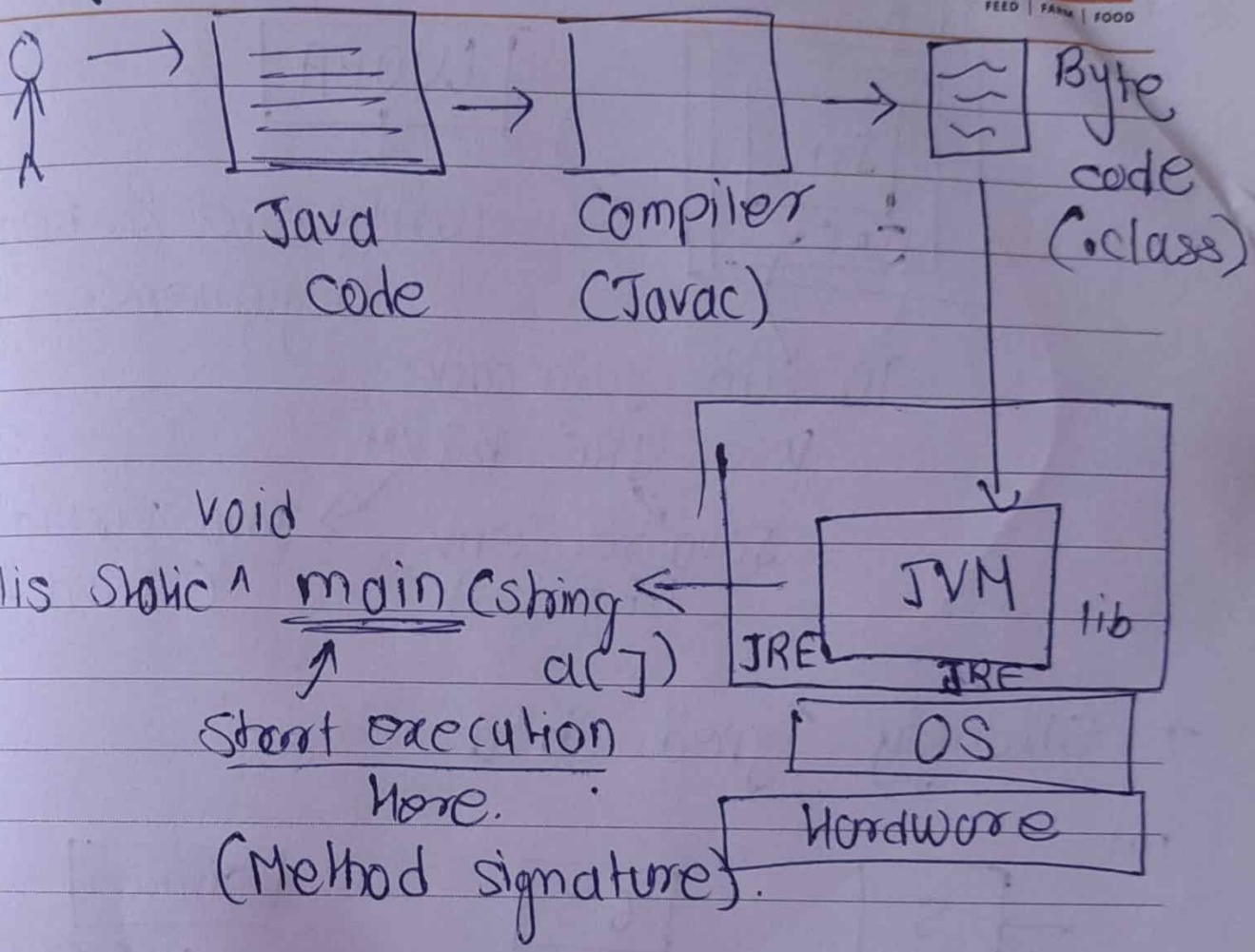
Java is independent language, but JVM is dependent.

Date

JAPFA
MAKE ALWAYS
FEED | FARM | FOOD

IRE → Java Runtime evn.



public static void main (String a[])

Start execution Here.

(Method signature).

public static void main (String a[])

Signature (not run without it).

Java → is oop

∴ every thing in object & class.

Java library → environment.

Date

JAPFA
MAKE GOOD ALWAYS
FEED | FARM | FOOD

JDK → Java Development tool(s)/kit

WORA

write once & Read anywhere.

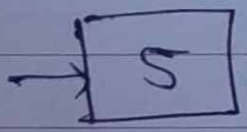
To run Java code use JRE & JVM.

JRE → Java run env. (libraries)

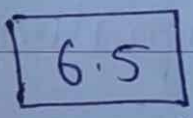
JVM → Java virtual Machine (Java to byte)

* Strongly Typed data.

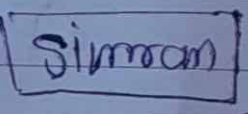
int



num



marks



user

Data type

- Primitive
  - → Integer → byte, short, int, longs
  - → Float → double, Float
  - → Character
  - → Boolean.
- ?

Date

JAPFA
MAKE GOOD ALWAYS
FEED | FARM | FOOD

Integer
→ int → 4 bytes → -2
long → 8 bytes. →
short → 2 bytes →
byte → 1 bytes → -2^7 to 2^7-1
-128 to 127

Float → 4 byte
Double → 8 byte → default

double num = 5.6;
float num = 5.6f;

Char → 2 byte

UNICode → Not ASCII

char C = 'k'; → single quate for char.

Boolean → true or false (0, 1)
× not work here.

bool b = true;

Date

byte by = 129 → exceed length.
by = 127 → OK.

Type casting

① int a = 127 Explicite ~~Implicite~~.
byte b = (byte)a; // casting

② int c = b → conversion / casting (Implicite)

int a = 256
byte b = 256 × error
byte b = (byte) a;

b = 1 (set)

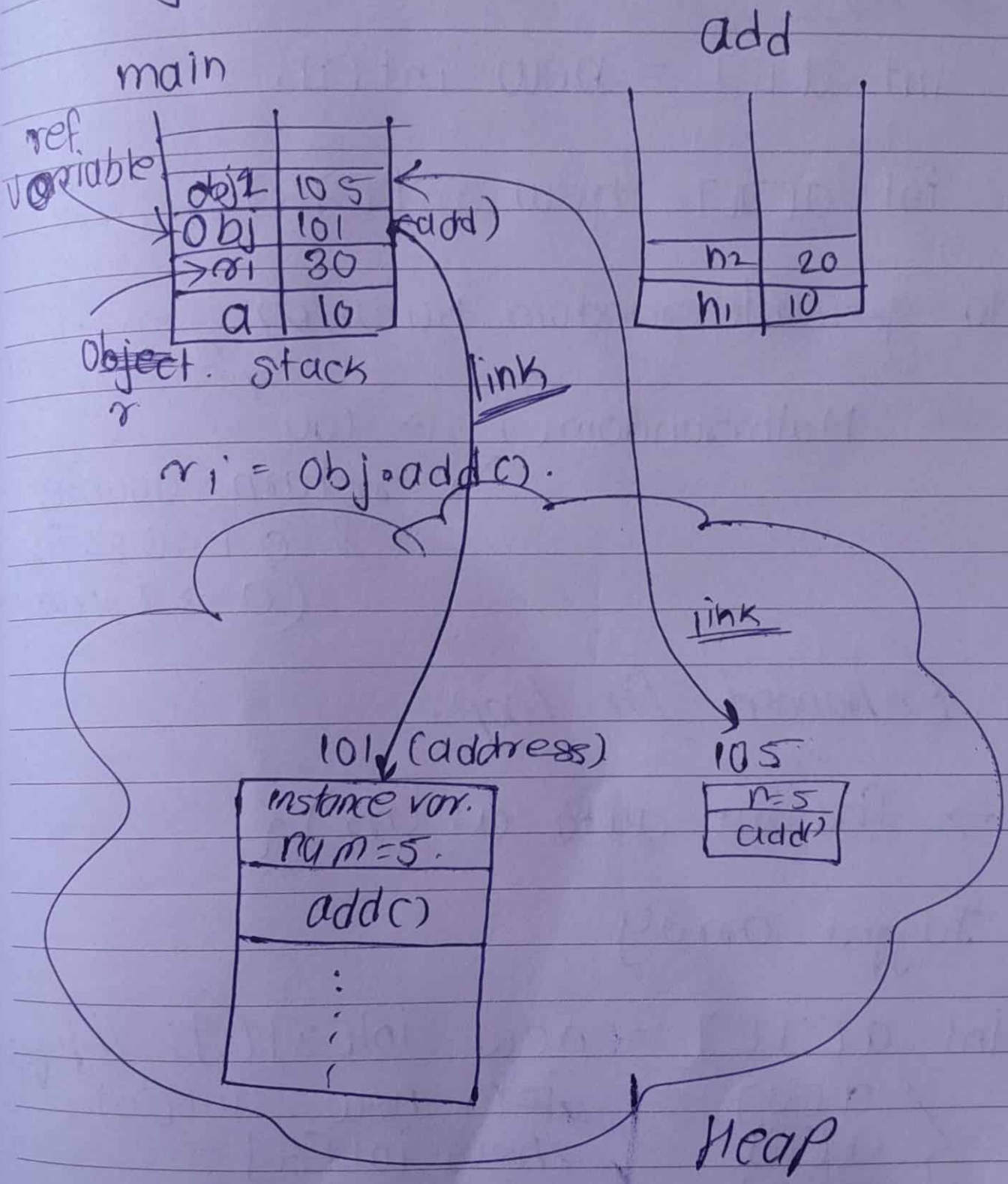
256 % 256 = 0
↓
modulus

* Type ~~Prot~~ promotion

byte a = 10;
byte b = 30;
int result = a * b; // 300.

promoted to int

Every method has own stack.



Date

JAPFA
MAKE GOOD HAPPEN
FEED | FARM | FOOD

Array

```
int a[] = new int[9];

int a[][]= new int[3][6];
```

to generate random Number:

```
Math.random() * 100
```

↳ 2 digit op.

↳ return double
& 0.1, 0.84 like
(0 → 1) range

* Enhanced for loop:

→ for(auto a: arr){

* Jagged array.

```
int a[][] = new int[3][]; // jagged.
```

define seperately:

```
a[0] = new int[3]
a[1] = new int[4]
a[2] = new int[5]
```

Date

~~for (auto a[] : a)~~

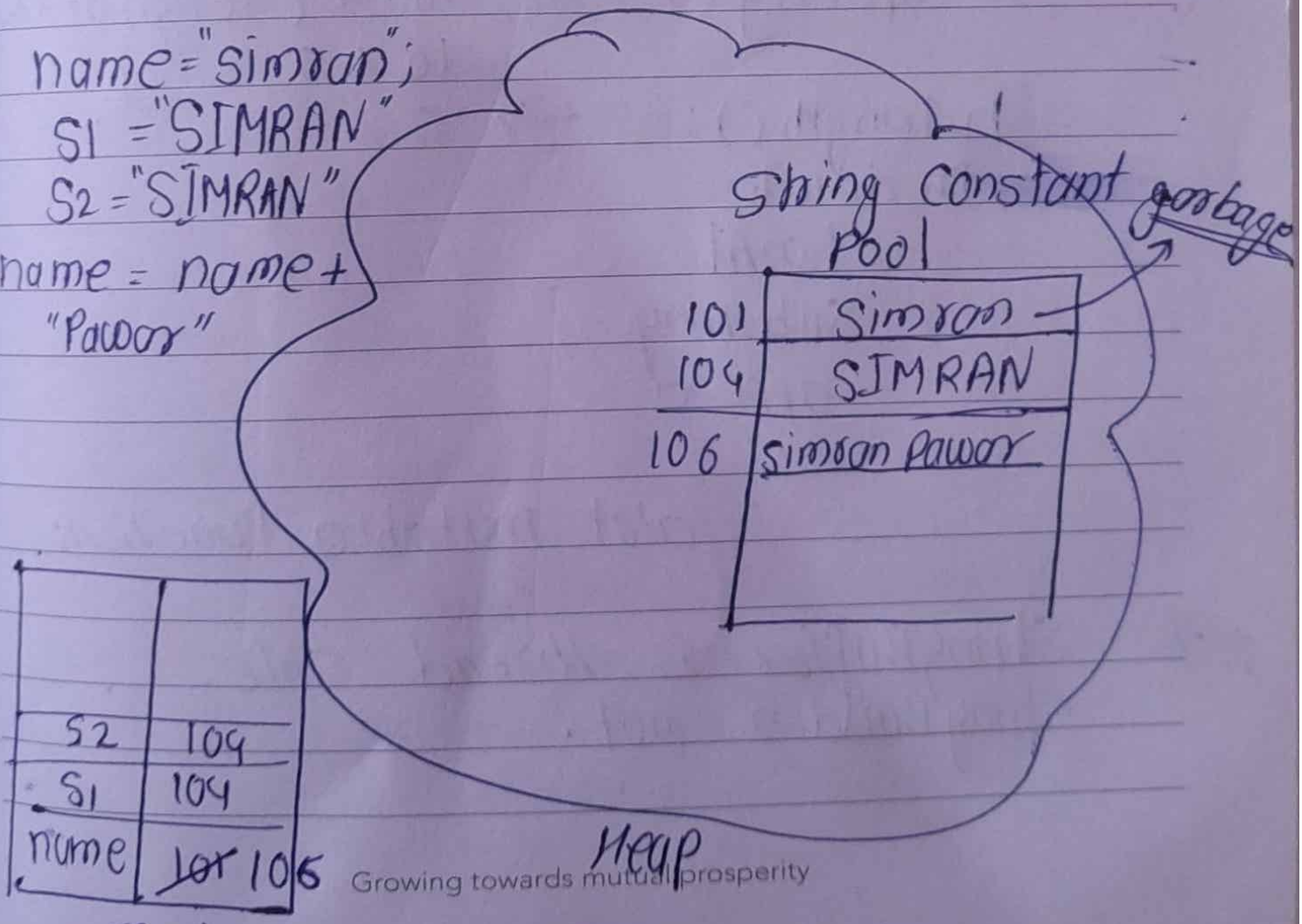
for (auto a[] : a) {
  for (auto n : a) { } MD array

* array - is object.
  It is immutable.

* String.

OR String n = new String ("name");
String n = "name";

name = "simran";
S1 = "SIMRAN"
S2 = "SIMRAN"
name = name + "Pawar"



Date

Strings are immutable (can't change).

mutable string → use → StringBuffer
StringBuilder.

* StringBuffer.

StringBuffer n = new StringBuffer();
n = "Hello";

n.capacity() ⇒ 16 + (length of string)
16 + 5 = 21.

n.length(); → 5.

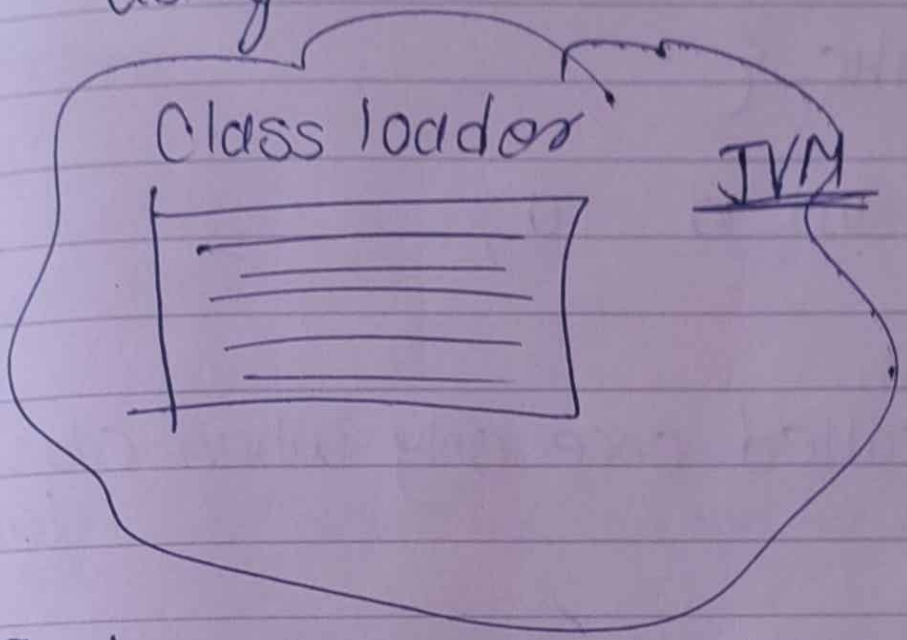
n.delete
charAt
substring
append
.
.
.
alot mutable functions..

* StringBuffer is thread safe.
StringBuilder not.

Date

JAPFA
FEED | FARM | FOOD

* Class loads & object initialization.
using



To load class without obj.

Class.forName("Mobile") → throws Exception.

Class Name{

static int a = 0;

↳ initialize same value for all object.

}

to change value.

* className.a = 10;

Date

* Static block in class

```
static {
    int a = 10;
}
```

↓ called once only when class load.

* Static method :- in class

Call :- ClassName.methodS();

use → static variable } inside
don't → non-static var } static method.

to access non-static variable.
~~paso~~ Pass obj

```
ClassName.method(obj).
        ↓
static vaid method(ClassName obj) {
    obj.name;
```

Date

✡ Encapsulation :-

setter() → set private variable of class

getter() → get a ~~private~~ te. private variable.

✡ by default String value is null.
✡ by default int value is zero (0).

✡ Constructor :-
When object is created, constructor is called.

```
public className(){
    .
    .
    .
}
```

✡ this & super :-
By default, every constructor has super(); → call the constructor of super class.

Date

JAPFA
MAKE GOOD ALWAYS
FEED | FARM | FOOD

we can pass parameter in super(5) for parent class parameterized constructor.

* every class in Java extends object class

- this() ;-> will execute the current class constructor.

-> ~~Camelcase~~ for className.
like ClassName

-> Variables & methods -> small()
show()
showMyTV()

-> Constant = PIE, BRAND

```
{
    new A(); // anonymous object
}
```

can't use later in code.

Date

Inheritance :-

multi level inhe

parent
↑
child
↑
grandchild

} single level inheritance

(Parent) A ↑ C (Child) ↑ B (Parent).

} multiple inheritance. (not work in Java).

Ambiguity problem :- confusing when both parent have same method, child call it, it get confused which one use this is ambiguity.

method overriding :-

same method name, same parameter. It override a parent class reference is most to own class.

Date

* packages :- it structure of folders.

package tool; → to part of tool Folder/ package.

* to use class of tool package outside

use :- import tool.*; → all files
import tool.className; → only class

* default package imported in java.

import java.lang.*;

→ folder of folder / package of package

import other.tool.*;

OR

import other.*;

Date

* in build

MYN Respository → Java package. *

standard way to name your own package reverse domain name;

* package com.google.calculation; *

* Access modifier :-

- public
- private
- default → not mensions → it can ~~use~~ access only in package ~~or~~ not ~~out~~ outside of package.
- protected → Access by derived class. also accessable in own package with subclass or without subclass.

	public	private	protected	default
in class	yes	yes	yes	yes
in pckg. with subclass	yes	No	yes	yes
in pckg without subclass	yes	NO	yes	yes
outside package	yes	NO	NO	No
outside with subclass	yes	NO	yes	NO

Date

✱ Polymorphism.

many behaviour

- Compile time
  - Overloading.
    - add(n)
    - add(n,n,n)
- Run time
  - Overriding
    - A add(n)
    - B add(n)

✱ Dynamic method ~~des~~ dispatch.

Overriding → ~~which~~ method call of which class is not dependent on type. it totally dependent on Object.

```
A obj = new B();
```

imp.

only in inheritance.

Date

JAPFA
FEED | FARM | FOOD

✱ final keyword :-

① final int num = 8; // constant variable in Java

② final class Cal // stop from inherite // ~~from~~ your class.
{
:
}

③ public final void show() // stop from override.
{
:
}

✱ Object class. → extends to every class of Java
obj.toString()
obj.equals(obj2); → can be @override in own class

Date

* Upcasting.

```
Class A {
        :
        }
Class B extends A {
        :
        }
```

* A obj = new B(); => A obj = (A) new B();
*//Upcasting //
// Background

* B obj = (B) obj; // downcasting.

Date

JAPFA
MAKE GOOD HAPPEN
FEED | FARM | FOOD

Abstract

```
public abstract void drive();
    // No implementation.
// abstract method only in abstract
   class.
// we can't create object of abstract
   class.
// if we extends abstract class
// compalsary to implement all
// abstract method.
// if not then declare that class as
   abstract.
// abstract class may have or may
not have abstract methods
```

Inner class.

optional class A {

    static class B {

        obj of A.
        obj.

A.B obj = new B();

Date

If use static to inner class

```
A.B obj = new A.B();
```

✡ ~~Anam~~ Anonymous class

First obj => class implementation

```
A obj = new A()
        {
            methodname(){};
            .
            .
        };
obj.methodname();
```

→ A is extends to anonymous class

✡ abstract & Anonymous class

```
abstract class A
{
    Public ~~void~~ abstract void show();
}
.
PSVM(-.){
    A obj = new A()
    {
        public void show(){
            .
            .
        }
    };
```

Date

JAPFA
MAKE GOOD. ALWAYS.
FEED | FARM | FOOD

* Interface.

```
interface A
{
    void abstract show();
    void abstract config();
}
```

not need

```
class B implements A {
public void show() {
        :
    }
Public void config() {
        :
    }
```

* Variable in interface.

(every)

```
int a = 10; // final & static
```

* ~~Interface can~~

class can implements A, B.

both interface => multiple inheritance

Allow in interface.

Date

* to inherite interface to interface use extends

```
interface x {
    :
}
interface y extends x {
    :
}
```

// class - class -> extends
// Interface - class -> implements

// interface - interface -> extends

* Default method in interface

```
interface test {
    default void show() {
        S.o.p("Test it");
    }
    void squre();
}
```

Interfaces to have method with implementation

ENUMS :- enum → Act as class extends Enum class

```
enum Status {
    Running,
    failing,
    pending,
    success;
}
```

} this is obj of class in C++ they all ~~treading~~ as constant.

```
Status s = Status.Running;
```

s.ordinal() → index. (0 - length-1)

// for all value.

```
Status[] ss = Status.values();
for (Status s : ss) {
    S.O.P(s);
}
```

Date

* switch with enum

```
switch (s) {
    case Running:
        :
        break;
    case Pending:
        :
        break;
        :
        :
    default:
        SOP(...);
}
```

s.getclass().getSuperclass(); → enum.

✡ enum and variable & method.

```
enum Laptop {
    Thinkpad
    Running(price:2000), macbook(10000);
    // constructor
private int price;
    private Laptop(int price) {
        this.price = price;
    };
```

obj

method

also can set & get price with help of getter() & setter() in enum.

```
for(Laptop lap : lap.value())
{
    sop(lap.price);
}
```

We can also define default constructor.

* Override → ensure that method is overriding.

```
@Override
functionNameOverride
```

@Deprecated → can use but don't use it, there is better option

```
@Deprecated
class ClassName{
}
```

→ Anotation

Date

@FunctionalInterface → ensure that

interface A

{ void show();

}

SAM (single abstract method)

Using Lambda (Java 8)

A obj = new A() ⟹ A obj = () →

{

public void show(){

sop("show");

}

};

obj.show()

{

sop("show");

};

(Optional (single statement)) :-

obj.show()

For only FunctionalInterface

It also pass value to lambda function.

* Void show(int i);

↓

A obj = i -> S.O.P("show " +i);

* return value

A obj = (i,j) ->
{
return i+j;
};

OR

A obj = (i,j) -> i+j;

↓

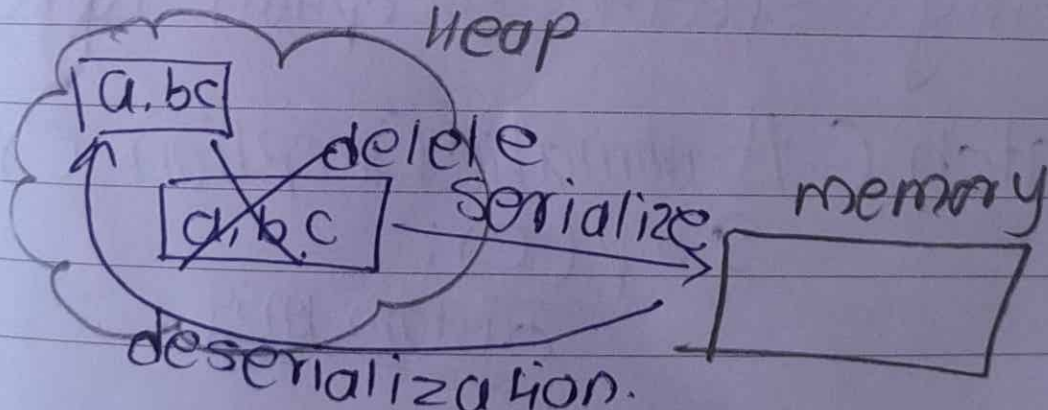
No need to return keyword use

Type of Interface

Normal (abstract methods)

Functional / SAM (single abstract method)

Marker (NO methods)



Date

✡ Exceptions :-

1) Compile-time error
2) Run-time error.
3) logical error.

Handle

```
try
{
    j = 18/0;
}
```

this block has critical statement that can be throw exception

```
catch (Exception e)
{
    S.o.p("Something went wrong");
}
```

this block catch exception

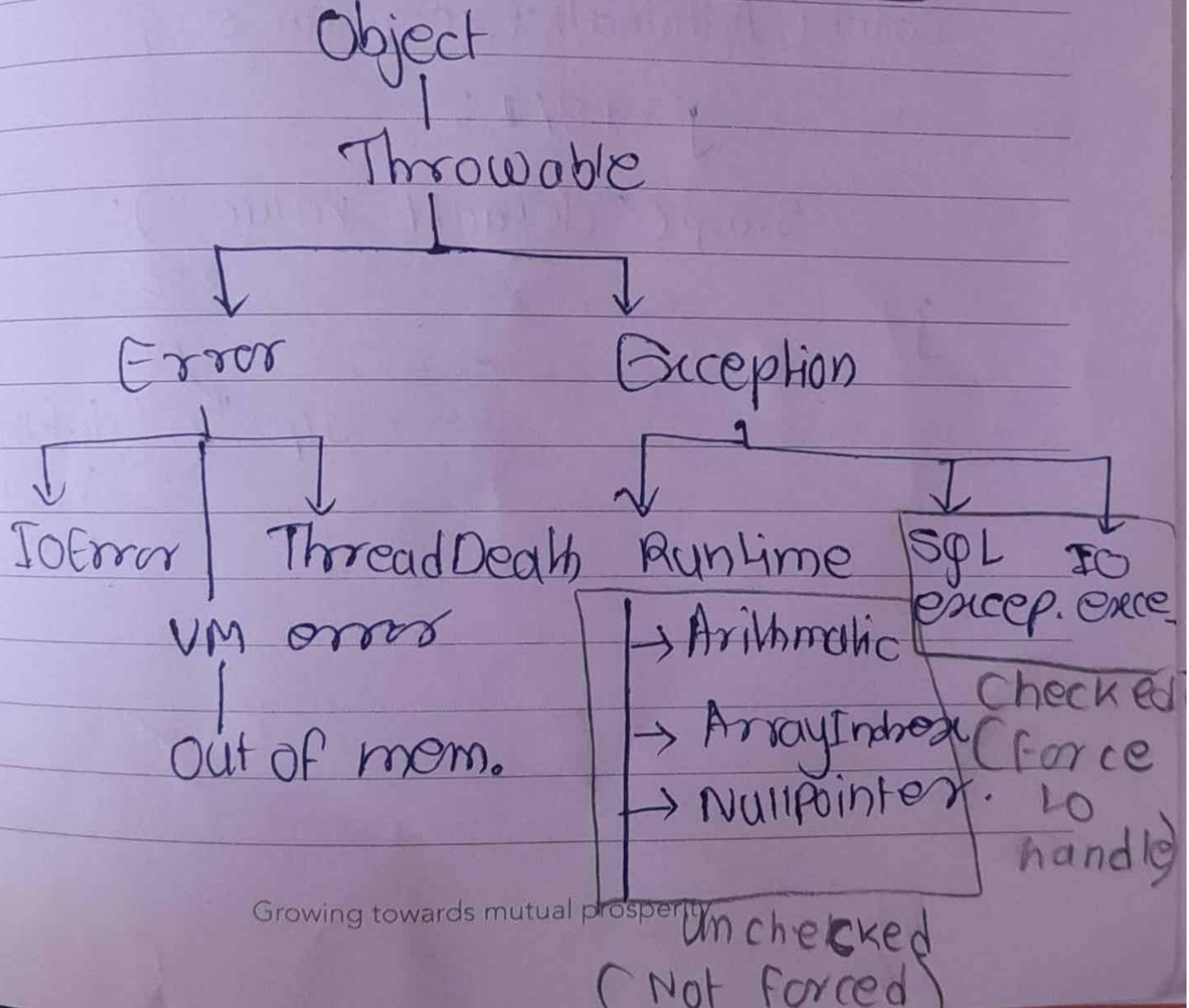
✡ Catching Specific exception type.

```
catch (ArithmaticException e) {
    S.o.p(e);
}
```

Custom msg

```
catch (ArrayIndexOutOfBoundsException e) {
    // print custom msg.
}
catch (Exception e) {
}
```

This class handle everything (Parent class)



Date

JAPFA
FEED | FARM | FOOD

* Throw:-

```
try
{
  j = 18/i;
  if(j==0) {
      throw new ArithmaticException();
                                  " message"
  }
}
Catch( ArithmaticException e) {
      j = 18/1;
    S.o.p(" default value ");
}
```

, e

o/p :- (" default value, message")

User defined Exception class

RuntimeExcep

OR

```
class SimranException extends Exception
{
    public SimranException(String str){
        super(str);
    }
}
```

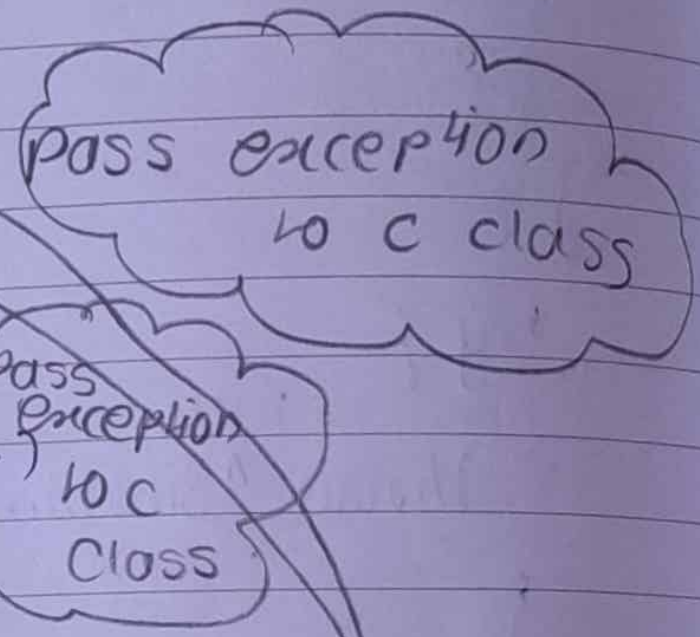
"pass "message" to exception calss.

```
try {
    throw new SimranException("message");
}
catch(SimranException e) {
    S.o.p(e); // message
}
```

Date

✱ Throws Exceptions.

```
Class C
{
  =
  =
  =
  try
  {
    d();
    a();
  }
  catch(C -
  {
    :
  }
}
d() throws Exception
{
  :
}
a() throws Exception
{
  :
}
```



* Checked Exception :- Compulsory to handle exception.

```
Class A ~~throws~~
{
    Void show() throws Exception.
  {Class.forName("method");}
}

main() ~~throw~~ throws ClassNotFound-Exception{
    obj.show();
  }

catch(e) {
e.printStackTrace();
}
```

main() fun ↑

there we try to throws exception to JVM. It can stop execution.

(e.printStackTrace) → print all class stack where problem arrived

Date

✱ taking input from user :-

① int num = System.in.read();

reads one char at time
return ascII value.
not for multiple char.

② BufferedReader.

InputStreamReader in = new InputStreamReader(System.in);

Object of in class

BufferedReader bf = new BufferReader(in);

Object of InputStreamReader

int num = Integer.parseInt(bf.readLine());

S.o.p(num);

return String

Convert to Integer

BufferedReader can read from anywhere

Date

bf.close(); → close resource

✱ Scanner :-

```
Scanner sc = new Scanner(System.in);
int num = sc.nextInt();
```

✱ try - finally → It will run if an exception or not

```
try {
    :
    :
}
finally {
    // statement at last runs.
}
```

used to close a resources in finally block.

like bf.close()

BufferedReader can read from anywhere

Date

bf.close(); → close resource

✱ Scanner :-

```
Scanner sc = new Scanner(System.in);
int num = sc.nextInt();
```

✱ try - finally → It will run if run exception or not

```
try {
    :
    :
}
finally {
    // statement at cast runs.
}
```

used to close a resources in finary block.

like bf.close()

Date

br will close (autoclose) when try block ends

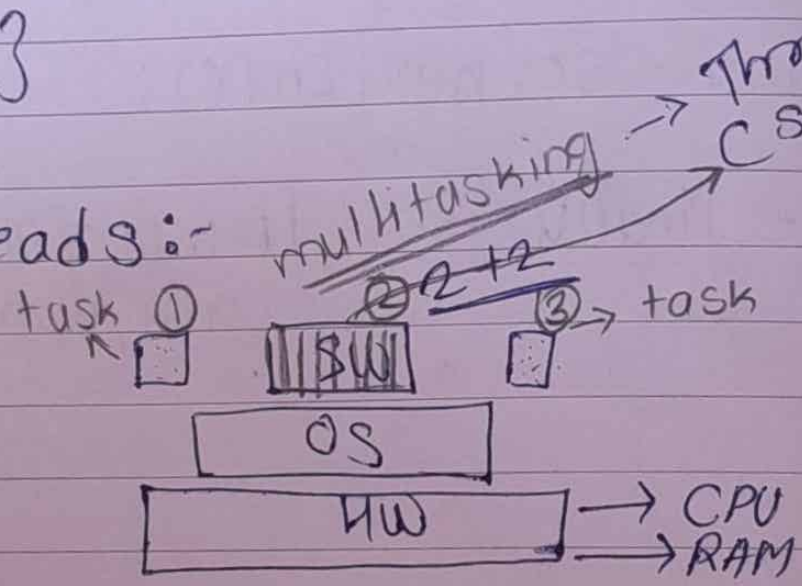
JAPFA
MAKE GOOD GROW
FEED | FARM | FOOD

try (BufferedReader br = new BufferedReader(new InputStreamReader(System.in))){

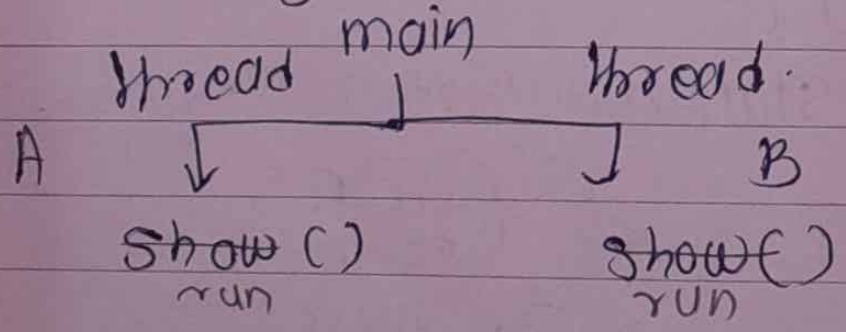
:

}

✡ Threads :-

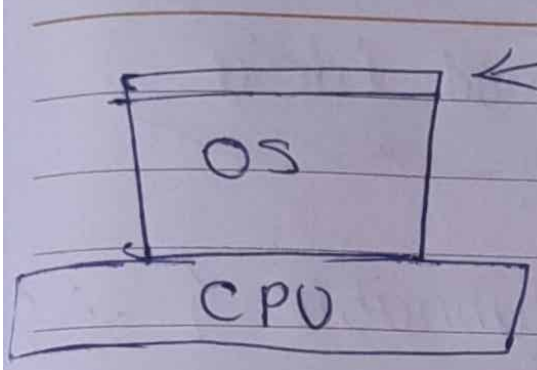


✡ Simultaneously run two method at time



~~There is~~ every Thread has start()
-> run() to override.

Class A extend Thread { . . }
Class B extend Thread { . . }



✱ Priority of Thread.

obj1.getPriority() → integer.

get

set priority → obj1.setPriority(Thread.MAX_PRIORITY);

different scheduler have, different algorithm, here we only suggest to what to done.

```
class A extends Thread
{
 public void run() {
        for (int i<=100)
         S.o.p("j");
        Thread.sleep(10);
```

millisec

Wait to Thread

checked Exception

Date

extends Thread => Not good idea

↓ another way

class B implements Runnable { ... }

* obj1.start() → not work here.

* A obj1 = new A(); X
  B obj2 = new B(); X

↓

Runnable obj1 = new A();
Runnable obj2 = new B();

Thread t1 = new Thread(obj1);
Thread t2 = new Thread(obj2);

t1.start();
t2.start();

Date

* Anonymous class for Runnable

```
Runnable obj1 = new Runnable()
{
    :
    :
    :
}
```

⇓ convert into lambda.

```
Runnable obj1 = () -> {
    for(int i=0; i<=100; i++) {
        s.o.p("hello");
        try{ Thread.sleep(10); } catch..
    }
```