleep (1000);
        }
    }
}

class Main {
    P. S. V. main (String [] args) {
        Thread t = new Thread1 ("Another Thread");
        For (int i=0; i<5; i++) {
            Sysout ("From main" + i);
            Thread.Sleep (1);
        }
    }
}
```

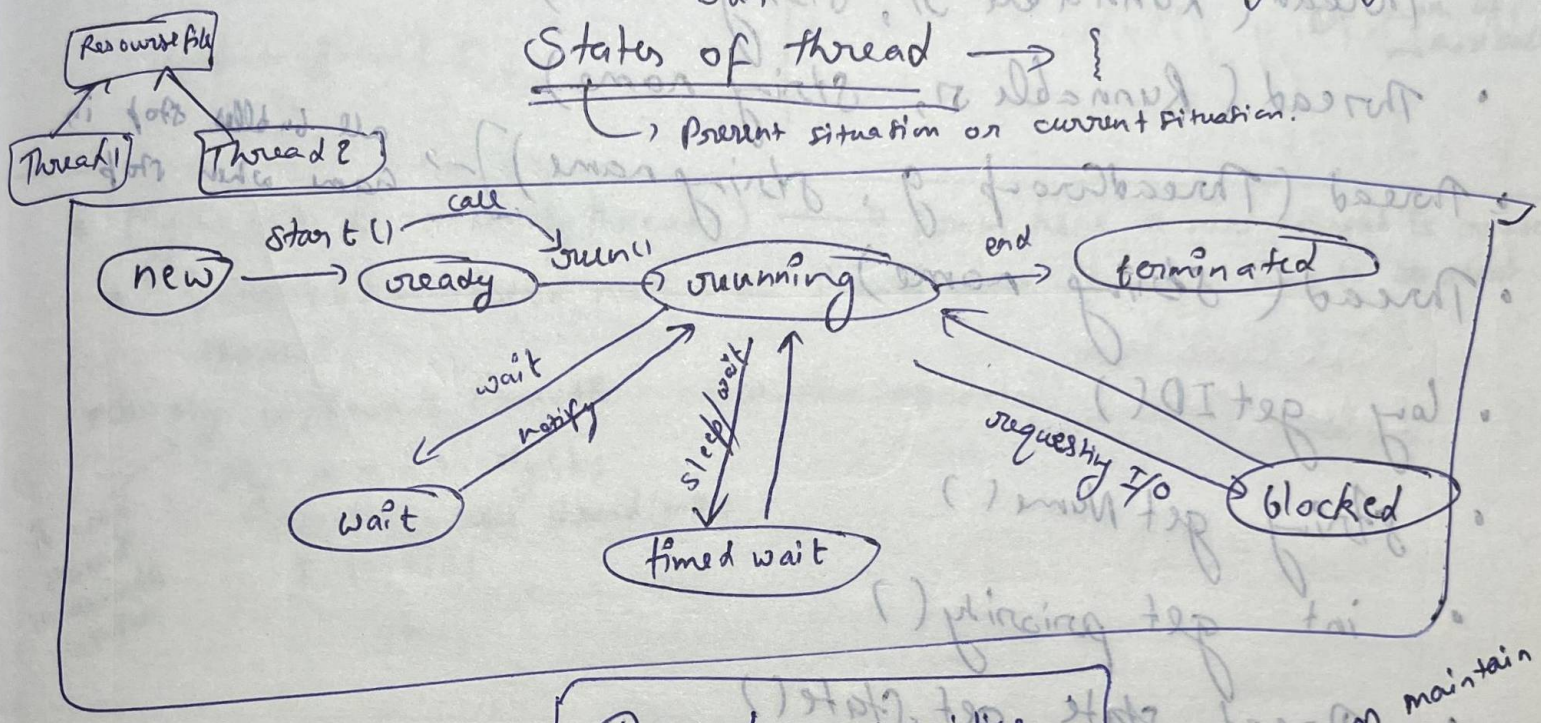
Multiprogramming :- running more then one programs

- Multi-user [one machine multi user]
- Multitasking [running more then one task single user] → [windows, macos]
  - Multithreading → light weight
    - chrome, MS word, Google → advancement

Unix, linux] - multiuser

States of thread →

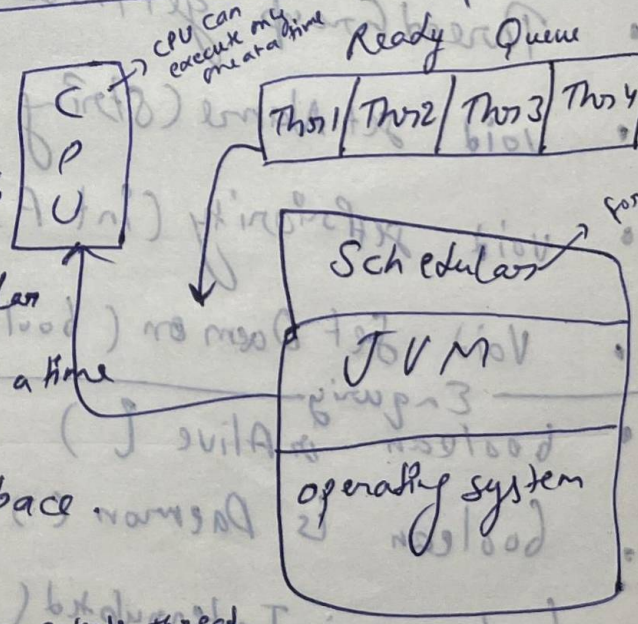
→ Present situation or current situation.



Thread priorities

Thread.MIN_PRIORITY = 1
" Normal " = 5
" Max " = 10

t.setPriority(Thread.MIN_PRIORITY + 2);



- JVM make ready Queue by Scheduler
- CPU can execute one thread at a time
- Time sharing OR priority
- Thread have their own stack space.

For achieving multithreading

1. Thread class → For you own class you can extends thread
2. Runnable interface → already extending from class the use implement thread.

must overrite void run() ←

it is capable of running but it cannot run by itself

Thread class

when ever a thread is created it will get a id by JRE

Constructors

- Thread()
- Thread(Runnable r)
- Thread(Runnable r, String name)
- Thread(Runnable r, String name)
- Thread(ThreadGroup g, String name) → all bubbles stop in same when stop
- Thread(String name)
- long getID()
- String getName()
- int getPriority()
- Thread.State getState()
- ThreadGroup getThreadGroup()
- void setName(String name)
- void setPriority(int P) // given by JVM
- void setDaemon(boolean d) → least priority

Enquiry

- boolean isAlive()
- boolean isDaemon()
- boolean isInterrupted()

Instance method

Void interrupt ()

Void join ()

Void join (long millis)

Void run ()

Void start ()

Static Methods

int activeCount () // how many // thread are active

Thread CurrentThread () // reference // of current // thread

Void yield () // min priority chance

Void dumpStack () ↳ depth of method calls

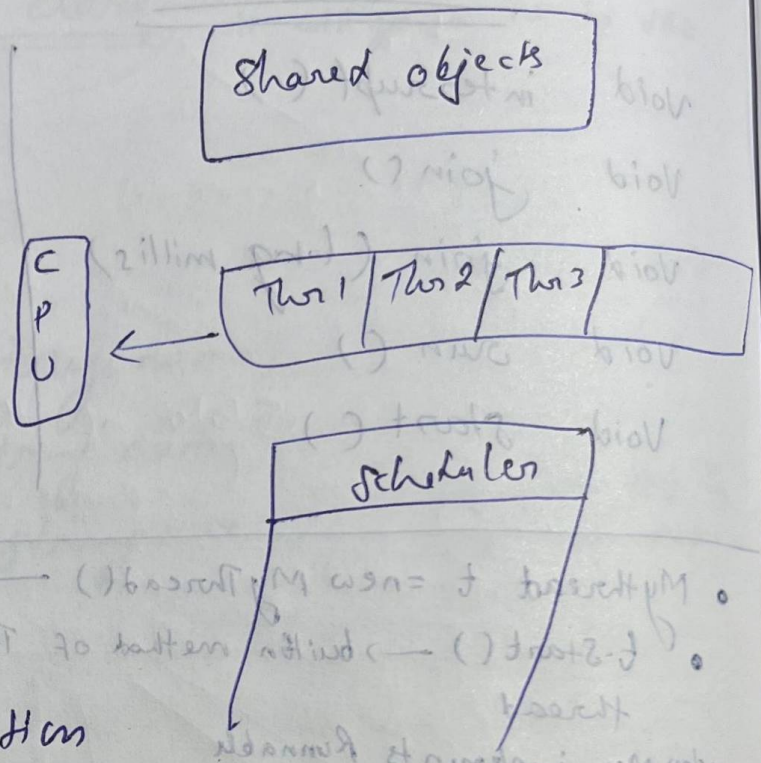
- Mythread t = new MyThread() ⟶ from here a new thread is created.
- t.start() ⟶ builtin method of Thread class it know how to start thread
- class My implements Runnable

```
My m = new My();
Thread t = new Thread(m);
t.start();
```

it not a thread its runable interface

Synchronization

1. Resource sharing
2. Critical section
3. Mutual Exclusion
4. Locking / Mutex
5. Semaphore
6. Monitor
7. Race Condition
8. Inter-Thread Communication



- In our program we have shared object of two Threads object
- you have to synchronized that object, method, variable, which is shared by both.
  piece of code.
- Communication b/w threads this is called interthread Communica
  interprocess
- Race Condition.

Inner class

1. Nested inner class
2. local inner class
3. Anonymous inner class
4. Static inner class

(1) Nested inner class

* inner class can access members of outer class directly
* Inside the outer class you can create the object of inner class
* inner class is limited to outer class [Reducing complexity]
* Directly you can't access the members of inner class but using object you can.

```
class outer
{ int x=10;

  class inner
  { int y=15;

    void indisplay()
    {
      S.O.P(x);
      S.O.P(y);
    }
  }

  void outdisplay()
  { inner i = new inner();
      i.indisplay();
  }
}
```

```
class Test
{
  p.s.v. main()
  { outer O = new outer();
    O.outdisplay();
    Outer.Inner i = new Outer().
                    new Inner();
  }
}
```

firstly create an object on outer class then object of inner class.

* **Note** for accessing inner class entirely in outer class make an object of it inside the outer class. [i.y]

we can also make a class inside a method also & that class is only visible inside the method only.

Anonymous Inner class

→ it can be defined at the time of creation of objects {you can define the class as well as create obj

→ usefull for interface as well as abstract class.

```
abstract class My
{
    abstract void display();
}

class outer {
    Public void mth() {
        My m = new My() {
            Public void display() {
                S.O.P("Hello");
            } ;
        }
        m.display();
    }
}
```

reference

Static inner class

* Static inner class are static ~~member~~ members of outer classes & object of static inner class can be created ~~at~~ outside the outer class [do not need to create object of outer class]

* Static inner class only access static members of outer class.

```
class Outer
{
    static int x = 10;
    int y = 20;
    static class Inner
    {
        void display()
        {
            s.o.p(x);
            s.o.p(y);
        }
    }
}
```

```
class Test
{
    p.s.v.m()
    {
        Outer.Inner i = Outer.Inner();
        i.display();
    }
}
```

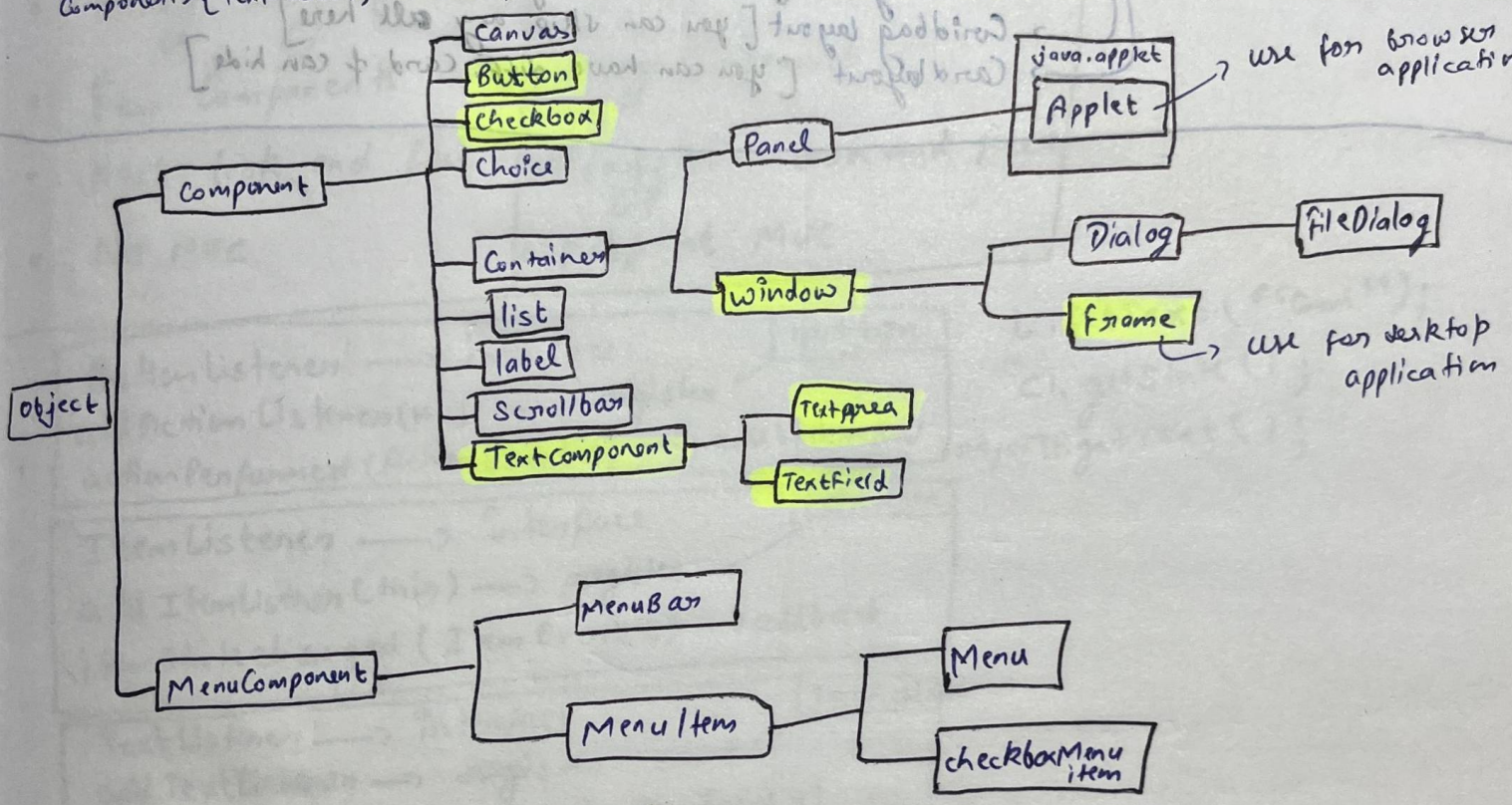
Note :- Not an object of abstract but an object of anonymous class.

interface → GUI {Graphical User int-} **GUI** :- Graphical User interface → is a point of communication to communicate with computer.

- Every o.s. has its own look and feel for its GUI.
- Java provides various frame-work for developing GUI application on window applica.. Computer.

1. AWT → abstract window tool kit
2. swing
3. JavaFx

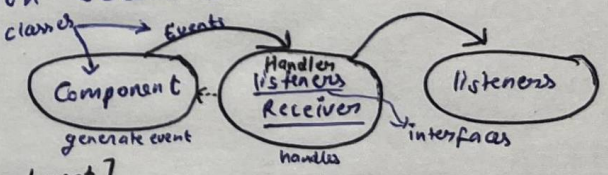
window means without ~~frame~~ boarder and when boarders are given then it is called frame. frame have icon, title, minimize, maximize, exit button. frame is a Container of other Components {text area, textfield, checkbox, ~~Radio~~ Radio etc}



isA → inheritance
has A → Containment

{Event delegation Model}

Component :- is a graphical element which appears on screen which can contain inside a Container.



GUI Components → textare, textfield, checkbox etc.

All components have → Properties, Methods, Events [Something happening on component]
↳ if a button is clicked, selection in list, type in textbox.

when event occurs it must be handle, for handling we have listeners.
↳ call back method is called that method must be there inside handler.

In java we have Components and they generate events we should define our own listeners and the listener should be register with the Components that when an event occurs then the Callback method of that listner is called.

When an event generate the object of this class is generate.

for handling

For handling Item event we need Item listner

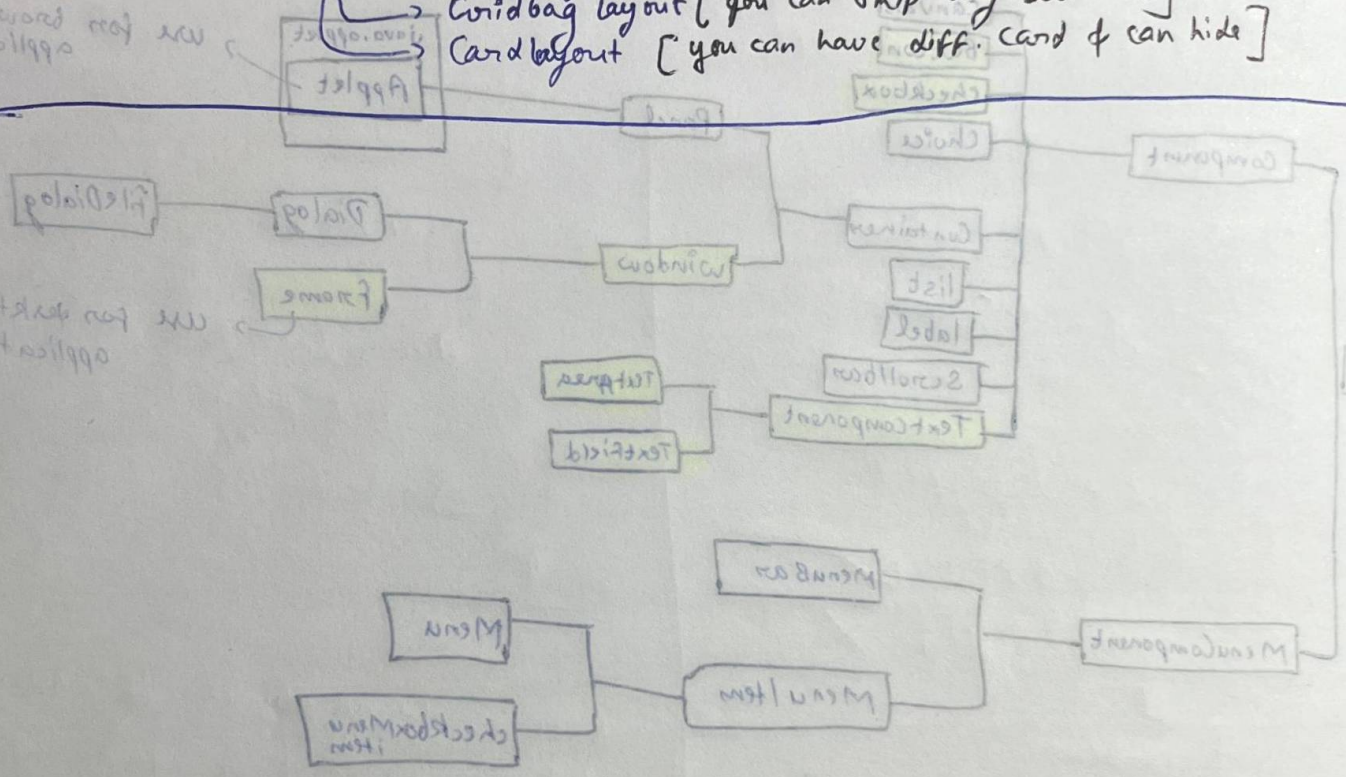
Item event :- Generates when you select an item.
Actionevent :- Generates when you select an item & hit enter.
listbox :- multiple item can select.
choice :- single item can select.

Panel is also container and it can also have components, default layout manager is flowlayout.

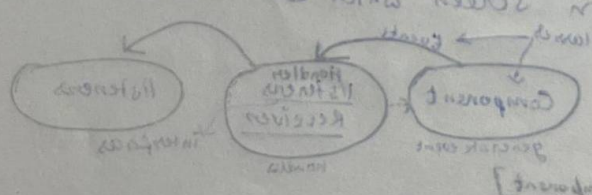
- if checkbox are increasing then use listbox.
- if radio box are increasing then use choice.

layout :- is arrangement of components.

layout manager in JAVA
- → FlowLayout [Components arrange line by line & default alignment center]
- → Boarder layout [Default layout of frame, it have region NEWS]
- → Grid layout [components arrange in Grid structure]
- → Gridbag layout [you can skip any cell here]
- → Cardlayout [you can have diff. card & can hide]



Event delegation Model



Java Swing

abstract window tool kit

AWT VS Swing

borrow defination of button, look & feel from local O.S.

• Platform dependent	• Independent
• Heavy weight [∵ they interact with the O.S.]	• Light weight
• Few Components	• Many Components
• Host look and feel	• Plaggable look and feel
• No MVC	• support MVC

button

ActionListener → interface
addActionListener(this) → register
actionPerformed(ActionEvent e) → callback method

L.setText("Count");
Cl.getState();
strings = Tf.getText();

checkbox

ItemListener → interface
addItemListner(this) → register
itemStatechanged(ItemEvent e) → callback

TextField,Area

TextListener → interface
addTextListener → register
textValueChange → callback(TextEvent e)

Harbert Shield → Book

Java I/O

Stream is flow of data.

Byte stream → 1 byte

Character stream → handle internationalization.

1 Byte stream → [inputstream, outputstream] → interface

Buffered input stream
" O S

Byte Array Input Stream
B A O S

File Input Stream
F O S

Input Stream has read()
Output Stream has write()

System.in → keyboard
System.out → Console [Normal data]
System.err → Console [for error]

Print stream → extends outputstream
→ printf, println, write
→ handle string

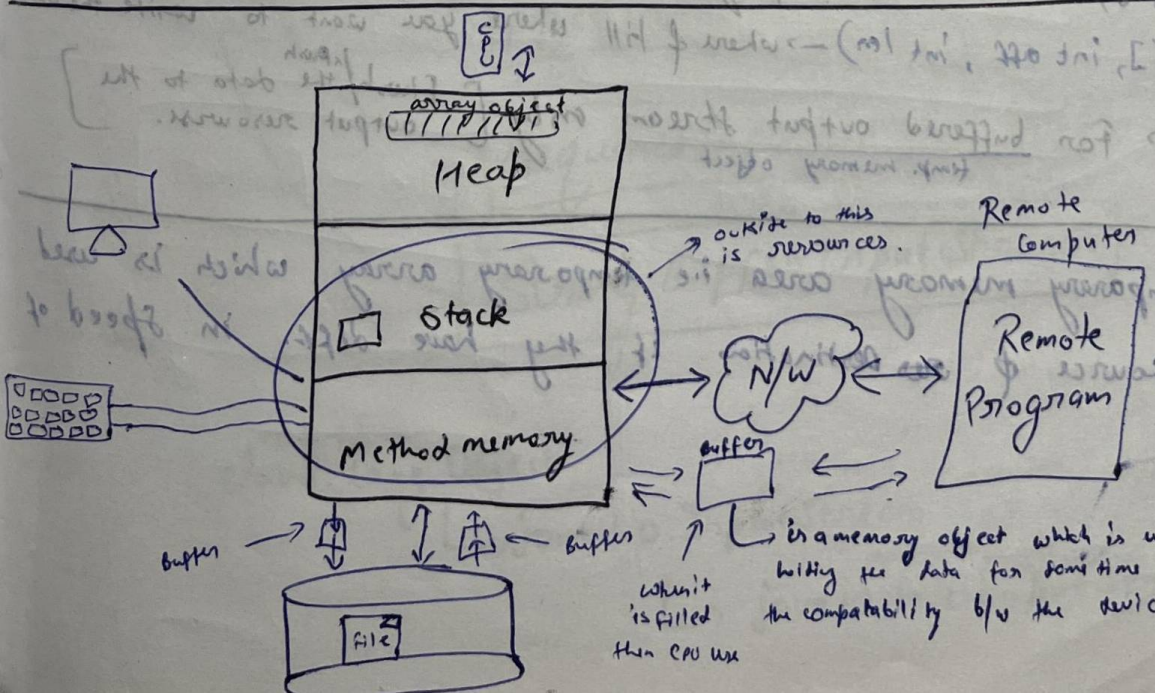
char c = 'A'
System.out.write(c) → only bytes.

Character stream → 2 byte

→ Reader → BufferedReader, FileReader, InputStreamReader.
→ Writer → BufferedWriter, FileWriter, OutputStreamWriter.

transfate byte to characters

Basically data is tranfer charter by character and single unit can also send.



is a memory object which is used for holding the data for sometime to bridge the compatability b/w the devices.

Byte Stream

The base class for Byte Stream is InputStream & OutputStream. to handle Java Io system.

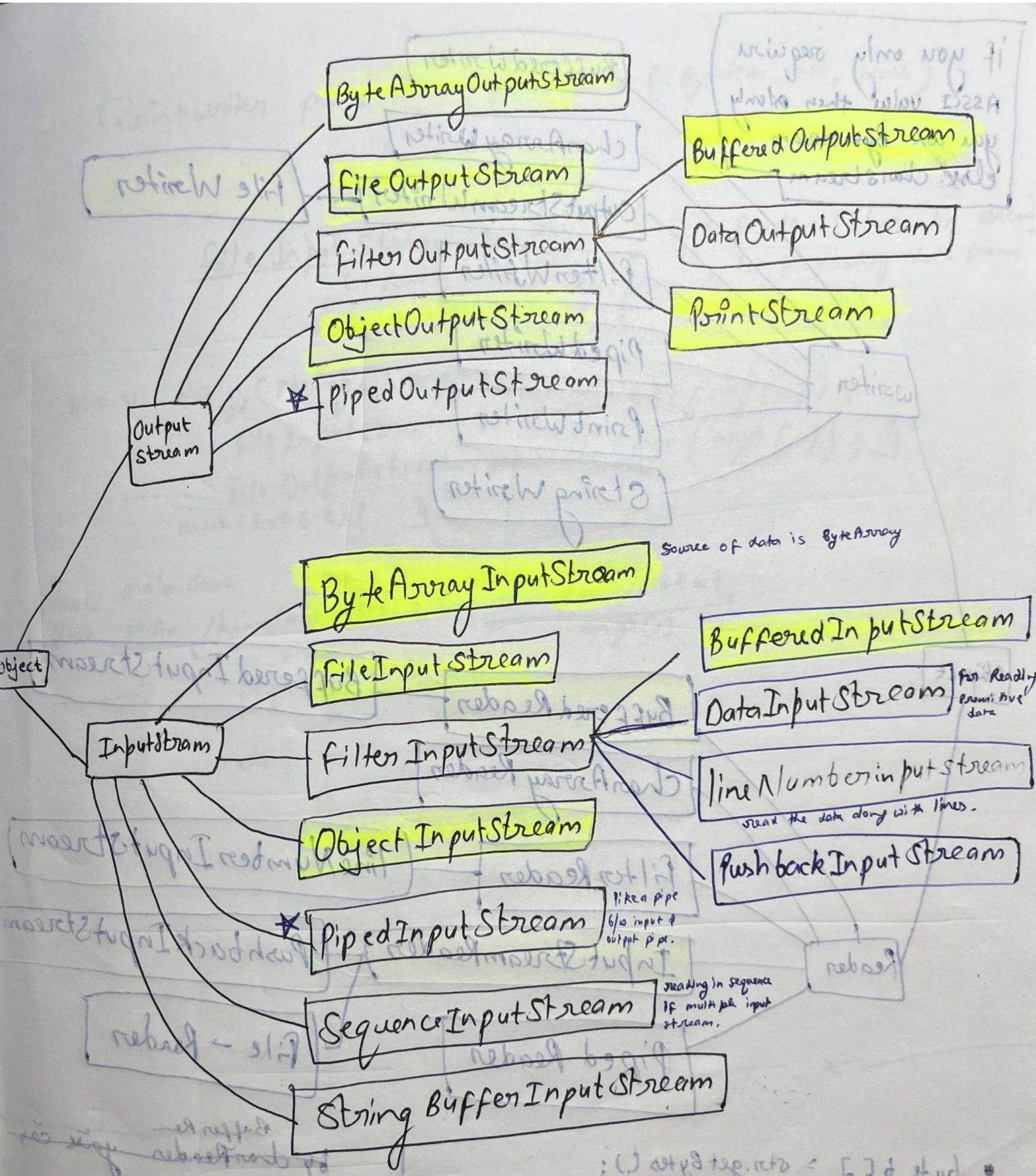
Class InputStream

- int read() → for single byte
- int read(byte[] b) → collection of byte
- int read(byte[] b, int off, int len) → where & till where you want to read the data
- int read(int b) → ASCII No.
- int available() → how many bytes of data available.
- long skip(long n) → you can skip bytes.
- Void mark(int limit) → set limit till where you can reset. [Remain in buffer]
- Void reset() → go to the initial position in mark.
- boolean markSupported() → for checking is it in buffer
- Void close() → for closing stream & resource after use.

Class OutputStream

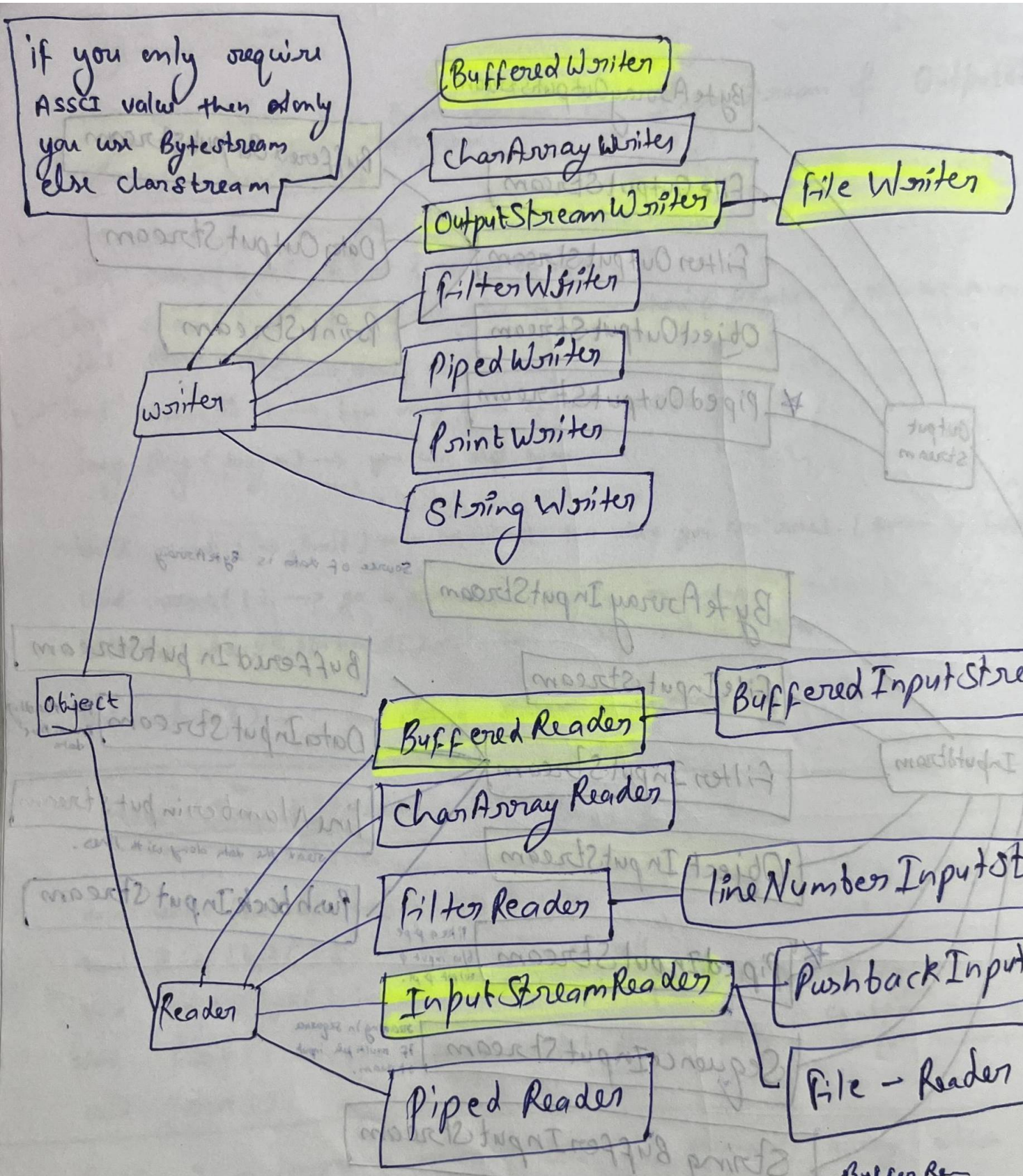
- void write(int b) → for single byte
- Void write(byte[] b) → for collection of byte
- Void write(byte[], int off, int len) → where & till where you want to write the data
- Void flush() → for buffered output stream only [flush/push the data to the output resource.]
  - temp. memory object
- Void close()

Buffer is a temporary memory area i.e. temporary array which is used to connecting source & destination if they have diff. in speed of working.



Java.lang.Object
↳ java.io.InputStream
↳ java.io.FileInputStream

if you only require ASSCI value then only you use Bytestream else charstream



- byte b[] = str.getBytes();
- ~~this~~ String str = new String (bis.readAllBytes);
  byteArrayInputStream
- byte b[] = bos.toByteArray()

~~by charReader you can read lines~~
BufferReader

PrintWriter pw = new PrintWriter(System.out, true)

File Input Stream [Byte stream]
↳ Reading data byte by byte

File Reader [character stream]
↳ for reading data from file

P.S.V main(String[] args)
try{
FileInputstream fs = new FIS(arg[0]);
FileOutputstream fos = new FOS(args[1]); }
catch(FileNotFoundException e){ }

javac Main.Java
Java Main /home/user/test.txt /home/user/text.txt
args[0] args[1]

int i;
do{
i = fis.read();
if (i != -1)
fos.write((char) i);
}
while (i != -1);

~~catch() { }~~

x x x