8/3/21.

Modifiers

Access Modifiers
① public
② private
③ protected
④ default.

Non Access Modifiers
① static
② final
③ strictFP
④ abstract
⑤ synchronised
⑥ transient
⑦ native
⑧ volatile.

1) public → same class, same package માટે પણ different package માટે પણ.

2) private → same class પૂરતું

3) protected → same package પણ આવી ગયું. package ના child class

4) default → Same package પૂરતું.

1) static → ① data member → memory એકવાર જ storage લેવા માટે વપરાય
② static Method → વાપરવા માટે non static ના બદલે object ના બનાવ્યા વગર સીધે સીધા call કરી શકાય.
* local static ચાલતા નથી.

2) final → ① data member → to create constant variable
② final methods → prevent method overriding
③ final classes → prevent inheritance
* we must initialize final variable.

3) Synchronised
- multithreading માટે multiple threads same resource ની માંગણી કરે છે.
- એક વખતે એક thread can access the resource
- આ java નો way છે જેમાં threads ને synchronised keyword માટે execute થાય છે using "Monitors".

4) transient → serialization
Object નું conversion file માં કરવામાં આવે
એટલે serialization.
transient keyword serializn રોકવાનો..
security purpose માટે.

5) Native → java ની hardware સાથે interaction
C/C++ થી કરે.
C/C++ માંથી funcn use કરવાનો જરૂરિયાત પર
Native

6) abstract
funcn ની declaration કરવાના એ keyword
use થાય
આવી આવા funcn ની defin ફરજિયાત child class
માં આપવી જ પડે. (આ class abstract જ હોવો જોઈએ?)

7) strictFP.
ત્રણ PU આવેલા
1) GPU → Graphical processing unit → જે CPU
નો બધું load માંથી ને distribute કરે work
થાય
GPU નો ઉપયોગ rendering માટે થાય.
2) FPU → floating processing unit એ આવેલ
બધા floating ના ઉપયોગ કરવા એ keyword
વપરાય છે.
બધા processor અલગ અલગ o/p આપે
પણ strictfp લખવાથી o/p same મળે છે.

8) volatile.

volatile int *P = (int*) mem_addr → don't perform any optimisn on P
anything which can change beyond compiler scope you should declare it volatile.

~~Computer Network~~

JVM Architecture.

Javac નામ tool ઓટે આપણો input લઈને આપણી prg. java નામના ની direct Ram પર run થઈશકે નહીં તેથી Java Demo કરીની command આપવાથી ~~e~~. JVM એ O.S કને demand કરશે પણ O.S ના class loader નામના ની file હોવાથી JVM પોતાની class loader કરે દેશે

Programe.java —javac→ ...

↑

એ Ram પર હોવાથી (process)

javac Ram પર run હોવા છતાં પણ આપણો input લઈને આપણી file નથી.

Q. ~~#~~
- .pdf
- ~~jav~~ .exe.
- .pptx
- —
- —

} એ બધાને O.S directly run કરતું નથી.

⟹ ~~Market માં રોજ નવીન નવીન applicⁿ આવે.~~

પણ પ્રત્યેક વખત O.S ને કોઈ નવ નિયમ instructⁿ set લખવું મોકલ શકતું

આપણા કોઈ extension ને નિયમ કરીને આપવામાં આવે

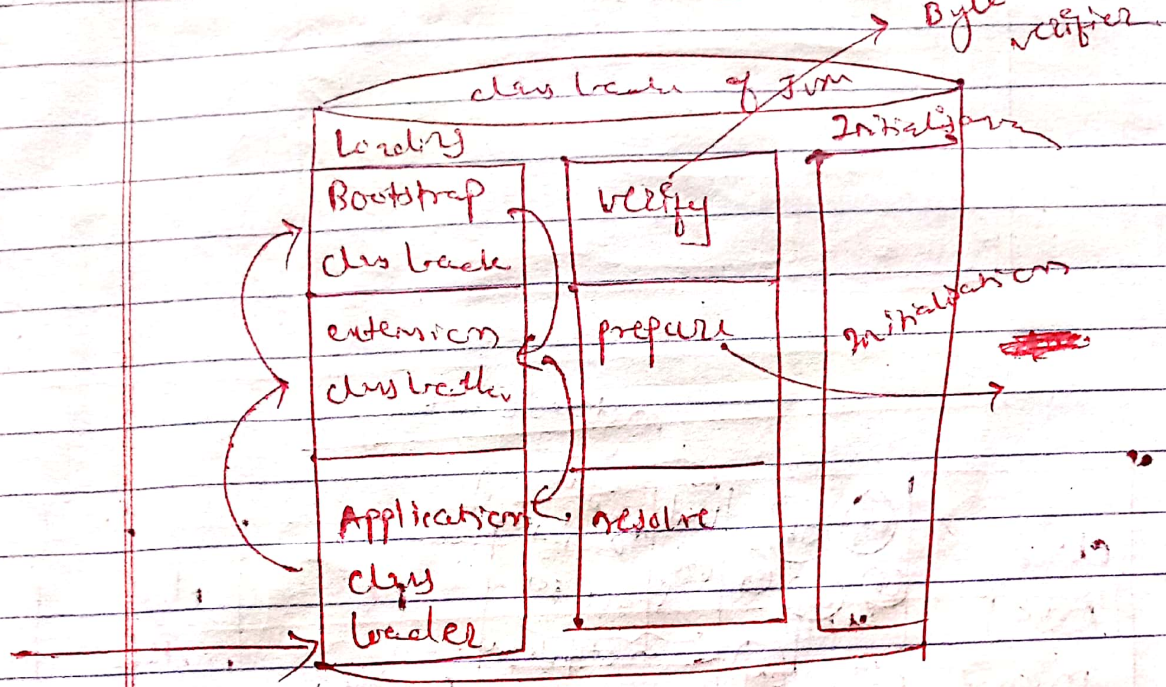
∴ જે તે applicⁿ type નું હોય તે run કરશે.

device driver → જે company printer બનાવે તે જ આપે device driver આપે. જેનાથી O.S ને સમજવામાં તથા device સાથે interaction કરવામાં મદદ કરે.

* Virtual address કેટલા generate થવાના થયા

32 bit $2^{32} - 1$ physical addresses RAM

64 bit $2^{48} - 1$ size



* A.C.L કહેશે call E.C.L ને અને નિચેના call B.C.L માં મળ execute થવાના B.C.L માંથી નીકળ extension class loader અને Application.

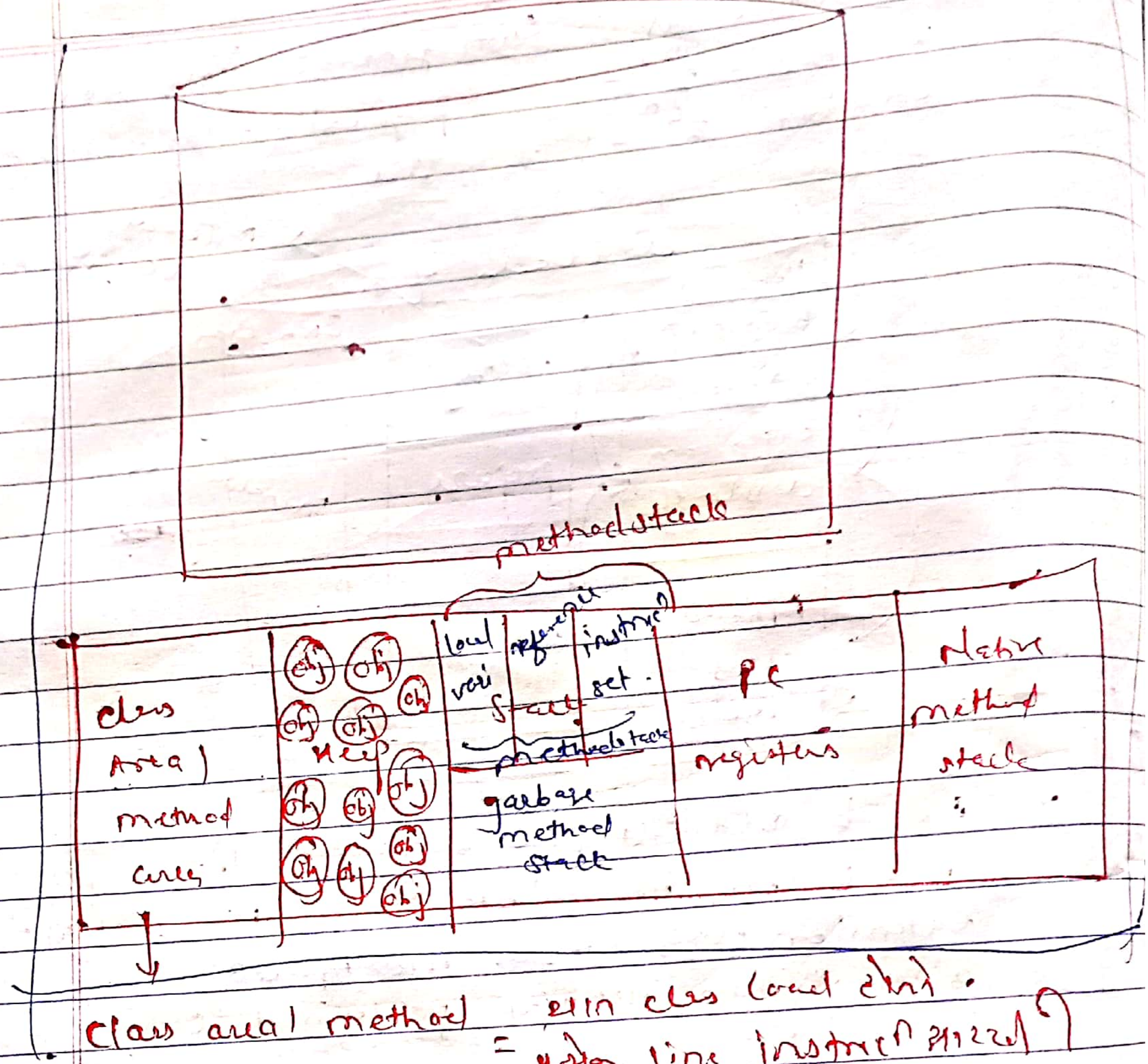
* કોરલા મળવાના Handling predefined classes મળતા

prepare → static field માં memory આપી initialised to 0.

Initialisation → static field initialised to user defined value.

static int a = 10 આપણે નહીંતર આપી 0 માં initialise થશે અને 10 ગણાં કરશે મળશે

static int a = fun() ← fun માં call થશે તો garbage વેલ્યુ આવિએ લેશે મળશે 0 માં initialise કરતા દેખાય.



Class area / method area = દરેક class load થાય.
= પ્રત્યેક line instruction સાચવે
store થાય. variables પણ
instruction સાચવે ~~સ્ટેક~~ અહીંયા. ત્યાંથી memory ફાળવાય
નહિ
જ્યારે function માં call થાય ત્યારે તે function કયું
કરવાનું stack પર તેનું execution માટે જગ્યા
ત્યાંથી variables માં value મળે.

Heap → બધા object heap પર સ્ટોર થાય.

Stack → function નું execution

A) ① method stack → method માં call આવવાથી
કરવું થાય તેનું execution
ત્યારે 3 part.

① local variable → funcⁿ માટેના local variables ની memory લે છે.

② reference → જે heap પર રહેલ ઓબ્જેક્ટ objects માટે મારો reference રાખે છે.

③ function set → execution માટેના instruction.

B) Garbage method stack →

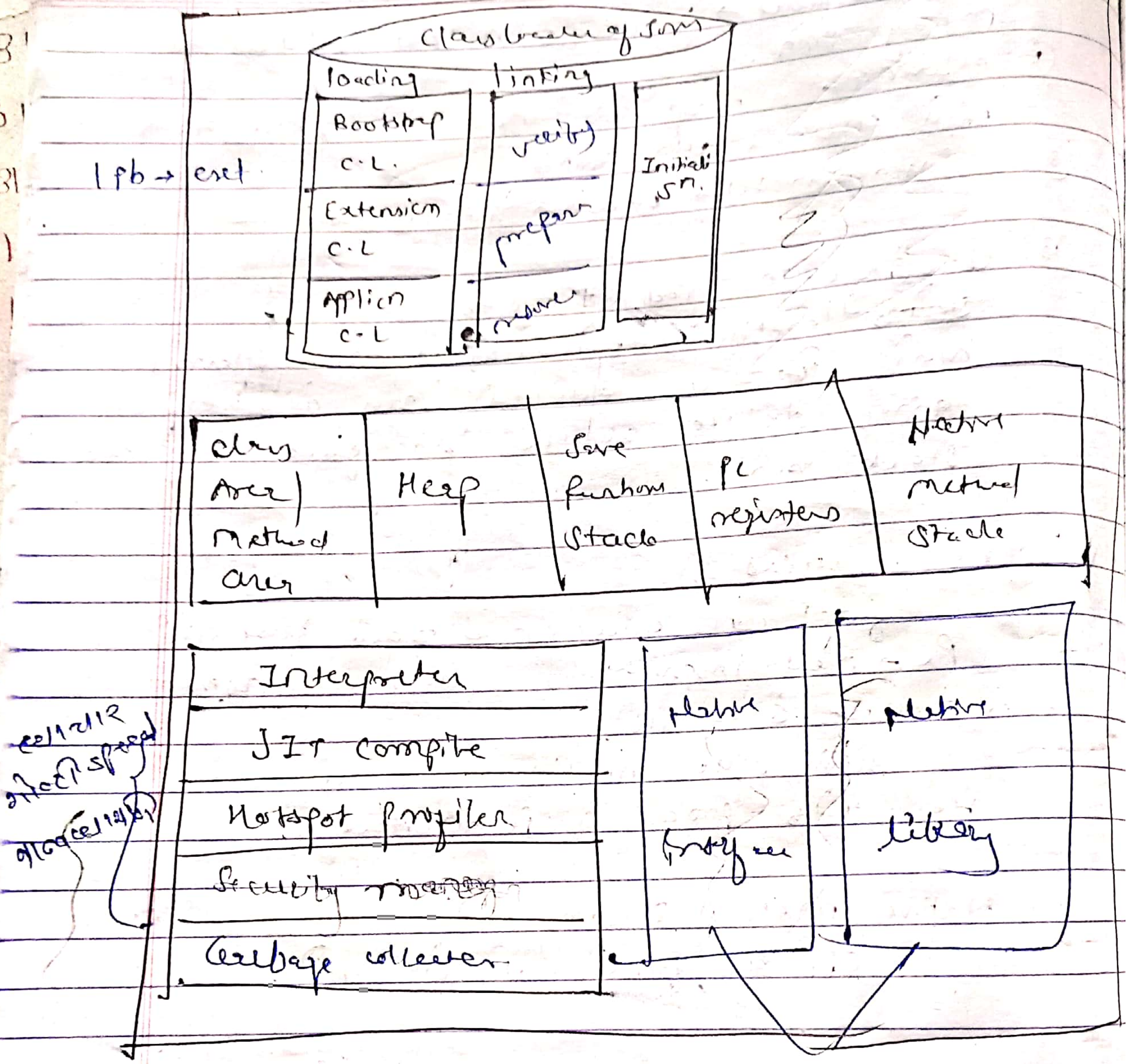
આ method માં object રચાય ત્યારે તેનું address હોય તો object heap માંથી delete કરવાનું કામ garbage method stack કરે છે.

· Pc registers →

funcⁿ ના execution નું track રાખવામાં આવે છે.
PC → next instruction, current instrucⁿ નું tracking રાખવામાં આવે છે.
કારણ આપણે multithreading use કરીએ છીએ. દરેક process ને pc registers.

Native method stack.

C, CPP ને funcⁿ run કરવામાં આવે છે.



JIT → Just in time   C|C++ છે.

Interpreter → line by line code execute કરનાર (speed ઓછી હોય છે)

Compiler → એક જ task રહેલ. તેને કરવામાં આવેલ વાર કરવા તો interpreter ના બદલે પાછળ compiler કરશે.

Hotspot profiler →

Client — ઓછી resources જોઈએ, simple config^n

Server — વધારે resources જોઈએ, મોટું config^n

ડાર server ની setup કેવો। તર તો setup
તૈયાર બનાવવાં થોડા ચેક લાગતો
cmd પર ની બેવડાં નાબેરનાળા તેવટ્યાલગોરી
start / setup કર્યા દેવામના

by default client ને થેઈ ડાર server કરવાનાં ઝાથોમ તર switch આપે
java -server Demo ↵ ← થઈ ચેક થેરો પણ server માટે run કરતો.

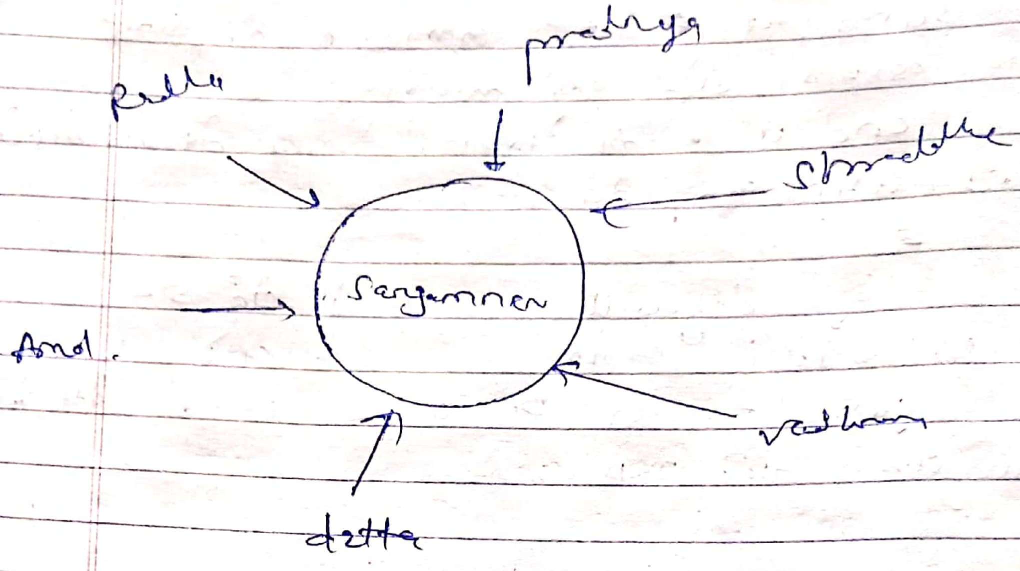
Security manager →
Multithreading → synchronised keyword ડી ની security લાગાનેલ। કે જો keyword security માટે વાપરેલ તો environment તૈયાર કરવું program execution માટે
application ની security provide કરવામાં આવી.
~~[illegible]~~
પ્રોગ્રામ ચલાવ code વાળ્યો free છે કે નહીં કે verify (byte code verifier કરતો)

multiple threads કામ કરતાં ની security provide કરવાતો ની security mgr કરતો.

Garbage collector
~~Garbage~~ collector — (લાવે છે garbage method stack)
stack and heap area માંથી નિકાળ્યો
object ને delete કરવામાં આ prg છેતો.
તો મેમરીમાં કામ Garbage collector કરતો.
dynamically allocated object ને

reference count → એક laptop ત્રણ લોકો વાપરતા
તર તેના reference count = 3

म्हणजे resource वापरणाऱ्या रूप एक तर त्यांना त्या resource चा reference count वाढतात



Pradnya ने तिच्या कुणी Sangamner102 कोणं नर त्याच्या केलेली memory सोडणार

Pradnya ⟶ Sangamner102

∴ dynamically object create केला असेल तर तो object फक्त reference variable वापरत अस नाहीये

```
demo d = new demo();
```

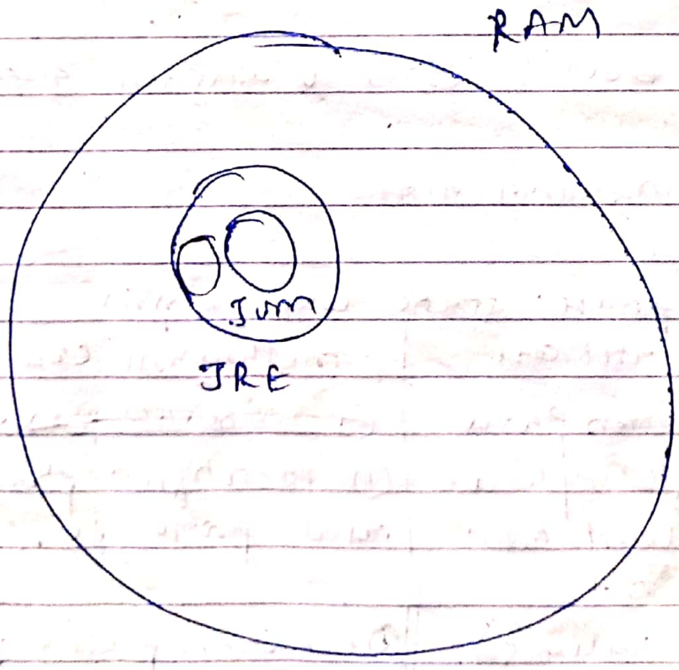
* garbage collector वापरतात जेव्हा object च्या reference count असे. तसं तो delete करणार त्याला stages आहेत पहिले टप्पा नंतर दुसरा मग reference count कमी (0) झाल्यावर ~~delete~~ direct destroy नाही करत. त्यानंतर पण stages आहेत. (नंतर बघू)

* Heap memory ची वाट लावतो garbage collector.

* Native method interface
Java ની communication C++ / C સાથે
કરવામાં xml files આધાર હોતા
data communication કરવામાં ની
∴ interface.

* Native method library.
C/C++ ના code ધરાવતી
libraries

resolve (Symbolic link)
O.S ના
JVM ના ← virtual ← MM ← RAM
address   address
[illegible]
મેળ. mapping



JRE પહેલા
JVM prg load કરે. ∴ દરેક prg નો અલગ અલગ
[illegible] ની જગ્યા RAM ફાળવવા માટે.

Static & non static

member.

① Static variable | field
② static method
③ static block → loading વખતે ડાયરેક્ટ execute હોશે.

* Static field

* Static local ચાલતા નથી
⇒ static variable class loader load કરતો
∴ તો પહેલા load કરતો પણ પણ ઉપયોગ નથી.

* **Static field**
દરેક ઓબ્જેક્ટ object માટે ફાળવવા
field નો same જ value આપે છે આપીએ
ને common આપે આપી તો object બનાવવાનું
આપીએ એટલે આપે તો ત્યારે static
કરાવવાનું field માં.
eg → Avcol

* Static block :- class loading વખતે ડાયરેક્ટ execute
થશે.
main વગર પણ ચાલે.

* બંને phase static load કરવામાં

field વાળો case	method વાળો case માટે
① ~~initialization~~ phase	~~① initialization phase~~
① identification phase var. નો load કરવો memory ફાળવવી default value હોય.	① identification phase method prototype ઓળખાણ ઉપરથી
② Execution phase → user defined value assign કરવી.	② execution phase → function નો call થયા પછી ત્યારે prototype ~~માળખું~~ પ્રમાણે તેનું execution કરવું.

①

Static method

→ method નો call કરવામાટે object નહીં પણ call by classname.

object બનાવ્યા વગર જ static function નો call થઈ શકે non-static field ના ઉપયોગ કર્યા વગર જ વિનાકારણ object બનાવી દેવાથી memory wastage થયા છે. Static કરો methods.

∴ static માટે ∴ object ની જરૂર નથી.

* Static block.

```
static
{

}
```

static block ક્યારે આવે

① static field ની value સેટ કરે છે.

② Static block માં native function નો load ~~static block આવે~~ છે ~~...~~ એક્ઝિક્યુટ થતો code થાય છે.

આપોઆપ execute થવા માટે code જે main method પહેલા આવે.

ત્યાં load / execute native method stack પર થાય છે.

* Prog

Q. → static block માં આપણે static function નો call કરી શકીએ છીએ? ~~...~~

→ હા.

class નામ દ્વારા call કરો.

execution માં તે method stack પર જાય છે.

તે method static block માં static block ના સ્ટેપ માં class ના ... direct પણ થાય છે.

Encapsulation

(methods)
Binding of fields & member functions

object orientation चे ~~...~~ स्तंभ.

1. object
2. class
3. encapsulation, ~~abstraction~~
4. abstraction
5. inheritance
6. polymorphism

Bank

```
{
private float balance;
    {

    }
}.
```

clerk

```
{

}.
```

clerk ने balance पाठवला त्यात valid changes करावे पण ते valid आहे की नाही हे bank नीट ठरवायचं.

Bank मधला balance clerk ला पाठवला आहे पाहिजे. तो getter method ने

* abstraction ही पण एक paradigm आहे O.O चा.

data abstraction	method abstraction
field किंवा member hide करणे.	– implementation hide करणे C++ → pure virtual function java → abstract function आणि interface ने implements करणार आपण method abstraction

Scanned with CamScanner

* final Keyword *

C/C++ માં હતો const.

const int a = 10;
a = 20 reassign કરવા દેતા નથી.

final Keyword →

1. final field (datamember)
2. final local variable
3. final method (member function)
4. final class.

} as a constant અને immutable data types બનાવે.

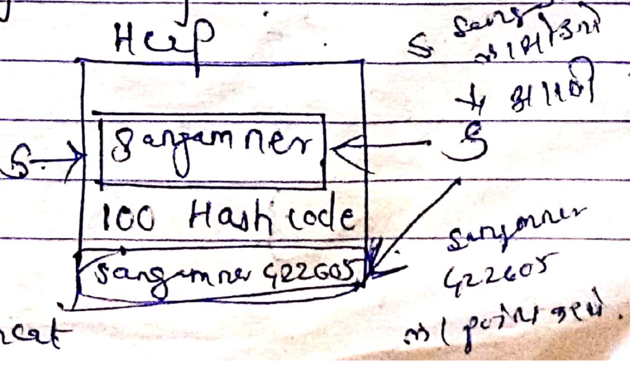
- non final method ની અંદર final local variable હોઈ શકતા.

1. final field.

```
class Demo
{
final int a;
p. s. v. m (String args[])
{
Demo d = new Demo();
}
```

reference variable એ ~~સાચા class ના બનાવેલ object નું address~~ હોવાલાયક ~~કિંતુ સામે parent class ના~~ બનાવેલ.

String S = new String("Sangamner");
Stack પર reference variable S = 100
new થી dynamically memory allocate થશે. Heap પર



S.concat("422605") S →

String is immutable
∴ ફરીથી allocate કરીને concat

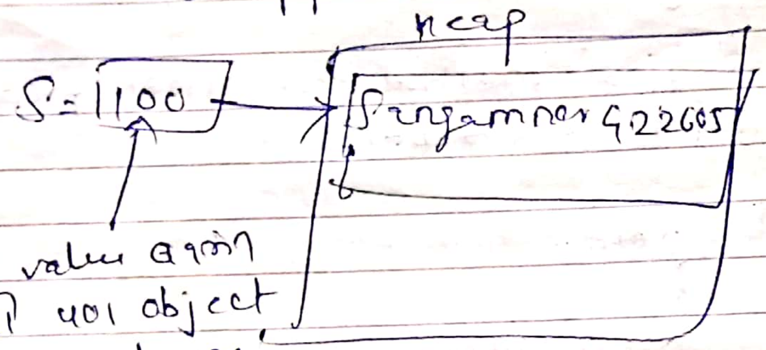
immutable → Cannot change

immutable data types
(1) primitive
(2) String नावाचा class } यांसाठी final.

StringBuffer → Mutable class.

mutable → object state म्हणजे बदल करतो.

```
StringBuffer S = new StringBuffer("Sangamner");
S.append("422605")
```



ही value बदलली नाही पण object inplace change होणार आहे.

पण immutable म्हणजे S direct आधीची जागा सोडून नवीन जागा पकडतो.

① प्रथम initialise केल्यानंतर बदलणार नाही.

```
class Demo
{
    final int a = 10;
    p.s.v.m (String args[])
    {
    }
}
```

✓ पण इथे initialise करायचे नाही. प्रत्येक object साठी वेगळी value असणार

e.g. PAN number पण इथे constructor मध्ये initialise करावे.

* constructor ના call સમયમાં કી ત્યાંથી ~~default~~ value લેવા ગોળ નાહી.
Constructor સંપત્તાવર ત્યાંથી default value લેવે.

final variables cannot reassign
then initialise

(2) Constructor વ્યા આગ.

```
class Demo
{
final int a;

  Demo()
  {
   a=10;
  }

}
```

(3) Non static block વ્યા આગ | instance આગ.
→ શા block constructer વ્યા આવી
આગળી object creation વ્યા નાર
થઈશ્યા મરશે.

```
class Demo
{
 final int a;

  {
   a=10;
  }

}
```

```
static
{
Demo.collegename="AVCOB";
```

```
class Demo
{
static final int a;
```

```
static
{
a=10
}
```

static block પહેલાં આવીને execute થશે પણ static ન
static વાળા આગ ક્યાં static block વાળા
લેવાનો. મળી ગઈ static
આગ value વલીચારી ... ને નો
variable static પાડેલી compulsory
e.g collegename final+static કરવું કારણ
ને શાળાનાં same + કોલેજનાં વડળું ન થવું ન

field માં કર્યા વાર પણ value (કિંમત) નથી
તો error થાય. Constructor માં કર કિંમત નથી તો
default value final થાય.

② final local variable
method વાળા માં

class Demo.
{

void fun()
{
final int a; આમાં default value નથી
જાણી blank final
variable
આમાં error થાય નહીં.
}

}

blank final var. માં
value તો કર થાય પણ ફક્ત વાર.
reassign કરવા થાય નહીં.
value ન થાય ત્યાં વાપરવામાં પ્રયત્ન કરો
તો error થાય.

③ final method.
class Demo.
{
final void fun()
{
}
void fun()
{
}
}

Scanned with CamScanner

```
class test extends Demo
{
void fun()
  {
  }

void fun()
  {
  }
}
```

error 6201 final methods cannot overrided.

body need changes also

4) final class → cannot be inherited.
   inheritance prevention.

Method overloading

Same scope
Single name multiple behaviour

parameters

Same operation — different function

prototype

Compile time polymorphism
early binding

~~late bind~~

dynamic binding

parent ની reference
માં object child ની

operator overloading → છે implicitly
ઓટોમેટિક.

String હોય તો + concat

+ addⁿ

ફક્ત string print હોય.

↓ + + વડે concat

println(   ) હોય.

9/12/2022 scope માટે overloading

① same class

② inheritance વખતે → child ક્લાસમાં

ત્યારે ① type of para   child + parent માં

② No. of para

③ sequence of para.

C++ માટે global function માં

overload કરી શકાય.

~~જ્યારે parent ક્લાસ માં static~~

~~function child ક્લાસ માં આવે~~

~~static function નથી આવતા~~

21/12/22

ફક્ત { ① type } method overriding

② no.

~~③ sequence~~

method overloading વાળા case

માટે modifiers ના કોઈ રીતે

ભાગ ભજવતો

method overriding

Access specifier वाला parent आणि child चे दोन्ही same असावे. पण child चा access modifier मोठा असावा.

```
class demo
{

  void add (float f)
  {

  }
void add (double d)
{

}
p.s.v.m(-)
{
  demo d = new demo();
  d.add (10);
}
```

int

```
fun(value);
fun(type value);
fun(a+b);
fun(fun());
```

↳ non void.

① exact match

② widening type cast convert

widening

byte (10)

fun(int)

③ autoboxing

④ autoboxing widening

int a = 10;

a. [10]

100 104

⑤ var arg

⑥ var arg widening

Integer a = 10; implicitly

new Integer(10);

a [100] → (10)

obj

a. method();

Wrapper classes

```
class demo
{
 void fun(float x)
{

 }
void fun(Integer x)
{

 }

 main
Demo d = new Demo();
   d.fun(10);
   float x=10;   widening
   Integer x=10;   Autoboxing
```

આપણે widening થાય ત્યાર
પછી ત્યાર autoboxing થાય.

exact autoboxing ત્યાર પછી
autoboxing widening ક્રમમાં.

---

overloading માટે (1) reference variable
જરૂરી નથી (2) parameters

Q2. (1) pass

(2) call કરવામાં reference વાર

Arrays

Demo d = new Demo();

d ત્યા શાન પ્રકાર value.

પ્રકા. તેની પ્રકાર ની chategory ના

value હેઠળ આવશે.

int a[10]; c/c++ defi

કાયમી memory રોકશે.

* memory ફિક્સ લાગતે કે આગળ runtime નક્કી કરશે.

~~data~~ variable

primitive	reference
int	Integer
float	Object
	~~Float~~
	Array

int a[10] પણ ચાલશે નહી પણ

ક્યાં? →

type આપી value ની
↓ ખબર પડવાની છે.

* 1) declaration stm — type આપી value ના આપવી જરૂરી

* definition stm
↓
object → runtime → new

optional

Syntax

<access modifier> <non-access modifier>
datatype variable-name
[ ];

→ declarat^n

int a[ ];
int array — memory મળતી નથી, ફક્ત type આપી

ત્યાર પછી memory માં જાય.

Java માં ~~address~~ ની pointer
નો concept નથી.

int a, [ ] b ; error

declar int[] a;
int a[];
int []a;

1 D array

int a[][];
int [] a[];
int [][] a;
int[] []a;
int []a[];
int [][]a;

2 D array

int [] a, b; → a[], b[].

int []a, b; → a[], b[]

↑ અહીં ડીક્લેર નો int. થી લેવામાં આવે.

int a[], b → a[], b is Integer var

int a, ([]b); a-variable | Compile error (error)

int a, b[]; → a int var | b array

~~comma separated આપવામાં છે~~

જો subscript અહીં આપવામાં આવેલ હોય તો

પહેલા var ની જ અહીં લાગુ થશે

બાકીનામાં data type આપવાનો નહીં

* decln વખતે size લખાતી નથી size run time આપવામાં આવી.

int a[10]; ✗

defin → value initialisation
→ using new keyword (explicite)

1) decln + defin + initionlisation

int a[] = {10, 20, 30, 40};

implicitely new keyword

a માટે object બનાવે છે.

a ની reference

2) int a[]; a [ ]

a[0] = 10 ✗ memory માં ફાળવેલી નથી

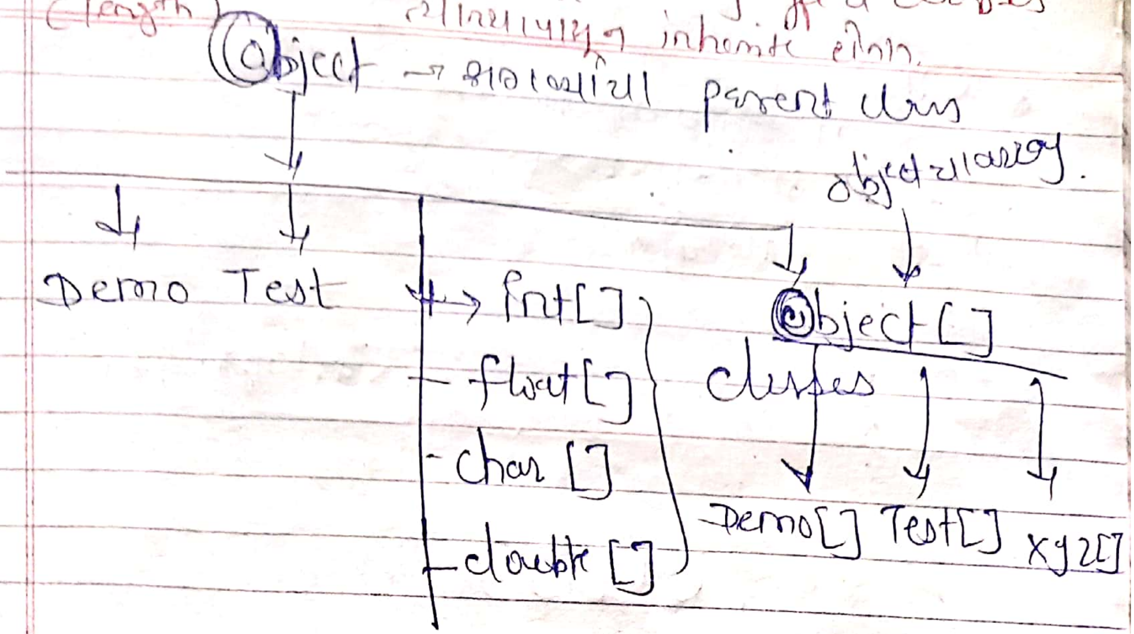
a → reference store કરવા માટે છે.

a = new int[10];

Array એ reference type છે

object array type છે

object એ parent class છે પણ array ને func. વગેરે તેમાં
object[] class ના માન છે બધું array type ને classes
(length) વારસામાંથી inherit થશે.



10, 20, 30 primitive

Integer → wrapper classes.
primitive data object બનાવે.

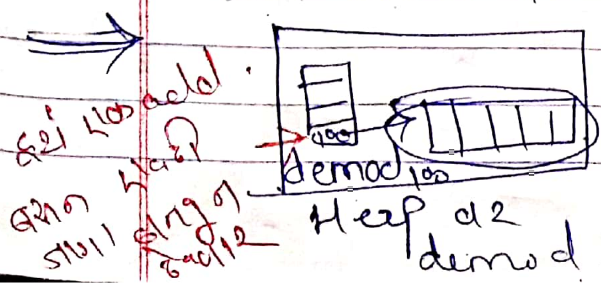
Ex. int[] → class.
primitive data primitive બનાવવા
ઉપયોગમાં.

int[] માંથી કોઈનું વારસો આપણા કોઈનું
વારસો નહિ.

class de
{

int a

Q. array માં ano a field ક્યાં આવે છે?
આપણે જોયું નથી object બનાવે કોઈના?



બધું object આપે reference
ના ૨ જ છે.

* (ભરવો)

n સભ્યોની ગણતરી

$$n = \frac{\text{સાઈઝ of array}}{\text{ઓબ્જેક્ટ સાઈઝ.}}$$

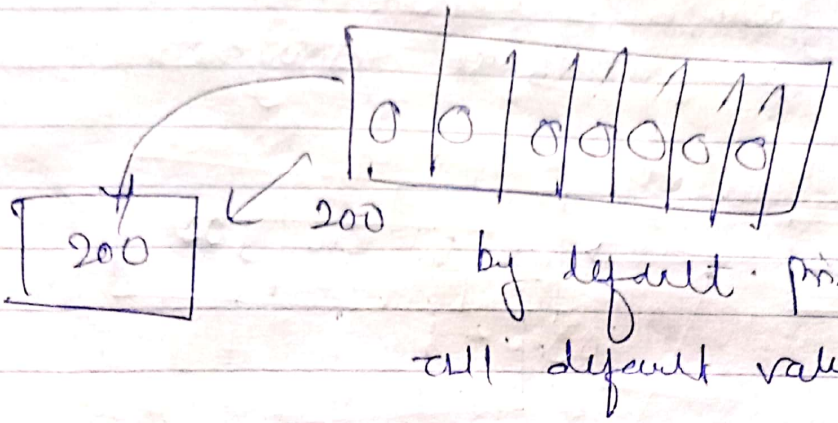
```
for (int i=0; i<a.length; i++)
{
S.O.P(a[i]);
}
```

* દરેક array માટે length → nonstatic, → final

ડીક્લેર implicitely.

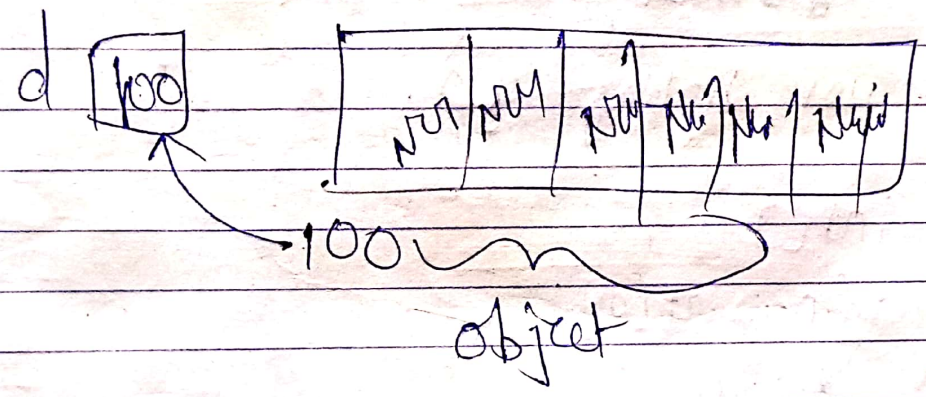
ટી variable object[] array ની

int [] a = new int[10];

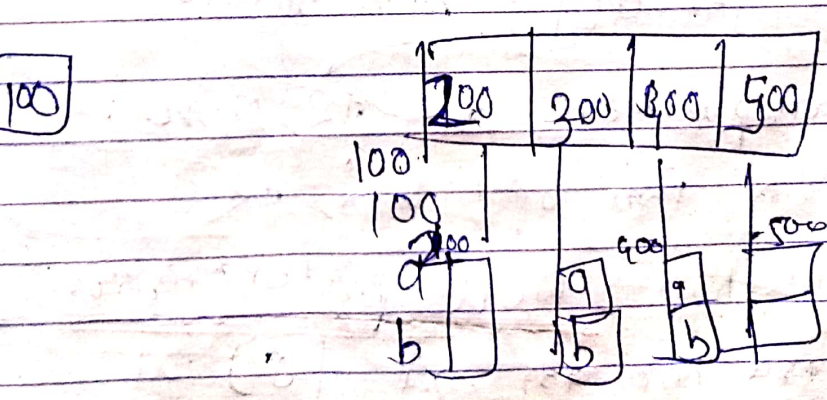


by default primitive all default value

Demo [] d = new Demo[10];



Demo [] d = new Demo, new Demo, new Demo, new Demo, new Demo



var - arg.

fun(10); 1'લાંબા parameter

fun(10,20)

જોકે પણ તેવું func def.

method values આપવા multiple

પણ લખવાની array બની જ જશે

int a[] = {10, 20, 30};

fun(int[] a) ← fun(a);

var-arg →

```
void fun(int... a)
{
    a.length
    for loop થી a પકડવું
}

fun();
fun(10)
fun(10, 20).
```

fun(new int[]{10,20,30})

પહેલાં આ રીતે

program

* ~~Multidimensional array~~

abstraction

Concrete Class →

abstract class

Concrete method

abstract method

Interface

abstraction

data abstraction using private keyword Access specifier	method abstraction (implementation abstraction) method માં private કોઈ નર વાપરવા ઉપયોગમાં નહિ.

Interface

```
interface Demo.
{
    abstract method.
    abstract void fun();
    abstract void gun();
}
```

Java 8 मध्ये पुढे static आणि default method interface मध्ये आत implement ~~करायला सुरू~~ केले जाऊ शकतात.

static आणि default Keyword वापरून.

```
interface Demo
{  void fun();

}
```

हे चालेल. ~~1201~~

आपल्याला आत फक्त abstract method

✓ class ← extends class

✓ class ← implements Interface

✓ I ← extends Interface

Int ← extends ~~class~~

I ← ~~Imp~~ C

C ← ~~implements~~ C

Java.lang.package

default parent class / default package → class files < interface, class

① Object, Class → root class

② String, StringBuffer, StringBuilder

③ Wrapper classes

④ Throwable, Exception, Error

⑤ System, Runtime, Process, ClassLoader, SecurityManager

⑥ Thread, ThreadGroup → multithreading

* Class → કેટલા આપણાં class method call ન થાય કેટલા class ભરાયા class વાપરીને તેની object બનાવ

JVM વડે

java.lang.Object → class
↓
default parent class of every user defined & predefined class

① reusability માટે
② loose coupling કરવા માટે
parent ના reference કરાવવા
function માં પાસ કરવાના direct parent ના reference રહેશે.

loose coupling

parent ના reference સાથે object બનાવવાથી વાળા બધા class ના

① public native int hashcode();

② public boolean equals(Object obj);

③ public String toString();

④ public final native class getClass();
Class getClass();

⑤ protected native Object clone() throws cloneNotSupportedException

6) protected void finalize() throws Throwable;

7) wait()

8) wait(-);

9) wait(-, -);

10) notify();

11) notifyAll();

Clone → ಒಂದು object ನ copy
ಇದರ ಯಾವ changes ಹಿಂದಿನ ನ ಇರಲ್ಲ.
original ನ compare ಆಗುವಾಗ etc etc
shallow copy
deep copy

* public String toString();

⇒

class Object
{
public String toString()
{
return
}
}

S.op (d.tostring())

S.o.p(d);

→ classname @ hashcode in hexadecimal format

आपल्याला class चे member print करायचे असतील पाहिजे.

मग override करा. असं ↓

```
public String toString()
{
return this.x+" "+this.y;
}
```

String

ते data member print करायचेत

* public native int hashcode();

→ आपल्याला object चा hashcode बदलायचा असेल तर

∴ ही override.

hashcode() → virtual address

C/C++ आधारून केलेला

∴ ही method native आहे.

return तो address → int.