source code — which language feature che
architecture } independent java
platform — OS
C/C++ on dependent language che.

features

Java is a compile and interpreted programming language

.net - virtual machine - CLR - common language runtime

python - PVM

java-bytecode - .class file

History of Java

vice president - mike green

Sun microsystem

James Gosling
Petre
mike

green team
green talk
oak technology

1991 — start to write a java — (public 1995)

WORA - write once run anywhere

~~vice president - mik~~

java માં શું શું

Dynamic
revolutionary
java

રાંની જાતે java જ્યારે

1991 → start
1995 → beta version
1996 — java 1.0
1997 — java 2.0
1998 — java 3.0
2000 — java 4.0
2002 — JDK - 5
2004 — JDK - 6
2006 — JDK - 7
~~2008~~ — JDK - 8

2008 oracle take over the full company of Sun microsystem

2008 - jDK-8

14 — jDK-14

17 — 22 march 2022

18

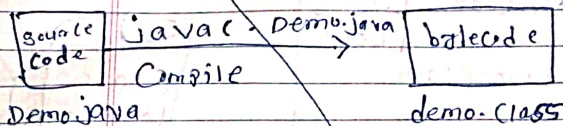
J2SE - standard edition

J2EE - enterprise edition

J2ME - micro edition
(mobile)

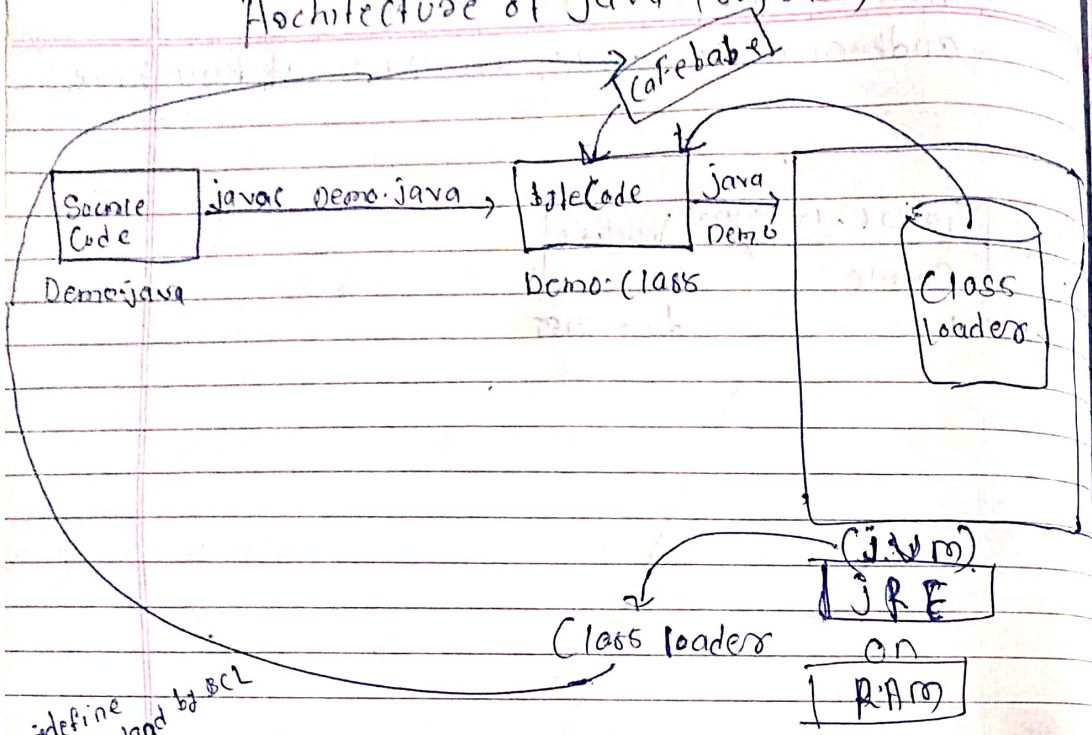
Architecture of Java Program

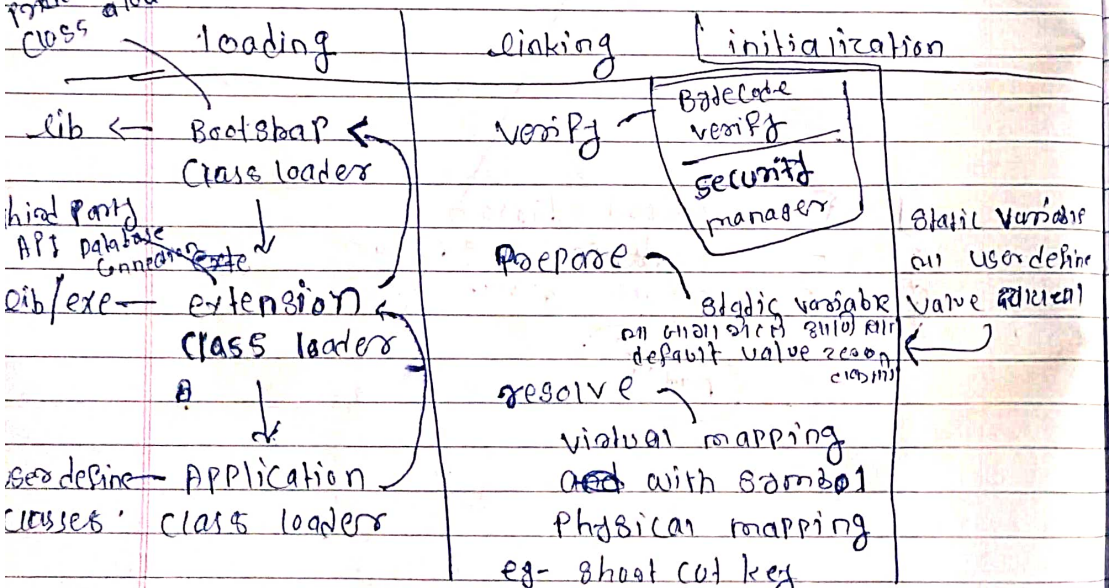
How to Compile java program and how allocate space in RAM at Run time



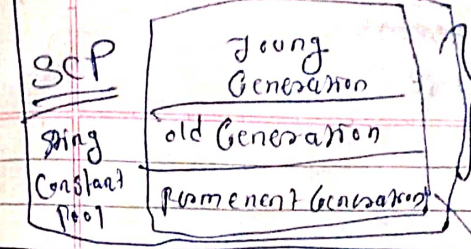
J2SE Standard edition
J2EE enterprise edition
J2ME micro edition
mobile

Architecture of java Program

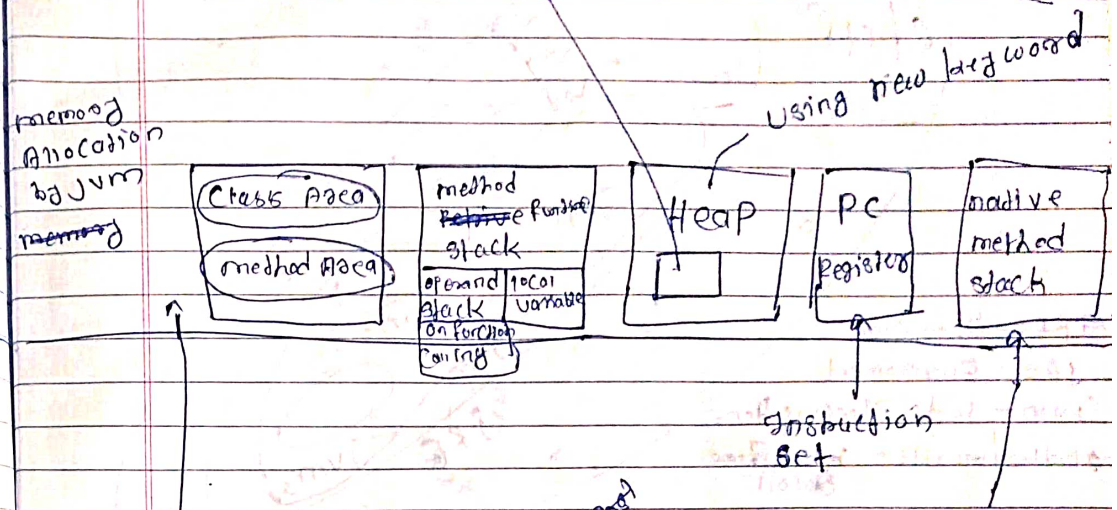


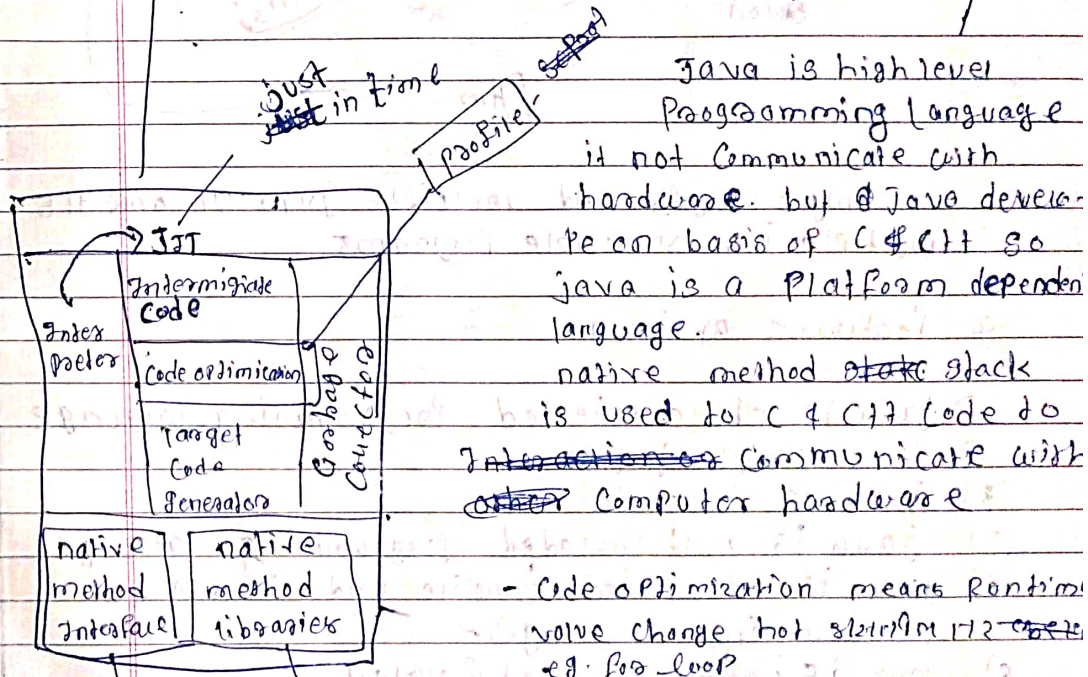


Delegation Hirarchy Algorithm (JVM)



0.06 milisecond





Java is high level Programming language it not Communicate with hardware. but Java develop Pe on basis of C & C++ so java is a Platform dependent language.

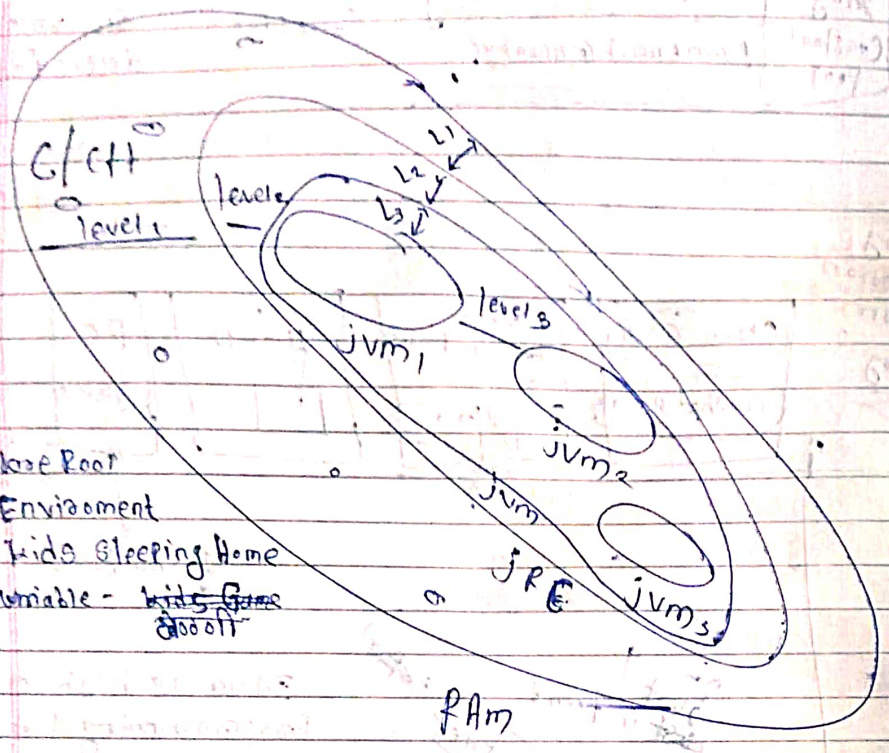
native method stack is used to C & C++ code to Communicate with Computer hardware

- Code optimization means Runtime value change hot spot in 1/2 eg. for loop

native method Interface is used to Connection between C/C++ and java

- Profile - C/C++ and OS Code seprate then

JAE - ~~java~~ ~~Archi~~ ~~Doctrine~~
java Arcaive

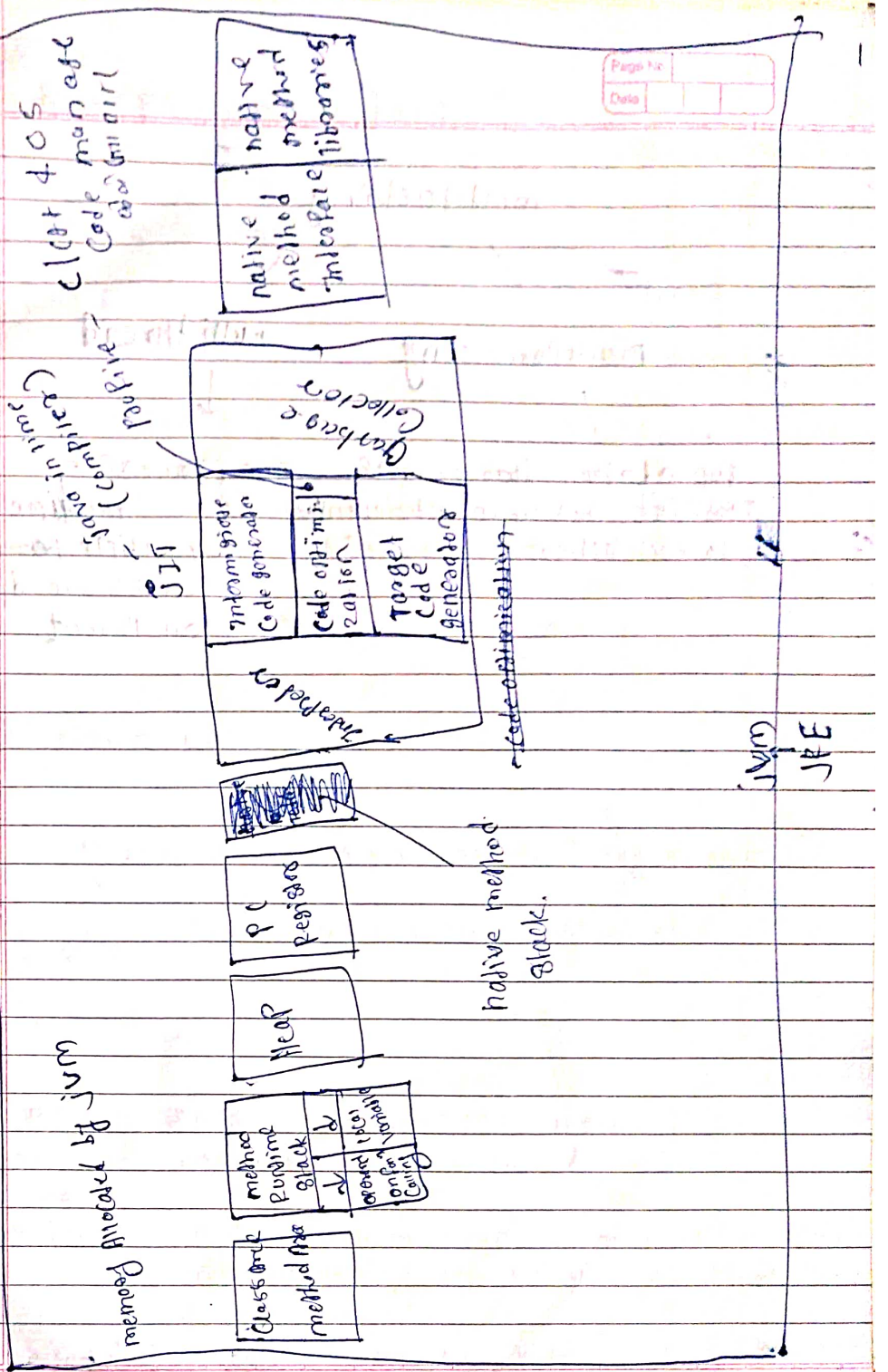


JRAM - Store Room
JAE - Enviroment
jvm - Kids Sleeping Home
method/variable - ~~kids Game~~ Toys

only one JRE and multiple jvm in one JRE.
multiple jvm → multiple programs

* features of java

1) java is object oriented Programming language
2) it is simple to use
3) java is dynamic
4) java is multithreaded programming language
5) java is ~~interpreted~~ compile and interpreted Programming language.
6) java is ~~robust~~ robust (strong)
7) java is high level Programming language
8) java is Platform ~~in~~dependent but java class file is platform independent
9) secure Programming language readable



multitasking

multiprocessing — Run atatime two or more process but it is not dependent to each other

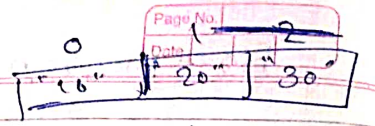
multi thread — one process) multiple process multiple process are dependent on parent

first program in java

```
class Demo
{
    int data;        — Data member (field)
    getdata();       — member function (method)
};  - optional in java
    - compulsory in C++
```

```
class Demo
{
    Public static void main (String args[])
    {
        System.out.println("Hello word!");
    }
}
```

main - entry point function name
void - not return any value to OS
Public - all data member and member function are accessible by outside of this class to any where.
static - call to method without creating object



(1) (a)

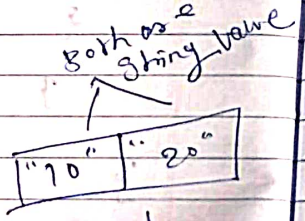
```
class Demo
{
  public static void main(String args[])
  {
    ~~printf("~~
    ~~System.out.println(String args[]);~~

    int a = args[0];
    int b = args[1];
    int c = ~~args~~ args[2];

    System.out.println(a+b+c);

  }
}
```

javac Demo.java
java Demo 10 20 30

output
10 20 30

(2)

```
class Demo
{
  Public static void main(String args[])
  {
    int a = ~~Para~~ Integer.ParseInt(args[0]);
    int b = Integer.ParseInt(args[1]);
  }
}
```



(10) "10"
(20) "20"

javac Demo.java
java Demo 10 20

output
30

04/04/2022

(4) દરેક ઉમેદવાર value string આવશે ... always remember
(5) no of value ...
(6) sequence ...
(7) ... program ...

* **Draw Backs of Command Line & argument**

2nd way — 0) no of Parameter, sequence of Parameter, type of Parameter

1) CL value આપવા cmd વાપરી દાખલ થી String રાખવી

2) arguments and parameter count are always same નહીં તો exception array out of bound.

3) value accept કરવાના sequence and value દાખલવાના sequence same હોવી જોઈએ નહીંતર exception આવે.

(4) cmd થી data દેવાવ number Format. Program run ... data ...

1st way →

(1) Program value લખવાથી તો Hard coded value હોય છે ... change કરવાના ... Program ... Run time value ...

3rd way — using BufferReader.

import java.io.*;

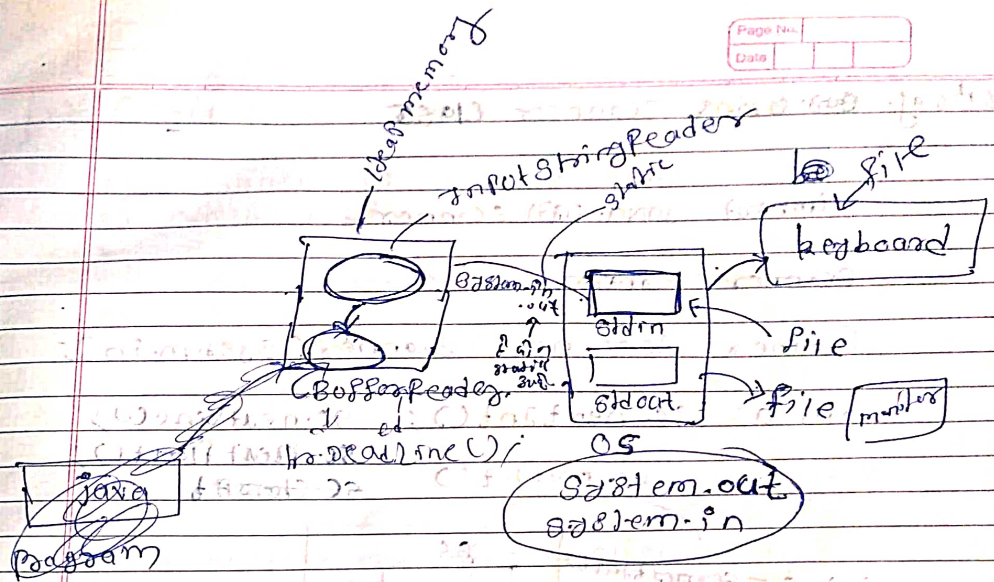
InputStreamReader is = new InputStreamReader(System.in);

BufferReader br = new BufferReader(is);

int a = br.readLine();

InputStreamReader — accept the one by one character

directly accept at time a line fill in



Draw Back

1) પ્રત્યેક value is String જ હોય

Point(1) ... program value ...

- Pascal Case -
- Camel Case -

Pascal case — Pascal Case Java ... class ...

eg —

class GaneshChandrabhanShelke

Camel case :— initial letter of word is small

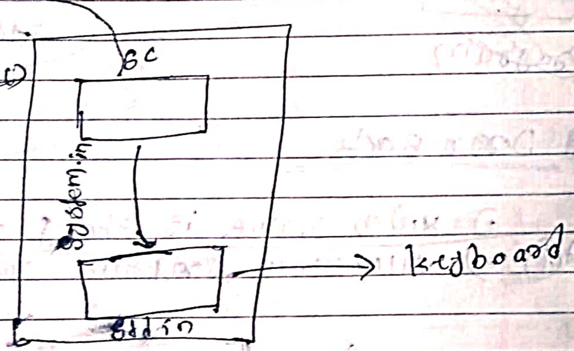
a naming convention java function ... use ...

4th way. using Scanner Class

- import java.util.Scanner;

~~Scanner Scanner~~

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

int a = sc.nextInt();      sc.nextLine();
                           sc.nextFloat();
                           ~~sc.charAt~~
        sc.nextInt()
int a = ~~sc.nextInt()~~
```



String java માં primitive નથી.

કારણ string ની size fix નથી હોતી
એ ની primitive નથી.

```
String s = sc.next();
String s = sc.nextLine();
```

} String ના function use કરવા string ને વાપરો using Scanner class reference variable.

✳ Constructor

1] Constructor is user defined method
2] Constructor and class name always same

```
Class Demo
{
    int a;
    Demo()
    {
        a = 0;
    }
}
```

3] Constructor ~~when~~ invoked implicitly (jvm) when class object has created.
4] Constructor not have return type not return anything.
5] Types of constructor

1) Default constructor.
2) Parametrised constructor
3) Parameterless constructor

6] Constructor is ~~SEP~~ special method in class
7] Constructor access modifier is always Public.
8] Constructor is not static
9] Constructor can not be ~~overloading and~~ overriding
10] Constructor can not be constant

Static	non static
Field (Data member)	non static field
methods (member function)	non static method
Block	Block
Inner class	inner class

cylinder volume = $V = \pi r^2 h$
Area of circle $A = \pi r^2$

Static — float Pi = 3.14; Common memory to all object.

① Static field (Data member / property)

Static field એ class માટે [illegible]. Static નો use [illegible]. [illegible] field static બને.

જો આપણે field static keyword વાપરતા નથી તો by default non static [illegible].

Static field ની memory class load થતાની [illegible].

By default ત્યાં field માં zero (0) [illegible]. [illegible] madhi [illegible]. જો [illegible] user define value [illegible] Initialization class loader [illegible].

Static field [illegible] class ના [illegible] access [illegible] object [illegible].

② non-static field

non-static field ખાલી object [illegible].

જો [illegible] use [illegible] object [illegible].

object [illegible] non-static field ના [illegible].

Static method madhi non-static field [illegible] static [illegible] load [illegible] object ના reference [illegible].

② Static method

જો [illegible] non-static [illegible] method [illegible] static [illegible].

ત્યાં static method ના [illegible] class load [illegible].

Class area

Static area

ત્યાં method ના call explicity [illegible]. [illegible] call [illegible] class [illegible] object [illegible].

Static method [illegible] non-static ના call [illegible] class ના object ત્યાં class madhi [illegible] using object.

Static method madhi [illegible] non-static field [illegible].

Static method madhi [illegible] static field [illegible].

~~Static method~~

② non-static method

non-static method ના [illegible] object [illegible] using class object ત્યાં non-static method ના call [illegible].

non-static method ના આધારે static method madhun call કરવા ને ઉપયોગ કરી શકાતી.
non static method માં સહેલી ~~static~~ static field ને સાથે non-static field દેખી પણ લખાતી.

non-static method ના static method madhun call કરવા માટે static method madhi class ના object તૈયાર કરવા call કરાવવા પણ static method ના call કરવા જોઈએ.

③ ~~Block~~

③ static Block

static keyword use કરવા આપણે static Block લખી શકાતી.

static
{
=
}

static Block class load કરતાં આપમેળે ~~થશે~~ execute થાય. તેનાથી call કરવાની જરૂર નથી.
static Block માંથી non-static method call ~~કરાવવી શકાતી~~ કરાવવા જોઈએ તો તે class ના object તૈયાર કરી call કરાવવી ~~જોઈએ~~ જોઈએ.
static Block માંથી static method ના આધાર call કરી શકાતી object તૈયાર ન કરતા.

③ non-static Block

non-static Block call કરવા માટે object તૈયાર કરવા જોઈએ.
object તૈયાર કરીએ ત્યારે non-static Block call થાય. ત્યારે જ Constructor ના call થાય.

non-static block દરેક constructor માં ~~ની દરેકમાં block of code મળે~~ ની multiple constructor માં Block ના code use કરી શકાય.
non-static Block madhi આપણે non-static field & static field use કરી શકાતી. ને static method ના call કરી શકાતી. non-static method ના ~~static~~ non-static Block માટે object તૈયાર કરવા જોઈએ.

①

* this keyword

this keyword છે object ના Reference પાછા મેળવા રાખે છે. જ્યારે આપણે તેને implicitly વાપરતા નથી.
ક્યાં વાપરી constructor ; method.

```
class Demo
{
    int a;
    int b;                        implicitly used this Demo this

    void setData(int a, int b)
    {
        this.a = a;
        this.b = b;
    }

Public static void main(String args[]);
{
    Demo d = new Demo();
    d.setData(10,20);
              ↑
             a,b
}
```

②

* final keyword

field method local variable class.	} both static and non-static

1) field

final field છે આપણા class ના આખા ના function ઉપર કરશે.
તેની value પહેલા set કરતા હોય તો Reconside કરતા શકતા નથી.

eg.

```
class Demo
{
    final int a;
    psvm (String args[])
    {
    }
```

તે value ક્યાંથી set કરી શકાય.

1. Hardcoded value દેવામાં આવે.
2. constructor ની પણ કરી શકાય.
3. non-static block ની પણ કરી શકાય.
4. જો તે field static હોય તો આપણે તે static block માં પણ set કરી શકાય.

(3)

② ~~final variable~~ final local variable

Final local variable છે class ત્યાં આવ્યા આવેલા method, function Block, static Block, non-St. Block માંથી આવતા.

Final local variable ના ઉપયોગથી value બદલી શકાતી નથી. જો final local variable Print કરવાના compile time error થશે.

③ final method

આપણે જ્યારે method final કરીએ ત્યારે તે override ~~થઈ શકે~~ થતી નથી.

જ્યારે static final method કરીએ ત્યારે sub class માંથી method hide ~~થઈ શકે~~ થઈ જાય. પણ final keyword થી તે method override કરવા દેતા નથી child class માં.

④ ~~final local variable~~

Final local variable

```
void func()
{
    int a;
    S.o.p(a);   <-- Compile time error
}
```

જો આપણે declare local variable without initialization વાપરવા જઈએ ત્યારે error થશે. કારણ local છે.

int a;
int a=10 / int a=10] non-final local variable
we can change a=20;

Blank variable

final int a;
a=10 / final int a=10;
a=20; – Compiletime error

Blank final variable

local variable ના જો પહેલા આપણે value દેવાનું ભૂલી જઈએ તો ... error થશે.
Java માં variable ના garbage value નથી.

Page No
Date

(4) ~~Final Class~~

* જ્યારે આપણે આપણા class ને બધાને સ્ટાર્ટ final કરવાના હોય ત્યારે object final કરવાનો.

(4) Final Class

~~જે class Inheritance ન થાય~~
જો class Inheritance કરવા દેવાના ન હોય તો ત્યાં class ને final કરવાનો.

eg:- આપણા ઘરમાં ન આવી શકે.

ત્યાં class મહેમાન સ્ટાર્ટ કરવાનો class બહારથી ~~આપણે બહાર વાપરતા હોય નહીં.~~

* ~~Class~~ Java માં const keyword હોવા છતાં તેનો use કરતા હોય નહીં.
as a Identifier તરીકે પણ use કરતા હોય નહીં.

(5) Static Final data member

જ્યારે memory મળશે Initialization will be in static Block.

static - ત્યારે કામ કરશે · ~~એક્ઝીક્યુટ~~ થઈ જશે memory મળશે

final - ત્યારે કામ કરશે ત્યારે memory ના final (constant) કરશે.

22022022

(5)

modifiers

Static	native	private
final	synchronized	public
abstract	transient	protected
volatile	strict fp	default

abstract - Implementation hide કરવા. method, class

volatile - multithreading
એક process ચાલે ત્યારે multiple process ચાલવાના.
બધા variable પર operation કરી શકાય using volatile keyword

Access modifier or visibility specifier

Access specifier

eg. 10 ચાવીઓ
8 - public
2 - private
5 - protected

(1) Private :-

Parent class ના data member private કે જેથી કોઈ બીજા class માં access ના થાય અથવા તો child class માં વાપરી શકાય નહીં

(2) Public :-

આપણા data member કે member function ને ગમે ત્યાં વાપરવા દેવા. eg. બીજા class માં, folder (package) class ની બહાર વાપરવા દેવા. eg. main fn

(6)

22022022

(3) Protected :-

Protected data member and member function આપણા પોતાના class માં, પોતાના child class માં, તે જ package માં વાપરી શકાય.
બીજા package માં પણ તેના parent ના child હોવા જોઈએ.
ફક્ત તે class child હોય તો protected data member & function વાપરવા દેવા માટે.

Note: package માં protected data member વાપરવા માટે class નો object બનાવીને વાપરવા દેવાય.

(4) default :-

આપણા default datamember અને function ને package, child class, પોતાના class માં વાપરી શકાય.
બીજા package માં વાપરવા દેવા નહીં.
જો default ના હોય data member ને default ગણાય class માં.

5) abstract

data member abstract બનાવી શકાય નહીં class અને method abstract બનાવી શકાય.
method abstract હોય તો તેનું implementation / definition ફરજિયાત optional હોતું. તેને hide પણ કરવું.

જ્યાં કેવળ Parameter pass હોય ત્યાં આ કામ ઉપયોગ થાય છે.

abstract class :-

class abstract હોવાથી ત્યારે object create કરવા દેતા નથી.

(6) Volatile :- (અસ્થિર)

volatile એટલે data member અસ્થિર. class, method volatile હોય છે.

volatile હોવાથી multithreading થાય છે. એક process ની under multiple process run થાય છે.

આપણા main variable આપણે multiple process માં હોય આપણે ત્યારે જે operation કરીએ છીએ તે main variable જ change કરવું પડે છે ત્યારે આપણે volatile લખીએ છીએ.

eg- linked list ના first node ની first node address આપીએ.

(7) native :-

java એ high level programming language છે. તેમાં Hardware Interaction કરી શકાય છે.

java Architecture માં આપણે native method Interface, native library જેવી આવે છે. ત્યારે આપણે Hardware Interaction કરી શકીએ.

native એટલે variables અસ્થાયી, ત્યારે આપણે Hardware Interaction કરી શકીએ.

(8) Synchronized :- એક વખતે એક જ process execute થાય.

Synchronized એટલે method અથવા variable આપણે synchronized કરી શકીએ છીએ.

Synchronized method ના lock કરવા માટે થાય છે.

એક વખતે એક જ method execute થાય છે, બીજી execute થતી નથી જ્યાં સુધી પહેલી પૂરી ન થાય.

Asynchronized -

એક સાથે two more Process execute થાય છે. eg- Phone Call ; Refres

(9) transient :-

class માંથી object બનાવી આપણે ત્યારે data આપણે secondary storage માં store કરીએ છીએ ત્યારે આપણે જે data store કરાવવા માંગતા નથી તે માટે Transient keyword લખીએ છીએ.

transient એટલે field અસ્થાયી method નથી.

(10) Strict fp :-

FPU - Floating Processing unit.

~~Floating Point ની~~
floating value ની calculations કરવામાં FPU કહેવાય.
java એ platform independent Programming language છે.
જ્યારે floating value ની ગણતરી કરવાની platform પર ની બદલાય તેથી float value માં strictFP વપરાય.
ની value not change In another platform and Architecture.

Strict FP
Strict floating Point

ASCII -
American Standard Code for Information Interchange.

Person

Properties
String name
int age
double height
String Gender
double weight
String occupation

Behavior
speak()
eat()
walk()
work()
play()
sleep()

Used this class in library most.

Book
String name
String author — Property: author name, author email, author gender, author age
Double price
int quantity

getBookName()
getBookPrice()
getBookQuantity()
getBookAuthorName()
setBookQuantity()
getBookquantity()

Encapsulation

Encapsulation means Binding together Data member and member Function.
Encapsulation માં આપણે Data member ને Private રાખીએ.
હવે Private Data member નો use કરવા માટે two method હોય છે.
1. Setter
2. getter

Object oriented Concepts

* **Class.**

Class is Collection of fields and methods.

~~In Class~~

Class ની existance આવતી નથી. In Real-world class is a virtual view હોય છે.

object create કરીએ ત્યારે physical existance હોય છે. એટલે કે વાપરી શકાય.

Class is similar to mold (સાંચા) for Sculpture

* **Object**

Object એટલે એક physical entity છે. object એટલે કે જેના દ્વારા class physical existance હોય છે.

eg:- Pen

Instance:-

Object કે જેમાં value હોય તેને class through બનાવાય. એટલે કે તેને class નો Instance કહેવાય.

Object is Instance of Class

* **Encapsulation**

Encapsulation means Binding together Data member and member function.

Encapsulation માં દરેક Data member ને Private રાખવો.

આ Private data member ને use કરવા માટે two method હોય છે.

1. Setter
2. getter

eg :-

```
class Student
{
    private int age;

    // setter method
    void setAge (int age)
    {
        // this.age = age;
        // using validation
        if (age >= 0)
        {
            this.age = age;
        }
        else
        {
            System.out.println("Invalid age");
        }
    }

    // getter method
    int getAge ()
    {
        return this.age;
    }
}
```

Example of Encapsulation (Real time example)

* encapsulated school bag.

School bag can keep our books, pens, notebooks.

* class SchoolBag — encapsulated class

```
{
    Private
        String books
        int pens
        String notebooks
    Public:
        void setbooks ();
        String getbooks ();

        void setPens ();
        int getPens ();

        void setNoteBooks ();
        String getNotebooks ();
}
```

Private Datamembers (Field)

* Setters and Getts method.

Page No. Date

suppgender	Char	Not Null	This field is used to store supplier gender

TABLE NAME: ITEM INFO

TABLE DESCRIPTION: This table gives information about Itemsuch as Item No, Name, Item Description, item Stock. Item Selling Price, Item Purchasing Price.etc.

Field name	Type	constraint	Descriptions
itemid	int	primary key	This field is used to store item no
itemname	Char	Not Null	This field is used to store itemname
item des	Char	Not Null	This field is used to store item description
Item sq	Int	Not Null	This field is used to store item stock
Item sp	Int	Not Null	This field is used to store item selling price
Item pp	Int	Not Null	This field is used to store item purchasing price

```
class clerk
{
    Public static void main(String args[])
    {
        Student s = new Student();
        int a;
        s.setAge(16);
    a = s.getAge();
        System.out.println("Student Age" + a);
    }
}
```

Q. Encapsulation એટલે data member ને private રાખવા અને data member ના value દેવા માટે ફક્ત value લેવા using getter and setter method વાપરવા encapsulation કહેવાય.

Q. આપણા Constructor થી value ઈનીશીયલ કર શકાય તો setter અને getter method ની જરૂર શી.

→ Constructor value આપણા one time દેવું શકાય constructor થી પછી value દેવું શકાય નહીં. પછી ની દિવસ ની value બદલવા માટે ફરીથી constructor call નહિ કરવા માટે બદલી કરવા માટે setter method ની જરૂર અને data get કરવા માટે getter method ની જરૂર.

Inheritance (વારસા) વારસદાર

Single class ⟶ multiple Instances

Single class ⟶ multiple classes → multiple Intances

Example:

Human and Student

પ્રત્યેક Human એ student નથી પણ પ્રત્યેક student એ Human આવે.

~~Student એટલે~~

student એ human છે તો ધ્યાનથી ત્યાં human ની property student માં પણ આવવી જોઈએ. student માં human ની property વાપરી શકાય તે માટે આપણે Inheritance વાપરો.

Code Redundancy - (વધારાનો code) એકનો એક code વારે વારે લખવો તેને code ની Redundancy કહેવાય. વધારાનો code વારંવાર લખવો પડે છે.

Code Redundancy ઓછી કરવા માટે આવે.

Scenario -

Two classes Human & Student ધારો. એક Student object બનાવો.

Using Student object દ્વારા Human ના class ના access કરવા હોય તો Human class પાસેથી Student class Inherit કરાવવા જોઈએ.

Ex.

```
class Human
{
    ===
}
class Student extends Human
{
    === } human Property access
    === } Student own Properties.
}
```

using extends keyword to possible Inheritance

Single Class - multiple Instance

8

Single Class → multiple Class → multiple Instance

Class Human
{
=
=
}

class Employee extends Human
{
=
=
}

Class Student extends Human
{
=
=
}

Class Teacher extends Human
{
=
=
}

જો Class Human ~~Hum~~ class પાસેથી Inherit ~~થતા~~ થતા હો તો પ્રત્યેક class ના object બનાવી ~~શકાય~~ શકાય

Single Class - Human

multiple Classes - Student, employee, teacher

multiple Instances - student, employee, teacher
obj obj obj

Three Model of Inheritance

Relationships -

IS-A - Inheritance
HAS-A - Composition
USES-A - Aggregation

IS-A - Inheritance (Relationship)

✓ Employee is a Person
✗ Person is a Employee

IS-A ~~Rel~~ Relationship when to use
એક બીજા ~~class~~ class એકબીજા ઉપર Depend કરતા હોય
~~તેઓ~~ પણ તે એકના type ના મહત્વ ધરાવતા હોય તો
એ પ્રોપર્ટીઝ અને મેથડ Inherit કરી શકાય.

HAS - A - Relationship

✓ employee has a car.

✗ car has a employee.

eg -

class Address

{

=

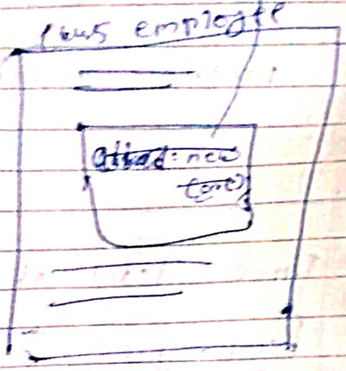
}

class Employee

{

=

}



paddress add ?
· new Address;

બે ક્લાસ એકબીજા પર depend ... same type ના હોય નહિ.

... class ના ... class access ... class ના object બનાવવો.

eg.

Composition (રચના) - ... class ...

... type ના class ...

કેટલા class ના object ... data member (fields) ...

USES - A (Relationship)

✓ Student uses a cycle.

✗ Cycle uses a student

aggregation - ...

Parent class ... child class ... using child class object ... parent class access ...

~~eg - employee~~

eg - employee and car

... employee ના car ...

class employee ... class car ... employee ના car ... employee car class ના object ...

Employee

{

~~void ...()~~

Employee Josh = new Employee();

Josh.start(Josh);

Car

{

void start(Employee josh)

{

this.Josh.run()

}

run

{

S.o.P.("Josh start here");

Coupling (Relationship)

Loose coupling

object

~~Class નો ઉપયોગ~~

Class ના object ને [illegible] [illegible] ના [illegible] Reference variable [illegible].

[illegible] એક જ Reference variable [illegible] object initialize [illegible] [illegible].

~~[illegible]~~

~~[illegible]~~

[illegible] object ના [illegible] initialize [illegible] [illegible] address [illegible] Heap [illegible]

object ના address [illegible] change [illegible]. [illegible] Loose coupling [illegible].

Demo d = new Demo(10);
Demo d = new Demo(20); — Loose Coupling.

Strong Coupling

variable

object [illegible] data member ના memory [illegible] memory Permanent [illegible] change [illegible]

[illegible] fix [illegible] value change not [illegible].

Demo d = new Demo (10)

10 — fix [illegible]

a110

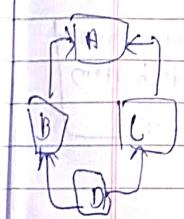
10x2025

object : value [illegible] physical memory
Instance : value [illegible] physical memory.

Types of Inheritance

Single (simple) Inheritance	multiple Inheritance	Hierarchical Inheritance
class A { }	class A { } / class B { }	class A { }
class B extends A { }	class C extends A,B { }	class B extends A { }
		class C extends A { }

Hybrid Inheritance



Multilevel Inheritance

class A
class B extends A
class C extends B

03/03/2022

Inheritance કહેવાય.

~~જ્યારે~~ એક થી વધારે type ને ઉપયોગ
થાય તેને Inheritance કહે છે.
એક Type same હોવા છતાં parent
class ની Property and Behaviour child
class માં વાપરી શકાય.

Parent and child વચ્ચે Inheritance
હોવાથી child class નો object બનાવવાથી
~~Chid~~ child class ની Property and Behaviour
and Parent class ની Property and
Behaviour access કરી શકાય.

eg. Person → Student
Employee → manager

*· memory Allocation in heritance

```
class Parent extends                 class Child extends Parent
{                     (objects       {
 → int a;              Implicitly     int b;
 → void fun()          (by JVM)       void fon()
   {                                  {
   }                                  }
 →Parent()                            Child()
   { }                                { }
}                                    }
Child c = new Child();
```

memory Allocation
Top to bottom
↓

Child class માંથી method or Function
Call થવા પછી શોધવા માં આવે છે.

Execution - Bottom to Top ↑

જ્યારે આપણે Child class ના main Function
માંથી કોઈ ~~ક્યાં~~ method અથવા Data member
~~method~~ ~~method~~ ~~ને~~ ...

method:-
~~Chid~~ Child class main Function
માંથી કોઈ method Call કરવામાં આવે તો
તે પહેલાં child ના main માંથી શોધશે
ત્યારબાદ child class માં, ત્યાં
ત્યારબાદ Immigiate parent ~~class~~
માંથી શોધશે એ રીતે ઉપર definition
execution કરશે. નહીંતર
error (Symbol not found error) આવે

~~* Static Data member, member method~~, Static Block.
~~memory Allocation~~
~~Execution → Bottom to Top~~ એટલે class load
થશે

~~memory Allocation~~
~~Top to Bottom~~

★ Static Data member, member function & Static Block.

memory Allocation

Top to Bottom ↓ each class load allocation.

Execution

Bottom to Top ↑

★ non-Static

memory Allocation:

Top to Bottom each object memory allocation

↓

Parent

↓

Child.

execution

Bottom to top ~~After Cation~~

Parent

↑

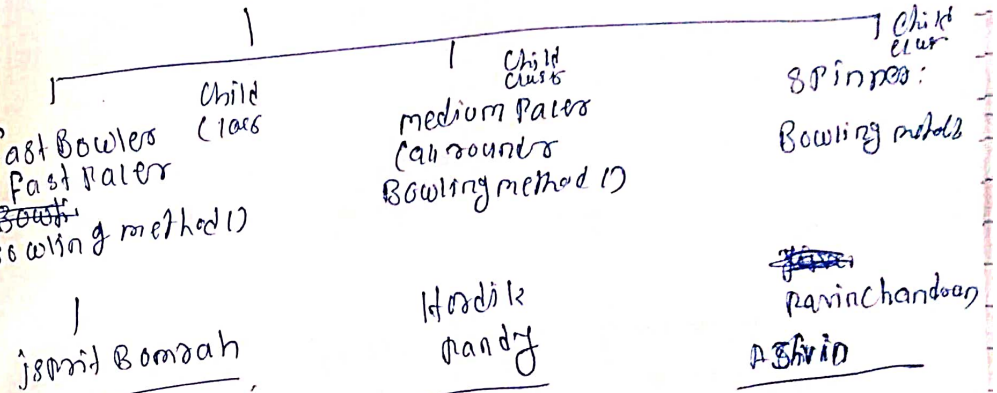
Child.

Bowler class : Parent

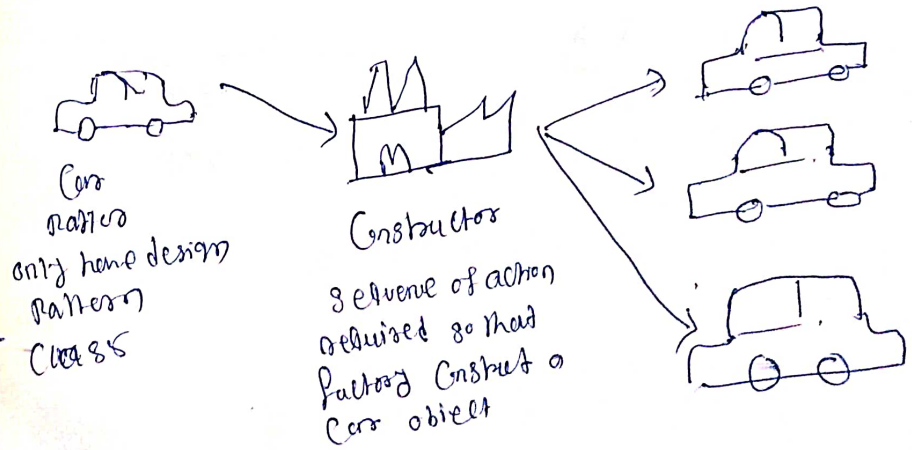
In a cricket team have bowlers and batmans bowler having faster bowler, medium matched as all rounder and spinner

class Bowler : Parent class

- bowlingMethod()



Class & object



Class and Interface
(extends and implements)

✓ Class —extends→ Class
✗ Class —Implement→ Class.

✓ Interface —extends→ Interface
✓ Interface —Implements→ Class

✗ Interface —implement→ Interface
✗ Interface —extends→ Class.

✗ Class —Implement→ Interface

* super()

Child class થી parent class ની value access કરવા.

04/03/2022

* Super keyword.

```
class Parent
{
    int a;

    Parent()
    {   super();
        this.a = 10;
    }
}

class Child
{
    int b;

    Child()
    {   super();
        this.b = 20;
    }
}

Child c = new Child()
```

class's object

implicitly by jvm

implicitly by jvm

Note:-

Child class ના constructor ની અંદર first line implicitly super હોય છે.

~~implicitly constructor ને call~~

Implicitly super() એ immediate parent ના default constructor ને call કરે છે.

જ્યારે jvm implicitly લખે

eg:- class માં એક પણ constructor ન હોય ત્યારે jvm એક constructor ને બનાવી દે છે ~~જેને~~ synthesis constructor કહેવાય.
(Default constructor - એક પણ parameter નહીં.)

આપણે class માં ~~parameter~~ parameterised constructor લખીએ તો jvm synthesis constructor બનાવતું નથી.

~~જ્યારે class માં આપણે કોઈ પણ constructor~~
~~child~~ Child class માં Parent class ના આપણે બનાવેલ constructor ને call કરાવીએ.

eg:- [DUagesh] - આ વ્યક્તિ હોય તેવા સાચા નથી.

super() explicitly લખવું પડશે.

```
class Parent
{
    int a;
    Parent(int a)
    {
        this.a = a;
    }
}

class Child
{
    int b;
    Child(int a, int b)
    {
        super(a);
        this.b = b;
    }
}
Child c = new Child(10, 20)
```

① ② ③

* method 4 (Data member) using super()

```
class Parent
{
    int a;

    void fun()
    {
        System.out.println(" Parent method");
    }
};

class Child extends Parent
{
    int a;
    void fun(int a)
    {
        super.a = a;      // Parent class
        this.a = a;       // Child class
    }
    p.s.v. main(String args[])
    {
        child c = new child();

        c.fun();
```

(i) Using super keyword [illegible] field [illegible] method access [illegible] parent class [illegible]

super.a; super.fun() — Parent class property & behaviour

~~Same name having~~ Number of behaviours having same name and its having multiple states (many forms) ~~called as polymorphism.~~ But each state to execute to depends on input is called as polymorphism. — definition of Polymorphism

(4) news

Animal — Animal — veg / non-veg

Animal → Eating — { veg, non-veg, grass } behaviours

(Animal = object, Eating = operation)

Poly^n → Polymorphism —
[illegible] object [illegible] operation [illegible] multiple behaviours is called as Polymorphism.

eg news example

```
class News_On_TV          // depend on input
{
    void news()
    {
        print(" good news");
    }                     // depend on input
    void news()
    {
        print(" bad news");
    }
```

કોઈ object ના control operation માટે multiple behaviour આપવો એટલે polymorphism

Types of Polymorphism

- Compile-time Polymorphism
- Runtime Polymorphism

(1) Compile time Polymorphism, Static binding, early binding — Static Binding

(2) Runtime Polymorphism, Dynamic binding, late binding. — Dynamic Binding.

(1) Compile time Polymorphism

Compile and linker Compile time ના સમયે કોઈ function invoked કરવાના સમયે symbolic mapping ચેક કરવા દેવાય. એટલે Compile time Polymorphism.

eg — Hotel

Family ના દીકરા જેવું જમવાના બાબત આવે તે આયોજન કરવામાં કે કયા hotel માં જવાનું.

(2) Runtime Polymorphism

Compiler અને linker Compile time function ની definition check કરતા હોય.

Runtime કોઈ function call થાય ત્યારે તે વખતના Type of parameter, no of parameter, and sequence of parameter ચેક કરવામાં આવે તેને Runtime Polymorphism.

(1) Function overriding — Runtime Polymorphism

eg - [Car], [employ → manager] — calculate salary.

def.ⁿ

Function overriding એ હંમેશા Inheritance માં થાય છે.

Consider - Parent→Child

Parent class માંનું function child class માં as it is લખવામાં આવે function નું નામ, return type, no of parameter Same રાખવામાં, ત્યારે function overriding થાય છે.

Function overriding ક્યારે કરવાની.

Parent class માંથી આવેલ function આપણે use કરી શકતા નથી. ત્યારે child class ની requirement fulfill કરવા માટે function ની overriding કરવાની child class માંથી.

Function overloading - Compiletime polymorphism

* એક જ function દ્વારા multiple behaviour

Parent class અથવા child class માં
[illegible] એક જ function નામ પણ ત્યારે
Type of Parameter, sequence of parameter
ને no of Parameter ને change કરવાના
Return type કોઈ matter નથી કરતા

eg - (1) Person (Human)

Family | Teacher | Developer | Student

(2) master નું example

master

ઘરે પપ્પા | સ્કૂલે વિદ્યાર્થી | Developer ઓફિસે

* Method hiding

Parent class માંની static method આપણ પછી child માં static કરીને લીધેલી ત્યારે આપણે method hiding કહેવાય.

method overriding
Parent class માંની non-static method child class માં as a તેમ જ method non-static કરીને

Parent ની static method આપણે તેમ child ની non-static ન કરી શકાય.
Compile time error હોય

Parent ની non-static આપણે તેમ child ની static કરીને આપણે method ને વાપરી શકાય નહીં.
Compile time error.

* Binding -

Function call ના ત્યારે appropriate definition શોધી compile time ઉપર જ binding કહેવાય.

★ Upcasting and DownCasting.

① Upcasting

Upcasting એટલે Parent ના ~~object રાખવો~~ reference variable રાખે છે. આમાં child ના object રાખવો.

child ના Object ના Heap માં રહેલા address Stack માં ~~emp~~ parent ના Reference variable માં રાખીએ તેને upcasting કહેવાય.

Using Parent Reference variable આપણે child માં Parent ની ~~Data mem~~ field + method access કરી શકાય.

જેના આપણે child ના object માં parent ના Reference variable રાખતા નથી child ના memory જેટલી જગ્યા હોવાથી parent ના ના હોય શકે.

② DownCasting.

Child ના Reference variable માં parent ના object.

~~Java~~

Java DownCasting not allowed

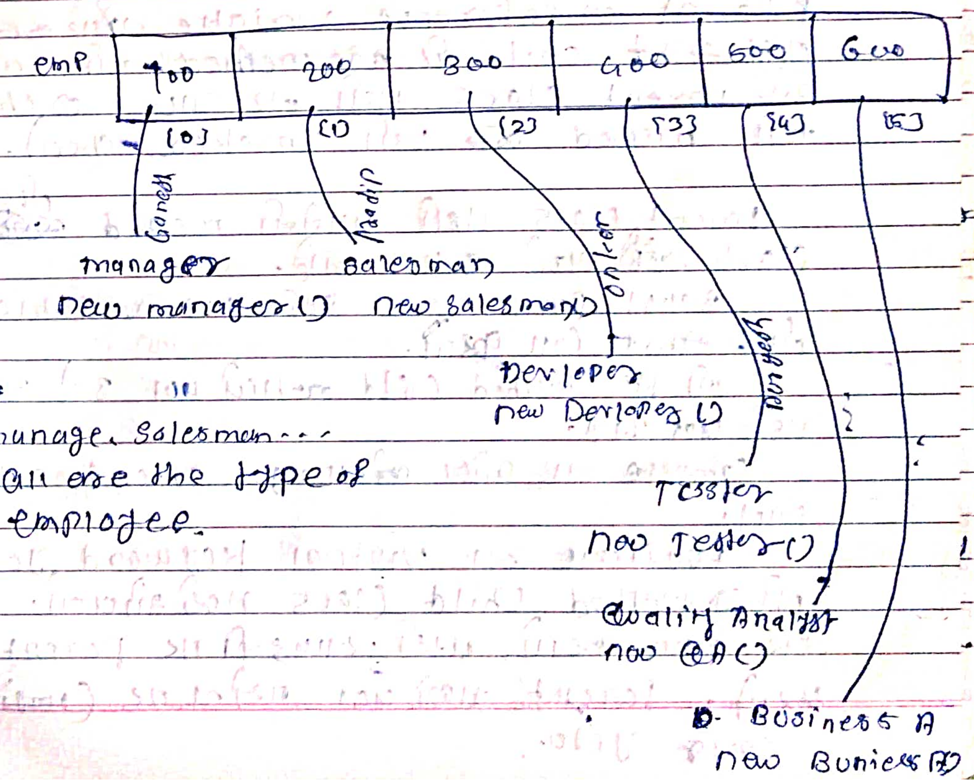
Child ના Reference variable બનાવવા માં Parent ના object રાખવા જઈએ ત્યારે Parent ના બધા મેથડ child ના બધા મેથડ નથી હોતા Downcasting allowed નથી.

Q.1] Parent Class માં આવેલ method ના name return type અને no of parameters as it is તે method Child Class માં હોય તો ...

Q.2] Child ના Reference variable માં Parent ~~Employee~~ ના object કરી શકાય નહીં.

~~Runtime Polymorphism~~

Employee emp[]



manage. Salesman...
all are the type of employee.

overriding char/ function/method
વખતે case મૂકેલ હોય. Data member વખતે
case મેલ હોય નહીં.

overriding થતી હોય.

By default written method ની virtual રહેતી java માં.

Parent class ના reference variable દ્વારા child ના object શરૂઆતમાં method overriding થતી હોય.

Child ને parent type ના ટાઇપમાં મૂકે આપણે parent ના reference variable દ્વારા child ની method. ની same parent class માંથી રહે આપણે child ની method કરી Invoked કરવી.

Parent class માંથી ફક્ત method લીધેલી child class માંથી જ પણ આવે.
આપણે parent ના reference variable ની ની method call કરવી.
ની method child માંથી પણ કરે આપણે parent માંથી પણ કરે.
Compile ના ફાયદા બંધાયેલા extend કરે છે કરવા જોઈએ.
Runtime ના virtual keyword જેટલું આવે ની method child class માંથી મળે. ત્યાર બાદ run કરવાથી ત્યારે સાથે ની parent ની run કરવી. Parent માંથી પણ મળેલ ના compile time error જેટલું.

virtual keyword implement jvm take to runtime ના

```
Class Employee
{
    int sal;
    Calculate_Salary()
    {
        this.sal + 2000;
    }
    int get sal()
    { return this.sal;
    }
}

Class Manager extends Employee
{
    int bonus;

    Calculate_Salary()
    {
        b = getsal() + bonus;
        b = getsal() + bonus;
    }
}

Employee e = new Manage()
e.Calculate_Salary()
```

ની કોલાણો compile ના ફાયદા extend કરે બંધાયેલ

આ રીતે બંધાયેલામાં method overriding
દ્વારા Runtime Polymorphism કહેવાય!

~~Function Overriding~~

~~winding autoboxing varargs~~ variable ~~or~~ arguments.

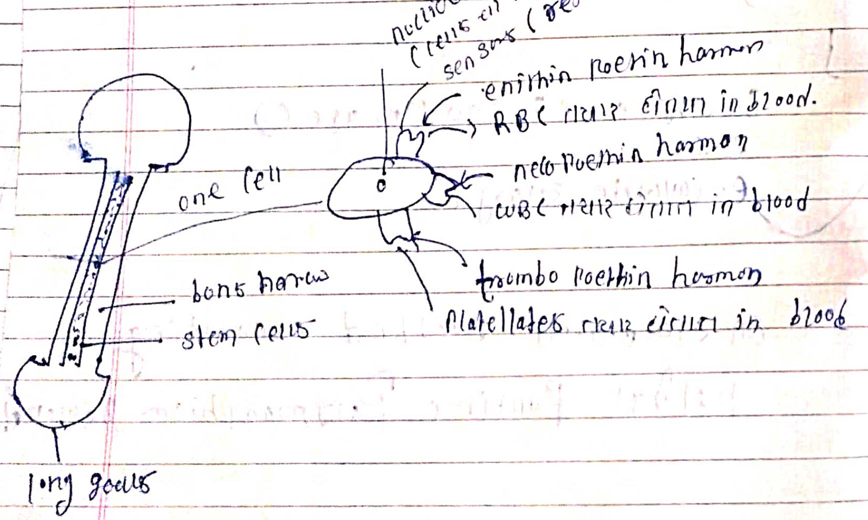
Example

In body

વારસાગત
આપણા શરીરમાં ત્રણ cells હોય છે RBC, WBC, Platelates

Stem cells હોય છે રક્તમાંથી પેદા [illegible]
[illegible] પેદા

Cells Can not multiply to eachother and Can't Reproduce new Cells they need to generate mannually.



② function overloading

જ્યારે એક scope માં જ્યારે બે કે તેથી વધારે function same name વાળા હોય પણ તેમાં input અલગ હોય અને logic same હોય ત્યારે Function overloading કહેવાય છે.

eg. Government Bank and Private Bank

① Function overloading કરતા સમયે function ના નિયમ યાદ રાખો.

① બંનેના નામ same હોવા
② બંનેના return type same હોય તે જરૂરી નથી.
③ બંનેના logic same હોય.
④ Parameter
① no of Parameter-
બંનેના function ના number of parameter એક સરખા ના હોય.

② sequence of parameter
બંનેના function ના parameter ની sequence અલગ હોય.

③ Type of parameter
બંનેના function ના type અલગ અલગ હોવા.

⑤ બંનેના function ને call કરતી વખતે [illegible] call કરવા.

widening

(1) Wideing :-

(1) નાના data type ની value મોટા data type માં implicit બેસાડે. તેને widening.

eg.

```
char c = 'A';
int a = c;
```

(2) બંને Data type સરખા હોવા જોઈએ.

(3) મોટા data type નાના data type માં implicit બેસતા નથી તેને widening કહેવાતું નથી.

(2) AutoBoxing.

```
Class Demo
{
    void fun( Integer i)
    {
        System.out.println(" value = "+ i);
    }
    public static void main( String args[])
    {
        Demo d = new Demo();
        d.fun(10);
    }
}
```

Function

method ના parameter એ ક્યારે reference variable સ્વીકારે ત્યારે method ના call કરતી વખતે value પસાર કરીએ તો value તે reference variable માં જાય તે AutoBoxing કહેવાય.

(3) Varargs (variable arguments)

જ્યારે function ના arguments ની સંખ્યા નક્કી ન હોય ત્યારે fix ન હોય (એક કરતા વધારે arguments) ત્યારે varargs વપરાય છે.

Package -

Package - java ની અંદર એક ~~folder~~ folder જેવું હોય છે જેમાં compile file રાખવામાં આવે છે. .class file આપણે કેવી રીતે રાખવી, તેના package બનાવવા.

src - source file (.java file)
bin - binary file (.class file)

Package બનાવવા માટે આપેલા Rules

~~Class Demo~~

~~Package com.monro.loan.ici~~

~~Package com.monro.icic~~

Package com.tcs.icici.loan.Homeloan
(company name / customer name / product / submodule)

```
Class Demo
{
    Public static void main(String args[])
    {
        System.out.println("Icici Bank");
    }
}
```

Rules:-

1. Package ની line no 0 હોવી જોઈએ (કોમેન્ટ સિવાય)
2. Package ની line પછી import statement આવે.
3. એક class માં multiple line Package ન હોય.
4.

Compilation અને Run કરી શકાય.

Compilation

1. Command: javac -d . Demo.java (directory) (linux માં પણ ચાલે)

   [illegible] compile file (.class) [illegible] Package system [illegible].

2. Run the Class file

   java com.tcs.icici.loan.Homeloan.Demo

3. Class file માંથી Change કરેલ Code જોવા માટે

   javap com.tcs.icici.loan.homeloan.Demo.Class

   આપણે class માંનો Code જોવા મળશે.

~~import.~~

* Include & Import.

(1) Include –

include થી line જ્યાં folder system માંથી ~~આવેલા~~ આવેલા code copy કરવામાં આવે ત્યાં line મા Replace કરાય છે.

બીજા ~~source~~ source માંથી external code ઉમેરવામાં આવે છે.

આને include કહેવાય છે.

(2) import

import કરવાથી ત્યાં folder system માંથી ની class file શોધીને વાપરો ને આપણા program ત્યાં આવે ને code ના લખવો પડે કે ઉપયોગ કરીને code વાપરો જરૂરિયાત ની class file ~~વાપરો~~ વાપરો.

- Bad class file / Bad source file :
- class file Deployment કરવા માટે
  - jar file બનાવવા માટે વપરાય
  - jar manifest બનાવવા માટે
  - ~~એક વાર jar file~~
- એક project માંથી jar file ~~ઉમેરવા~~ બીજા project માં ઉમેરી શકાય છે.

Abstract (Abstraction)

~~Interface~~

~~Abstract ...~~ Abstraction કોને કહેવાય?

Data અથવા method hide કરવી એટલે Abstraction કહી.

કેવી રીતે private કરવી class ની અંદર

Data hiding

આપણે class ના Data members private કરીએ તો class ની બહાર accessible નહીં થાય. આને Data ની hideing.

આપણે એની value set અને get ... કરી શકાય.

method hideing.

method private કરવાથી એ method બહાર accessible રહેતા નથી આને method hideing કહેવાય પણ તે abstraction હોતું નથી.

ખાલી method private કરવાથી કોઈ method બીજા class ની બહાર વપરાતા હશે તો તે private કરાતા નથી.

જે method ખાલી class માં જ વપરાય તેને private કરવી.

Example

1. ATM મશીન આપણને પૈસા દેખાય. પણ તે કેવી રીતે એ ખબર નથી.
2. Copy કરવા જો કરીએ operating system તે ખબર નથી.
3. Keyboard
4. Amazon પરથી પૈસા Pay કરવાના → Bank.

Abstraction ત્યારે થશે જ્યારે working હોય.

1) Abstraction માં આપણે Parent ના Reference માં Child ના object બનાવીએ.

eg.

Amazon ના Reference

↓

Bank ના object.

2) Abstraction માં method ના Return type Function નું નામ, no of Parameters ફિક્સ હશે. અને Return value ની PA ના Function ના Return type પરથી depend કરશે.

આ method ની ઉપર Body નથી. તેની Body બીજે ક્યાંક લખેલી હશે.

~~Abstract Class ના Concrete Class ...~~ જો એક પણ abstract method હોય.

Abstract Class

```
Abstract Class Amazon
{
    abstract void Pay_money();
    void cancel_payment(); {}      ← Concrete method
}
```

જો method abstract આપતો હોય તો દરેક class ના આગળ ફરજિયાત પણ તો class abstract આપવો.

Abstract class માં એક પણ abstract method હોય તો તેને લખેલા તો class abstract બનતો.

Abstract class માં concrete method પણ આપી શકાય.

Abstract class ના object તૈયાર કરવા શક્ય નથી.

* Concrete class

Concrete class માં Abstract method ન આવવા. ખાલી concrete method આવવા.

સામાન્ય બધા class.

* Interface

Interface ઉમેરતા સિવાયના બધા આપણને method abstract બનાવવી જરૂરી નહિ લખવાનું

Interface માં concrete method આવતા નથી ખાલી abstract method આવવા.

```
Interface Test
{
   abstract void fun();
   abstract void gun();
}
```

interface માં by default દરેક method abstract હોય છે તે public પણ હોય.

Interface માં variable ની value ફરજિયાત આપવી પડે અને તે variable interface ના નામથી access કરવા પડે.

interface to Interface - extends ✓
implement ✗

interface to class - implement ✓
extends ✗

class to class - extends ✓
implement ✗

Inner class.

Inner class કોને કહેવાય?

class within a class is called inner class.

eg.

```
class outer
{
    class Inner   — inner class.
    {

    }
}
```

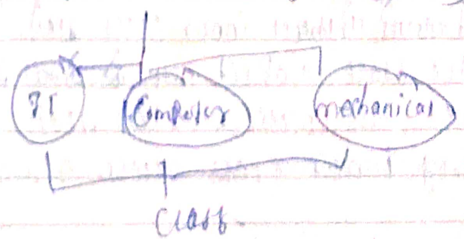
Inner class ક્યારે બનાવવા.

જ્યારે બનાવેલા class independent હોય અને કોઈ બીજા પર depend નથી. ત્યારે a class ને ~~બીજા ક્લાસમાં~~ બીજામાં class object બનાવીને use કરવામાં આવે તેને composition કહેવાય.

~~Inner class~~

જ્યારે ~~બનાવેલ~~ a બનાવેલા class નો direct object બનાવવાની જરૂર નથી ~~ત્યારે~~ તો inner class બનાવવા.

eg.



Inner Class ના 4 Types

1. Normal Inner Class
2. method local inner class
3. Anonymous inner class
4. Static nested class

1. Normal Inner Class.

1. એક Class માં બીજા class લખીએ ત્યારે inner class કહેવાય.
2. normal inner class કહેવાય [illegible] class [illegible] class [illegible] normal inner class કહેવાય. with Property and behaviour

Rules.

1. outer class માટે Inner Class single accessible હોય.
2. outer class ના Inner class [illegible] નહીં.
3. Inner Class માં static data member અને method લખાતા નહીં. jdk-16 પહેલાં jdk 17 [illegible]
4. outer class ના Private Data member [illegible] inner class માં.
5. outer class ના object [illegible] inner class access [illegible] નહીં.

3 way inner class access કરાય.

1. using outer object.

outer class ના ~~object~~ Reference variable (object) બની શકે.

અહીં આપણે outer class ના ~~data~~ field and method [illegible] outer class object [illegible] Reference variable [illegible]

Using outer class object inner class ને access કરાય.

Syntax:

```
Outer o = new Outer();
Inner i = o.new Inner();
```

2. outer class ના object [illegible] inner class ના object [illegible]

Syntax:

```
Outer.Inner i = new Outer().new Inner();
```

~~Outer Inner i = Outer.new Inner~~

~~Outer.Inner i = Outer.~~

3. [illegible] outer class ના object [illegible] Inner class ના object [illegible]

without creating Reference variable of any class and access inner class

Syntax:

```
new Outer().new Inner().<field, method>
```

② method Local inner class.

જ્યાં class ના અંદર જ્યાં method હોય ત્યાં આપણે method Local inner class બનાવી ત્યાંના રાખેલા Property & અને Behaviour રાખી શકાય.

method Local inner class ના object outer class ના method માં બહાર બનાવી શકાતા નથી.

• method Local inner class નો ઉપયોગ ક્યારે કરવો?

① function અંદર કરવા માટે.

જ્યારે આપણને no of times Repeatedly execute કરાવવાની જરૂરિયાત હોય ત્યારે multiple પ્રોપર્ટી વાપરવી જરૂરી હોય ત્યારે આપણે Loop ના બદલે function બનાવીએ.
આપણે તે Function માં Inner Class ના બનાવી શકીએ.

```
class Demo
{
    void fun()
    {
        class Test
        {
            Property &
            Behaviour
        }
        Test t = new Test();
    }
}
```

Rules:
1) outer Class method ના variable method Local inner Class માં વાપરતા હોઈએ તો method local inner class એ final વાપરવા પડે.

~~જ્યારે આપણે તે variable Data member~~
જ્યારે આપણે તે variable વાપરતા હોય ત્યારે Compiler compile time error આપે માટે final રાખવું.

③ Anonymous Inner Class.

1) Anonymous Class ને બનાવતા એ Class નું નામ આપવું પડતું નથી Physical existance હોતી નથી.

તેના class નું Physical existance હોય પણ class ના અંદર Under હોય છે.

• Anonymous class ના હંમેશા Parent child ના Relationship હોવું જોઈએ.

why to write Anonymous class

જ્યારે Parent class Pass child class inherit કરીએ, અને તેનો class અલગથી બનાવવો પડે, કોઈ પણ ક્લાસમાંથી વાપરવાની જરૂર

હોય ત્યારે Property અને Behaviour ના ફેરફાર કરવા ખાલી ઓવરરાઈડ કરવાના હોય ત્યારે આપણે Anonymous Inner Class બનાવીએ.

eg: Employee
↓
manager

Employee પરંતુ manager class ને Inherit કરાવી. પણ આપણને manager ને calculate-salary() function વાપરવાના જ હોય તેના પરંતુ બધી property Behaviour આવે છે.

બીજાના property અને Behaviour ના ઉપર કરવાની જરૂર નથી, આપણે manager ના Anonymous inner class લખીશું.

• Anonymous Inner class કેવી રીતે લખવા

Inner class child class

Anonymous inner class કહેવાય child class.

outer ના parent class ના object create કરતા inner class લખાય.
like,

```
Outer o = new outer()
{
    //Anonymous Inner class
    //fields and methods
};
```

• Anonymous inner class માં method overriding કરાવી શકાય છે.

Anonymous Inner class માં ફક્ત Inheritance મળે છે તેમાં નવી મેથડ ઉમેરી શકાતી નથી.

A

(4) Static nested class

Static nested class એટલે કે એક class ના અંદર બીજો class લખીએ તો static class કહેવાય.

Static nested class માં class load થાય ત્યારે જ memory એલોટ થાય.

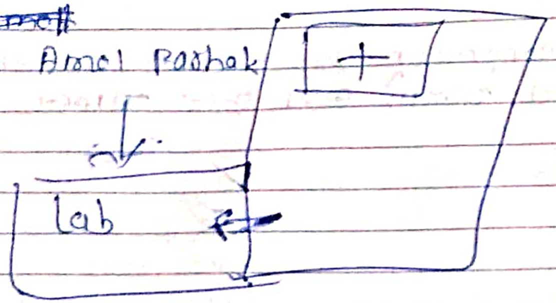
Static nested class માંથી આપણે બહારના property અને Behaviours ને સીધું access કરી શકાતા નથી. main method સિવાય લખી શકાતા નથી.

main method static હોવાથી તું એને ત્યાંથી ત્યાં static nested class માં static field તથા static method વાપરવા allow કરેલ છે.

outer class માં static હોવાથી static nested class માંથી વાપરતા non-static માટે outer ના object બનાવવા.

Static nested class કેવી રીતે કામ કરે છે વિચારવી.

eg- ~~Amit~~



કેવી વિચારવા

જ્યારે static class ના class load થાય ત્યારે memory ફાળવે. જેનો રીતે સમજો કે static nested class વિચારવી અને outer class નાં નામ સાથે મળી ને class ના નામ દ્વારા use કરી શકીએ.

Static nested class ના object બનાવવા માટે outer class નામ.

eg-

```
Class Demo
{
    outer_class_name.static_N_C_name.  r  reference variable name
        = new outer_class_name.static_N_C_name();
}
```

આના object દ્વારા કરવાના રહેશે તથા static nested class ની Property અને Behaviour access કરી શકાય.

outer class ના object બનાવ્યા વગર જ static nested class access કરી શકાય.

Abstract Class

Q. Concrete class માં શાથી હોતા abstract method ન હોય?

Concrete class ~~માંથી~~ ના object બનાવે છે તેના abstract method ના call કરતા તે program runtime crash થશે. કારણ ~~abstraction~~ abstract method ના body નથી.

જો class માં એક abstract method આપેલ હોય તો એ class પણ abstract કરવા પડશે.

abstract class ના object બનાવી શકતા નથી.

* abstract class ના Reference variable બનાવી શકાય છે.

abstract class માં Concrete અને abstract method હોય છે.

abstract method નો ઉપયોગ ક્યાં?

eg. Showroom — Tata Company (abstract method)

↓ Showroom ↓ (implementation of these method)

Bank, Amazon

abstract method ના Declaration parent class માં હોય છે.

જેમાં method ના return type, method name, parameter, તે અગાઉ થી નક્કી કરેલ હોય છે.

parameter ના sequence.

Child class ના object મારફત જ તેના med. ના implementation નક્કી કરવામાં આવે છે.

Child class ના object દ્વારા જ્યારે Child class માં આવે child class ના ... method નું implementation કરેલું હોય.

abstract method ક્યારેય static નથી. કોઈ પણ modifier ... વાપરી શકાય નહીં.

class abstract હોય તો જ ...

જે class ના object બનાવી શકાતા નથી તે abstract કહેવાય.

એ class માં concrete અને abstract method હોય.

abstract class ને Constructor હોઈ શકે.

abstract class માં abstract method private હોઈ શકે નહીં ... public જ હોય.

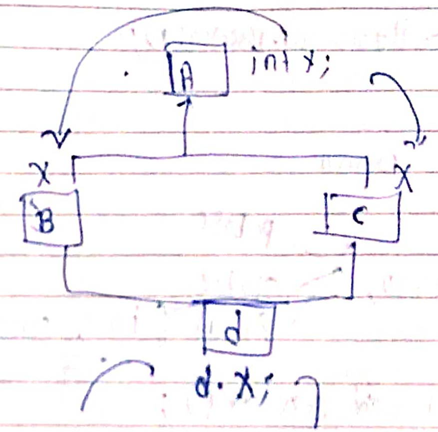
abstract class ના Constructor ને call ...

```
abstract class Parent
{
  Parent()
  { }
}
class Child extends Parent
{
  Child()
  { super();
  }
```

abstract Parent class ના આપણે object બનાવી શકતા નથી. તેના Constructor call થાય છે.

જ્યારે class inherit થાય ત્યારે આપણે abstract Parent ના Constructor Call કરી શકીએ using child object.

multiple Inheritance



C++ માં virtual base class use કરવામાં આવે છે. ambiguity generate થાય. તેથી java માં multiple Inheritance ચાલતું નથી.

Interface વડે આવે છે.

Interface થી multiple heritance શક્ય બને છે.

Interface ના method abstract હોવાથી ambiguity Resolve કરવા માટે Interface આવ્યા.

~~Interface Demo~~
~~{~~
~~int a = 10;~~
~~abstract void~~
~~abstract void getValueOfA()~~
~~{~~
~~System.out.println(a);~~
~~}~~

Interface Demo
{
int a = 10; ← public static final (in C constant)

abstract void getValue();
}

method:

Interface માં abstract by default public હોય છે. તેને આપણે private બનાવી શકતા નથી.

Interface માં બનાવેલ method આપણે static રાખી શકતા નથી.

આ method ના ફક્ત Declaration હોય છે implementation હોતા નથી.

Interface માં બનાવેલ method by default abstract હોય છે.

Data member:-

Interface માં બનાવેલ દરેક data member public હોય છે.

Interface માંના Data member static હોય છે. કારણ કે interface નો object તૈયાર કરવો શક્ય નથી.

Interface નો object તૈયાર કરી શકાતો નથી એટલે interface માંના variable by default static બની જાય છે.

આ variable interface ના reference થી access કરાય છે.

Final:

આ data member value સાથે જ ફરજીયાત આપવી પડે છે. કારણ, static block, non-static block કે constructor માંથી આપણે value ના આપી શકાય. એટલે value declaration વખતે આપવી પડે અને final હોય છે.

Interface to Interface	extends
Interface to Class	implement

extends કરવાથી એક જ interface વાપરી શકાય.

implements - થી એક સાથે ઘણા interface વાપરી શકાય છે.

* ~~Interface શરી લખવાની~~.

* Interface વિશેષતા જણાવો.

multiple object માટે એક common behaviour નક્કી કરે છે તેથી તેના Behaviour ના implementation પ્રત્યેક object કરતા હોય છે.

Interface માં જે બનાવેલા abstract method હોય તેને class માંથી implements કરાય છે.

જ્યારે આપણે બનાવેલા Interface class માંથી implements કરીએ abstract

ત્યારે તેના interface માંથી જે આપણાં abstract method ની definition લખવી ફરજિયાત બને છે.

જો આપણે તેના abstract method ની definition લખી દઈએ તો તેના class object બનાવી શકાય તો તે Behaviour access કરી શકાય.

જે class માં implement કરીએ તે abstract method ના access modifier Public હોય તો access modifier interface માં રાખેલા હોવા જોઈએ.

પણ આપણે તેના abstract method ની implementation લખી નથી તો તે class પણ abstract બનાવી abstract class ના object બનાવી શકાતા નથી.

જે કોઈ class માં abstract method ની definition લખેલી નથી તેની child class બનાવી તેમાં abstract method ની (class ને abstract ના બનાવો).

તે class માંથી એક child class બનાવી તેમાં abstract method ની definition લખવી.

eg.

```
interface vechile
{
    void engine();
    void break();
}
```

by default આ class abstract છે.

```
class BUS implements vechile
{
    Public void break()
    {
        System.out.println("Pushed break");
    }
}
```

આ class એ abstract નથી by default.

```
class Tata_BUS extends BUS
{
    Public void engine()
    {
        System.out.println("Engine V8");
    }
}
```

```
Tata.BUS TB = new Tata.BUS();
TB.break()
TB.engine();
```

Diff.

Class	Interface
① Class માં abstract અને concrete method હોય શકે. જો abstract method હોય તો class abstract હોય.	① Interface માં abstract method હોય છે. અને તે by default abstract અને public હોય છે.
② Class માં method ની Body હોય.	② Interface માં method ની body ન હોય.
③ abstract class નો object બનાવી શકાતો નથી	③ Interface નો object બનાવી શકાતો નથી.
④ Class to class extend થાય.	④ Interface to interface extend થાય.
⑤ Class ને interface implement થાય.	⑤ Interface to class extend થતો નથી.

Similarity

① Class ના reference variable બનાવી શકાય.	① Interface ના પણ reference variable બનાવી શકાય.
② class માં method ની definition હોય.	② Interface માં પણ method definition હોતી નથી.

* Interface માં abstract method body હોતી નથી.

```
Interface Demo
{
    static void fun()
    {
        //definition
    }

    default void gun()
    {
        //definition
    }
}
```

Static :-

1. Interface માંની static method ની body હોય છે.
2. Interface ના reference variable બનાવી શકાય ના ત્યાં method ને call કરવા હોય.

default :

1. Interface માંના method by default public અને abstract હોય છે.
2. જો explicitly default keyword લગાડીએ તો compiler ત્યાં method ને concrete method સમજે છે. અને ત્યાં method ની body હોય છે.

TPAJ - government company

Sim Card Example

```
Class Mobile
{
    void insertSim(Sim s)
    {
        s.DialCall();
        s.SendMessage();
    }

    public static void main(String args[])
    {
        Mobile mi = new Mobile();
        mi.insertSim(new jio());
        mi.insertSim(new Airtel());
    }
}

Interface Sim
{
    void DialCall();
    void sendMessage();
    default void check()
    {
    }
}

class jio implements Sim
{
    public void DialCall()
    {
    }
    P.S.V.m(String args[])
    {
    }
}

Class Airtel implements Sim
{
    Public void SendMessage();
    }}
}
```

Note:-

Interface [illegible] method
no body [illegible]

[illegible] object
[illegible] method Common [illegible].

[illegible] Common method Use [illegible]
child [illegible] object [illegible] override
[illegible].

lang - Package

[illegible] Package [illegible] Class file Store [illegible]
[illegible].

Package -

- Package } [illegible] Collection [illegible]
- Class file } package [illegible].

Package
- java
- javax (extended)

import java.lang;

[illegible] predefine Classes [illegible]
interface [illegible] and abstract Class [illegible].
Concrete Class [illegible].

lang Package [illegible] program Run
[illegible] default import
[illegible].

[illegible] Package [illegible] Class [illegible]
[illegible] import statement [illegible] fully qualified
used to

[illegible] predefined Case [illegible] identifier [illegible]
[illegible].

Fully Qualified Path:-
import java.lang.String;
[illegible] implicitly import [illegible].

category

lang - Classes

① Object
દરેકના Class ના Parent Class છે.
Object Class છે.

② Class
Class એ Class ની આપણા Class ના Run time ના represent કરવી. આપણે આપણા Class ના object તૈયાર કરવા કે આપણા Class ની બધી information store [illegible] Class વાપરતા Class ના object તૈયાર છે.

③ String / StringBuffer / StringBuilder - String handling

④ Throwable / Exception / Error - Exception handling

⑤ Thread / ThreadGroup / Runnable / - multithreading
Runnable → Interface
ThreadLocal

⑥ Byte / Short / Integer / long / float / Double /
Character / Boolean - Wrapper Classes

⑦ System / ClassLoader / Process - Special Classes
marker Interface → jvm વાપરતો આપણે વાપરતા નથી.

lang - Package

import java.lang; → predefine package છે. આપણે import કરવું નથી. [illegible] [illegible] છે. પણ import કર્યા વગર તેની class વાપરવી.

lang package ના under classes આવેલા. [illegible]

① Class
② Object

① Class.
Class નામનો એક Class છે જેમાં Class ની બધી Information store કરવી.
- Class નું નામ
- Class ના property (fields)
- Class ના methods
- Class ના object.

Class નામના Class માં આપણે Class define કરીએ તેના Class ની information કરવી.

આપણે લખેલા દરેક Class ની information એ Class નામના Class, object માં information આપે.

Function Chaining - એક function ના call કરવાના output ને બીજા function ના input તરીકે ઉપયોગ કરવો તેને function chaining કહેવાય.

② Object Class

કોઈ method શીખવા object ની ડેટા print કરાવવા માટે function કરે છે.

```
Public String toString()
{
    ~~return this.className() + "@" + Integer.hexa~~
    return this.getClass().getName() + "@" +
        Integer.toHexString(hashCode());
}
```

① આપણી class get કરાવવા

② class ના નામ get કરાવવા

③ @ symbol concatenation

* પહેલી class ની આપણી information get કરાવવી. આપેલી class ના નામ પછી @ symbol concatenate કરાવવું.

~~આપણે તે આપણે return કરાવવું~~

Integer નામના class માં આવેલ toHexString() method માં આપણે hashCode() method parameter તરીકે પાસ કરવી.

* hashCode આપેલા નંબર hexadecimal માં ને String માં convert કરાવવા toHexString() method Integer નામના class માં આવેલ છે.

* આપણે આ method return કરવા માંગીએ છીએ તે String છે. તેથી return type String છે.

Object Class નો use શું?

Object class એ આપણા બધા class નો parent છે.

Object class એ direct upper છે.

આપણા બધા class એ implicitly object class માંથી inherit થાય છે extends કરેલ છે.

eg.

```
Class Demo extends java.lang.object
{
}
```

↑ implicitly JVM ઉમેરે.

Object class નો use object માંના data ને String માં print કરવામાં થાય.

Object class એ import java.lang package ના under છે. તે predefine class છે.

~~આ object class ના આપણે override કરી શકીએ. પણ overriding કરવાથી નવી definition થી આપણા method same રહે.~~

Object class માં 3 method હોય છે.

① toString()

② equals()

③ hashCode()

(1) toString() - toString() method is non-static method.

toString method आपल्याला object मधला data print करायला देतो. म्हणजे data print करताना to as a String print होतो.

toString() method ला explicitly call करावा लागत नाही implicitly call होतो...

```
public String toString()
{
    return this.a + " " this.b;
}
```

toString method ही object class मधली आहे.

toString method ला call केल्यावर object मधला data console वरती दिसतो.

object class ची toString() method call झाल्यावर data print होण्यासाठी म्हणून toString method आपल्या आपल्या class मध्ये आपण override करतो. (म्हणजे overriding मध्ये definition बदलता येते नाही तर बदलता नाही).

implicitly calling कशी होते.

```
class Demo
{
    int a;
    int b;

public static void main(String args[])
{
    Demo d1 = new Demo();
    System.out.println(d1.toString());   // explicitly calling
    System.out.println(d1);              // implicitly .toString JVM लावतो.
```

(2) equals()

equal object comparison करण्यासाठी नाही आतली ती reference चे comparison करतो जो reference stack वरती असतो.

equals method object class च्या under आहे.

तीचा return type boolean आहे.

जर equals() method चा used object चा comparison करायचा असेल तर ती method आपल्या class मध्ये override करावी लागते.

```
public boolean equals(Object obj)
{
    final
    return this == obj;
}
```

public - कुठूनपण access करता आले पाहिजे.
boolean - true किंवा false return करायला.
equals - method चे नाव आहे

Object obj - प्रत्येक class चा parent हा object class आहे. म्हणून इथे up-casting वापरली जाते. म्हणजे parent चा reference आणि child चा object

~~return this == obj;~~ -
- this म्हणजे child reference असतो.
- obj म्हणजे

return this == obj;

this મતે પહેલાના class Child ના reference હોય છે.

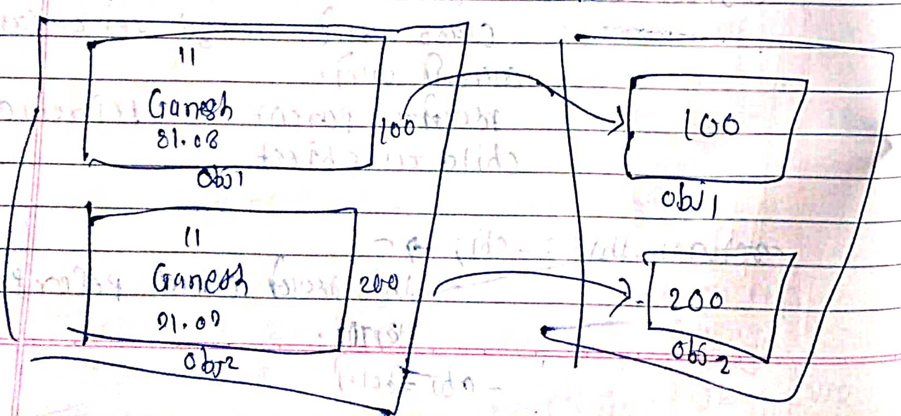
as a parameter કરવું હુઆ child class ના દરેક object as (ડી) method ના parameter મળ્યા હોય છે.

~~Class Demo~~

```
class Student
{
    int rollno;
    String name;
    float per;

    void calculate()
    {
    }
}
```

```
class Test
{
    public static void main(S arg)
    {
        Student obj1 = new Student();
        Student obj2 = new Student();
        s.o.p( obj1.equals(obj2) )
    }
}
```

In Heap   Stack



equals() method object ના value check કરવા માટે. ચકાસવું reference check કરશે.

દરેક object heap માં અલગ જગ્યા લેશે. ત્યારબાદ value ચકાસી શકાતી નથી. ત્યારબાદ object ના value ચકાસવામાં આવ્યા તો output false હોય. કારણ address અલગ હોય છે.

આ કારણથી આપણે આ method override કરવી.
Equals() method non static છે.

example

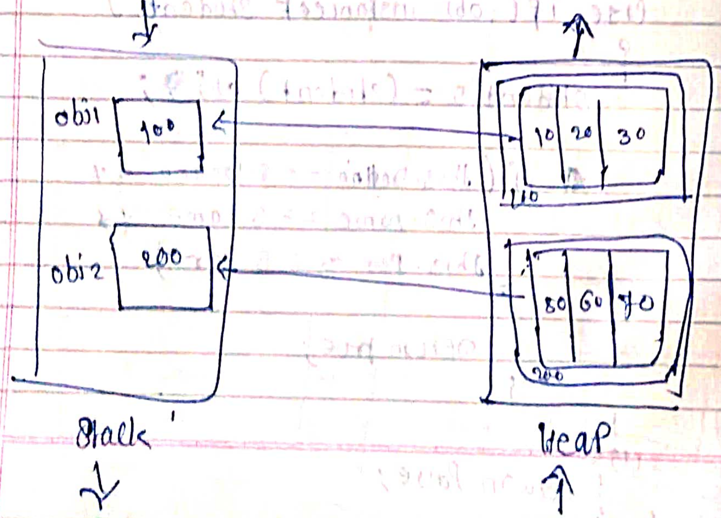
```
class Student
{
    boolean equals(Object obj)
    {
        if( this == obj )
        {
            return true;
        }
        else if( obj instanceof Student )   // java ના operator
        {
            Student s = (Student) obj;

            if( this.rollno == s.rollno &&
                this.name == s.name &&
                this.per == s.per )
            {
                return true;
            }
        }
        else
        { return false;
```

આના overrided method object data same હોય તો true return કરશે.

Note:-
જો બે reference variable એક જ object ને point કરતા હોય તો એક જ object ના address સાચવતા હશે.

Q. પ્રત્યેક વખતે object તૈયાર કરવામાં આવે ત્યારે value આપવા માટે instance variable માં પણ પ્રત્યેક વખતે નવી જગ્યા Heap માં જ બને.

→ જ્યારે એક class ના multiple object તૈયાર કરવામાં આવે ત્યારે ~~no of Data~~ no of parameters & type of parameter same હોય છે, પણ તેમાં રહેલી value પ્રત્યેક object ની value અલગ હોય છે.
તેથી જ પ્રત્યેક વખતે Heap memory માં નવી પ્રત્યેક object ની value store કરવા નવી જગ્યા બને, તે જગ્યા ના address stack માં હોય છે.



3] hashCode() method

hashcode method એ object class ની under છે.

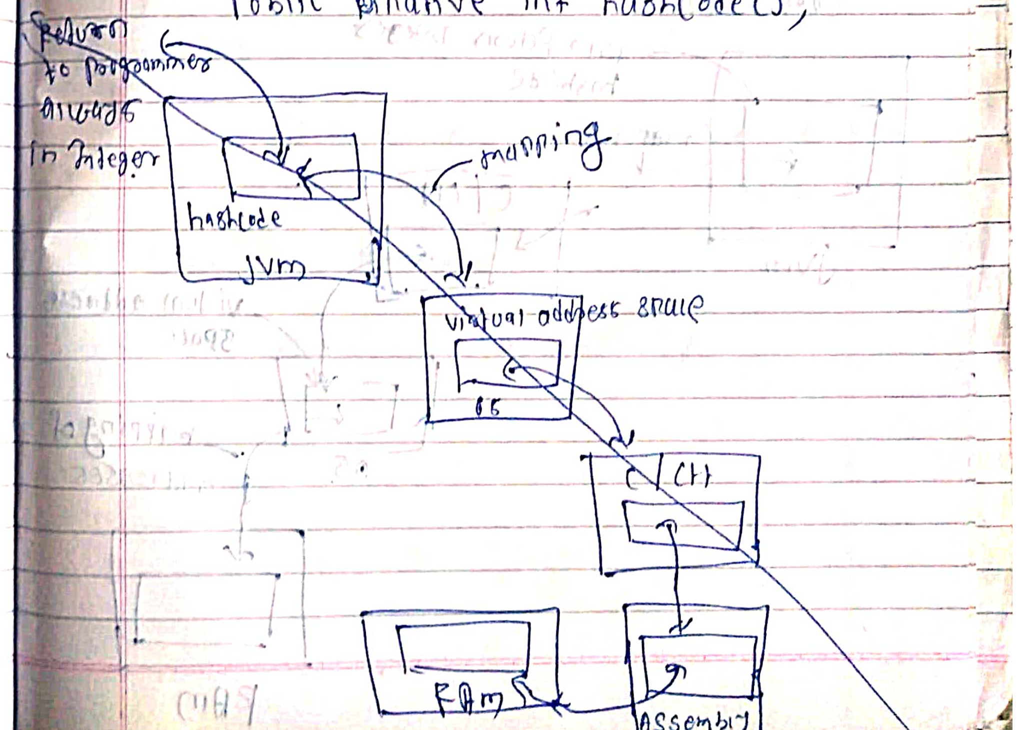
આ method એ native છે.

native એટલે C અને C++ નો code અને java અને C, C++ સાથેનું interaction કરાવવા.

આ method નું ~~...~~ implementation object class માં મળે નહીં. તેમાં ફક્ત declaration જ મળે છે.

~~Public native~~

Public native int hashCode();



① hashCode method આપણાબાં object નાં address print કરાવતી હતી.

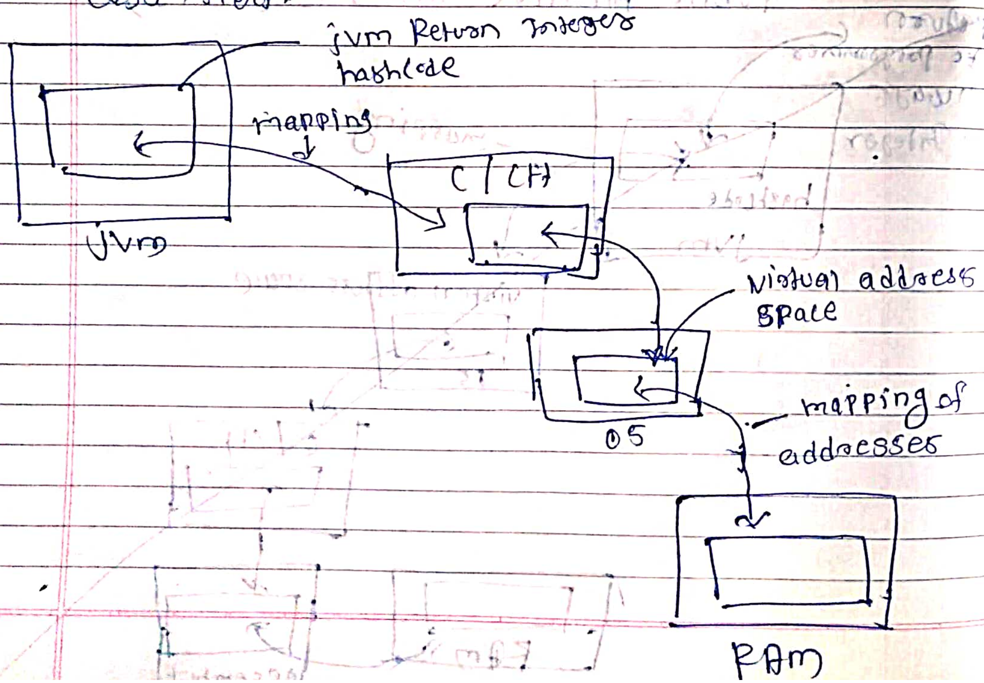
② પણ તો address હોતાની jvm મહેરે mapping કરેલા હોતી.

③ તો address આરાબ Integer ના સ્વરાત).

④ (અર object ના uniqueness findout કરવાના) કાંઈક ભર આપણો hashCode() method ના Call કરતો.

⑤ ભારે હોય multiple object ના hashCode same પણ આવું કરતાતો.

⑥ આપણો hashCode મહેરે આપણી unique Integer value પણ Return કરાવ શકતો. પણ uniqueness મહેનત પૂરી, તો method override કરવુ. આપણા Case ; મહેરે.



આપણાવાળા actual RAM વાળા address (આવી C++ મહેરે પણ જોતા નહીં આવી jvm ના Hardware પાણુના સુધવા લીધ છાંદે.

System Class

IdentityHashCode() — static method in System Class

Class Student

{

Public int hashCode() — unique value આપણો Return આપવા કરાવતો.

{

return Student.Roll_number;

}

જો jvm ની hashCode method Call કરાવતી હોય તો

public int hashCode()

{

return (super.hashCode());

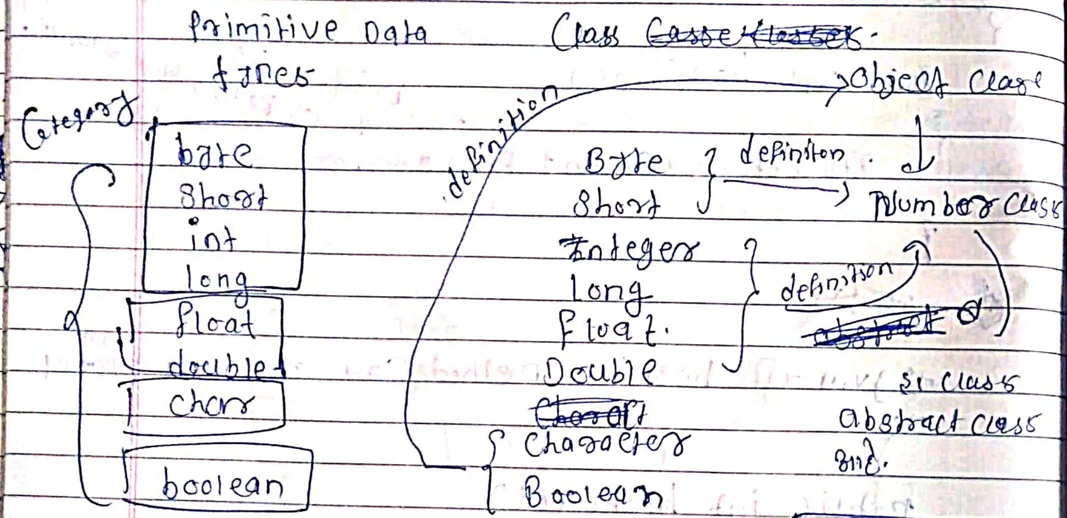
}

Wrapper Classes

Wrapper classes શા માટે આવ્યા.

આ Primitive data type ના conversion ને object type માં ~~convert~~ કરવા માટે wrapper classes આવ્યા છે.

Wrapper classes ને object collection માં use કરી શકાય.

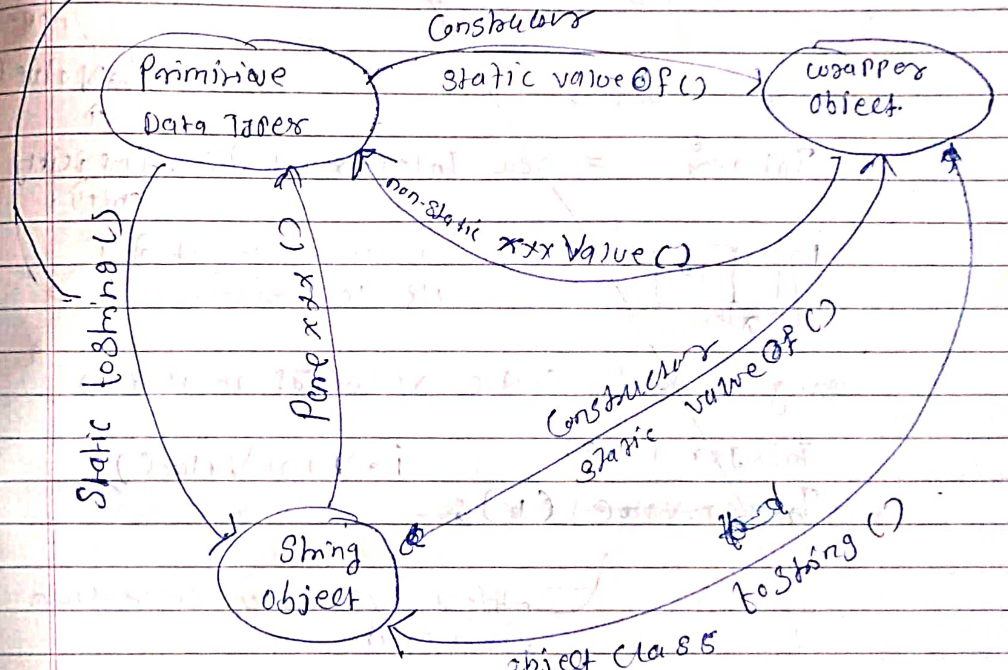


① Byte and Short class ના Number class માં method definition લખેલા છે.

② Integer, long, float, Double ~~class~~ ના classes ના method નામ abstract Number class માંથી. આપેલી class પણ abstract છે.

③ Character and Boolean class ના method with definition object class ના આપેલ લખેલા છે.

Integer, float ... આ class માં static override કરેલી નથી ની class ની જગ્યાએ parameter છે. ત Integer [illegible] છે.

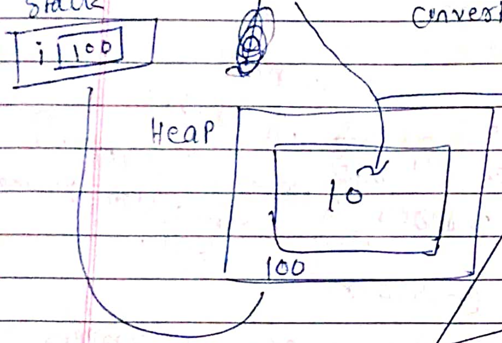


Byte Range - -128 to 127

Character માટે valueOf() method નથી.
8 Constructor 8 class માંથી
7 ના method છે.

① Primitive to wrapper object

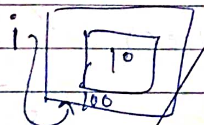
int a = 10; આવા conversion object type convert કરવાનાં.



integer variable ના conversion object માટે હોય.

Using Constructor

Integer i = new Integer(10); — implicitly call કરશે. intValue() (non-static)



i ના પર point કરે પર 10 ને રાખે.

Using method static valueOf method()

Integer i=10; — i.intValue();

~~Integer.valueOf(10);~~

object type માટે conversion થવાના માટે.

② object type to primitive conversion

non-static xxxValue()

Integer i = new Integer(10)

i.intValue();

---

① Primitive to wrapper object

Primitive Data type ના conversion object type માટે કરવાનાં છે. ten method આવી રીત Constructor દ્વારા થાય છે.

Using Constructor

Integer i = new Integer(10);

S.o.P(i)

Output = 10 — in object format.

Using static valueOf()

Integer i = Integer.valueOf(10);

~~S.o.P~~

System.out.println(i);

Output: 10 — in object format.

integer class માં આવી valueOf method છે. એની static મેથડ call કરીને આપણે convert કરવા પડે.

Integer i = 10; — JVM implicitly valueOf() ને call કરશે અને ત્યારે .valueOf(10); object type માટે conversion કરશે.

આવાને autoBoxing કહેવાય.

(e) Object to Primitive type

Using non-static xxxValue() method ના call કરવા દ્વારા તો object value specific primitive type માં convert કરવા દેવાશે.

```
Integer i = new Integer(10);
int a = i.intValue();
System.out.println(a);
```

Output : 10 - integer value.

~~int a = new Integer(10)~~

~~int a = new Integer.~~

~~Integer i = 10; — implicitly ત્રણ valueOf() method ના call થતાં હોય convert કરવા દેશે.~~

~~[.valueOf(10)] ← implicitly થશે. આને auto boxing કહેવાય.~~

int a = new Integer(10)[.intValue();]

↑ implicitly ત્રણ call થશે. તો method non-static છે. આને auto-unboxing કહેવાય.

(3) Primitive data to String object

Using static toString()

- toString method Object class નો હોય છે. તો Integer, Float, ~~Char, Boolean~~ & Long ના class માં override કરવામાં આવેલી છે. ત્યાર પછી તેને overload પણ કરી છે.

overloaded method

toString() - without parameter તો non-static છે.

overrided method

toString(___) - with parameter તો method static છે.

Primitive —static toString()→ String object.

```
String j = Integer.toString(10);
~~j = "100"~~
System.out.println(j);
```

Output : "100"

(4) String Object —static parseXxx()→ Primitive

~~String~~

```
int j = Integer.parseInt("10");
System.out.println(j);
```

j = 10

(5) String object —Constructor / valueOf()→ wrapper object

Using constructor

~~String i = "10";~~

```
String i = "10";
Integer j = new Integer(i);
```

integer type નો reference variable દ્વારા string object નું conversion wrapper object થશે.

Using method

~~String i = "10";~~

```
String i = "10";
Integer i = Integer.valueOf(i);
```

auto Boxing & auto unboxing

Primitive નું Conversion implicitly object type માં થાય.
Object નું Conversion Primitive માં conversion થાય. implicitly

Note

Function Calling માં જ્યારે formal parameter હોય ત્યારે actual parameter ને formal parameter માં copy થાય implicitly by using assignment operator

autoboxing

```
Integer i = 10;
```

.valueOf() jvm compile time પર કરાવે છે.

```
integer i = ~~new~~ Integer.valueOf(10);
```

```
void m1(Integer i)
```

.valueOf() - implicitly compiler કરશે

autoboxing

```
d.m1(10)
```

```
int a = ~~Int~~ new Integer(10).intValue();
```

implicitly jvm call કરશે intValue() method ને

object to primitive

auto unboxing

```
void m1(int i)
{
        .intValue();
}
```

d1.m1(d2)

100 → 10 (object નું address)

implicitly call થશે અને object નું conversion primitive માં થશે.

Demo d = new Demo();

System.out.println(d); implicitly .toString() ને call થશે.

Integer i = new Integer();
System.out.println(i); → .toString()

Integer Class ના ત્રણ toString ની method છે.

① without parameter

non-static toString() - primitive ને conversion string object માં કરવા માટે. એ override કરેલી છે.

② with parameter - એ overload કરેલી છે. ચાર class માં.

static toString(10) - string માં 10 print કરાવશે. એ static method છે.

⑥ Wrapper object --toString()--> String object

```
Integer i = 10;

String s = i.toString();
System.out.println(s);
```

Output : 10 — આ String object.

- Using field name function calling કરવા હોય તો આપણે function call કરવામાં reference variable ની જરૂર.
- String Class એ wrapper class નથી, કારણ string primitive type ના નથી, તે class type ના છે.
- String ના reference variable માં string object ના reference store કરીએ.

String એ primitive પણ નથી અને derived પણ નથી.

String class

import java.lang.String;

String એટલે sequence of character

String class

- String
- String Buffer
- String Builder

i) String ()

String class માં 8 constructor આવે.

(1) String ()

without parameterised constructor એટલે default constructor a synthesis constructor.

(i) String s = new String ()

new object તૈયાર થશે. Heap વાળી જગ્યા SCP વાળી પણ

(ii) String s = "India";

આવા વાળા SCP વાળી જગ્યા — reference variable બનાવે

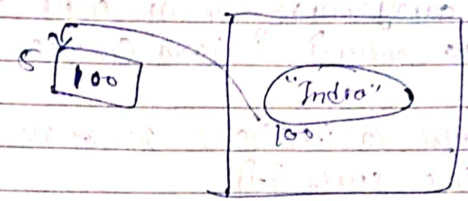
String literals કરવાથી કોઈ નવા object તૈયાર થતા નથી.

(2) String ("India");

String s = new String ("India");

~~parameterised~~ constructor call થશે.

parameterised constructor call થશે Heap માં India નામના object તૈયાર થશે. નામના એક Reference variable 's' stack વાળી બનશે.



(3) String (StringBuffer);

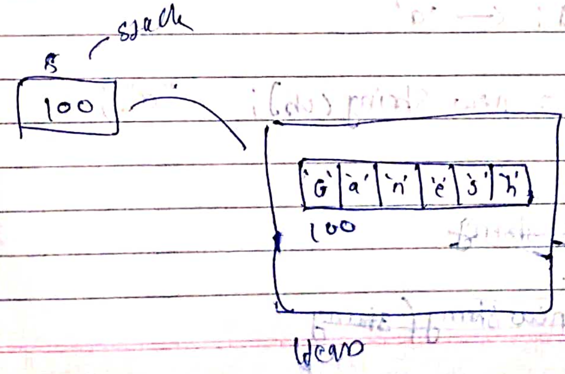
(4) String (StringBuilder);

(5) String (ch);

આપણે character ના array માટે String () constructor માં આપણે તો array pass કરી શકાય.

char ch[] = {'G', 'a', 'n', 'e', 's', 'h'};

String s = new String (ch);



java માંહી static memory allocation કેમ નથી
નથી ત્યારે ફક્ત Dynamic કામ કરે છે.

~~static array નું~~

array ડીક્લેર કરતા કેટલી કેટલી Byte allocate ~~કરાશે~~ કરાશે તે java compiler કરતા નથી.

~~static~~ (arr[int size]); આમ જ જો લખીએ તો compile time error આવે.

આમ static allocation થતું નથી.

⑥ String(ch, offset, count)

- ch → character array
- offset → ક્યાંથી start કરાવવા
- count → કેટલા character

string માંથી convert કરાવવા તે આપવા લાગશે.

String s = new String(ch, 2, 5);

⑦ char ch = a; ← 'a'

String s = new String(ch);   "a";

⑧ ~~Unicode ની string~~

~~String s = new String(string~~

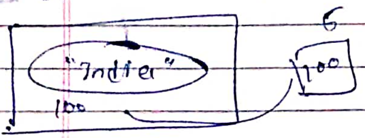
String s = new String(,);

- first argument: string (short b) — Unicode value (-128 to 127) થી 255
- (byte[], int offset, int count)

~~NullPointer Exception~~

Object

String s = new String ("India");

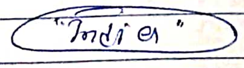


Object ક્યારે બને ત્યારે તે Object ના Heap Area માં અને SCP Area માં બન્ને જગ્યાએ બને છે.

Object કામ પૂરું થયા બાદ Heap Area ની memory destroy થઈ જાય.

જો object ની અંદર value અલગ આવે તો Heap Area માં અલગ object બને છે ને ત્યાંની memory અલગ હોય છે. તે એક reference variable દ્વારા object ને point કરી શકાય નહીં.

SCP (String Constant Pool)



Object ક્યારે બનાવાય છે ત્યારે reference variable બને છે.

આ memory Heap Memory માં object ની value બદલાય ત્યારે reference variable બદલાય નહીં.

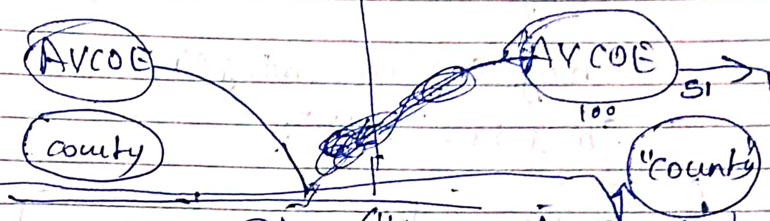
String s = "Ganesh";

- આ statement execute થયા બાદ object SCP Area માં બને છે, Heap માં નહીં બને.

- આ object ના multiple reference variable point કરી શકાય છે ત્યારે value સરખી હોય તો.

Heap

(SCP)



String s = new String ("AVCOE");

String s = "Country";

String S1 = "AVCOE";

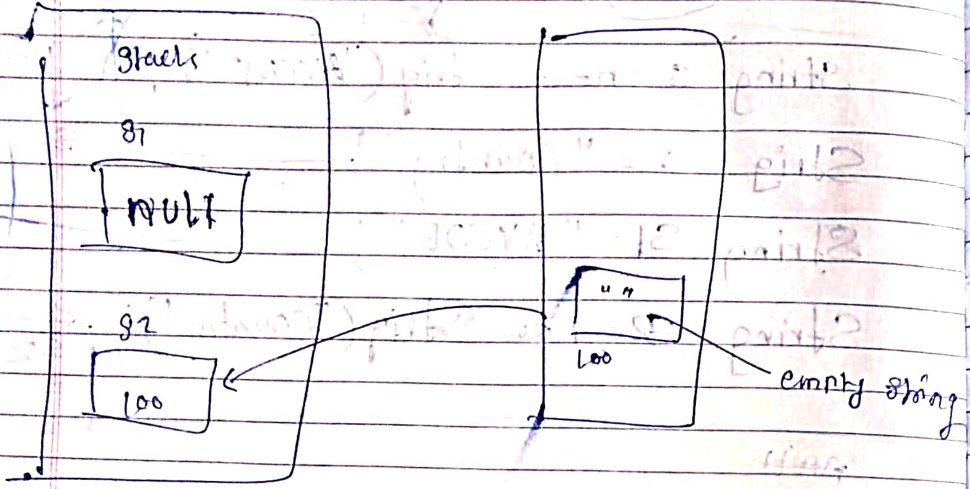
String S2 = new String ("Country");

* Null Pointer Exception

String s1;

String s2 = new String();

String s3 = "India";



s1.equals(s2) – Null Pointer Exception દેખાશે

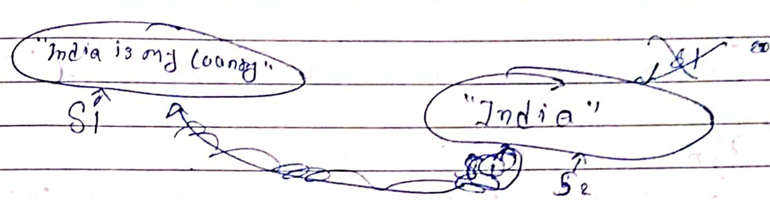
s2.equals(s3) – Compare કરશે.

String ~~imm~~ immutable કેમ કહે છે?

~~String s = new~~

String s1 = "India";
String s2 = "India";

SCP (String ~~Static~~ Constant Pool)



જ્યારે આપણે s1 String બનાવી અને વેલ્યુ ઉમેરીએ છીએ

s1 જે object ને point કરતો હતો તેમાં કોઈ change કરી શકતા નથી તેમાં કોઈ changes થતા નથી.

તેના બદલે નવો reference variable point કરતો હોય છે.

s1.concat("is my country");

આ કરવા માટે s1 ~~refer~~ થતી નથી એ new Object બનાવે છે. જેમાં s1 Reference variable નવા object ને Point કરશે.

existing object ~~માં~~ બદલાતો નથી ના change કરતા નવીન Object બનાવે ~~છે~~ તેથી આને immutable ~~String~~ String કહેવાય.

StringBuffer — Java 1.0 ver.

import java.lang.StringBuffer

StringBuffer મતલબ Buffer કે જેમાં Buffer ના નામ થી
StringBuffer કે mutable છે. તેથી જો કે
જે Reference variable બનાવેલ object ના
Print કરતા ... object ની value
change કરીએ તો તે value માં change થશે.

StringBuffer માટે હંમેશા object ના આધાર કરવા લાગશે

StringBuffer sb = new StringBuffer("India");

StringBuffer sb = "India";
કે આપણા લાઈ કારણ આપણા પ્રકારની ... જોઈએ
નહીં. Heap memory વગરની ... જોઈએ.

Heap memory Clean કરવાનું કામ Garbage Collector કરશે
તો 0.06 micro second ... specific time ...
... જો object ... ખાલી ...
જગ્યા clean કરવાનું કામ કરશે.
જેથી તેના 3 section માટે dividation કરેલ છે.

1) young generation

2) Old generation

3) Perm generation (Permanent generation)

String class

import java.lang.String;

String means - sequence of character

String Class

1) String
2) String Buffer
3) String Builder

1) String -

String Class :-

String એટલે sequence of character

Java માટે character ની unicode value store થશે

જ્યારે અહીં string for loop લગાવી print કરવા લાગીએ.
but in Java જ્યારે string ના reference variable
Print કરીએ તો full string print થશે.

Java ના data ને dynamically store કરે so Java
is dynamic Programming language.

dynamic એટલે data stored કરવાની જરૂર નથી
જ્યારે new થી object બનાવીને store
કરવા લાગશે. આથી તો object heap માં જોવા
મળશે.

impor

String એ primitive data types નથી.

કારણ, Primitive data type ની size ફિક્સ હોય છે.

e.g-

int - 4 byte
float - 4 byte
long - 8 byte
double - 8 byte
String - not a fix of its byte
char - 2 byte

String ની size નક્કી નથી તો dynamically store થાય.

Primitive variable ની static memory allocation

```
Demo d;
d.fun(); — Null Pointer Exception
```

In String class have 8 Constructor

① String s = new String( );
   ↓
   [" "] ← null, Empty String
   ખાલી object બનાવે છે.

② String s = new String("India");

③ String s = new String(sb); ← StringBuffer નો object String નો constructor માં pass કરો

④ String s = new String(sb); ← StringBuilder નો object String ના Constructor માં pass કરવો.

⑤ char ch[] = {'I', 'n'};

String s = new String(ch[]);

⑥ char ch='a';

String s = new String(ch);

⑦ subString

India is my country — e.g. 5 starting, 6 no of character

String s = new String(ch[], int offset, int count)

"_is_my"

⑧ unicode ની string

String s = new String(byte[], int offset, int count)

unicode values = 256

short ની one (1) byte -127 to 128

Reference loosly coupled:

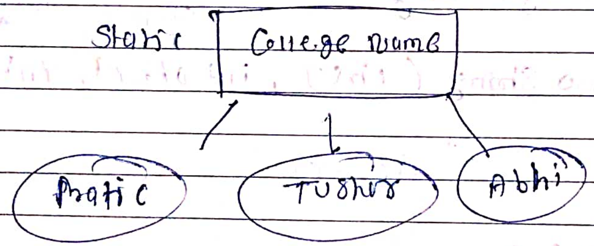
જેટલા આપણા String બનાવ ત્યારે જેટલા તે SCP માં check કરશે કે String છે કે નહિ, જો હશે તો s2 ના ત્યાંથી reference લઈ લેશે.

જેટલા આપણા

s3 = s3.concate(" is my Country")

જેટલા આપણા String માં change કરાવીએ ત્યારે તેટલા ત્યાં String માં change નહિ થાય. ત્યાંથી નવો object બનાવી દેશે. આપણે ત્યાં જે changes કર્યા હશે તે આપણા reference variable ત્યાં નવા object ને point કરશે.

static, final, immutable



Common for all Student

Pratic.CollegeName = "Sharmila" કરવા પર

static value ના change કરી શકો પણ immutable માં change નહિ કરી શકો.

String buffer

String buffer Java version 1.0 માં આવ્યા

String buffer mutable છે.

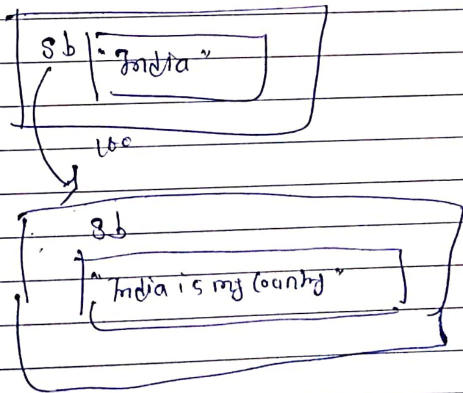
String buffer માં બધા function synchronized છે. તેથી performance slow છે.

~~1.5 માં StringBuffer આવ્યા ના unsynchronized છે.~~

String Builder

String builder એ Java 1.5 માં આવ્યા. તેના બધા method asynchronized છે. multiple thread એક સાથે કામ કરી શકે. તેથી performance fast થયેલી છે.

```
StringBuffer sb = new StringBuffer("India");
    sb.append(" is my Country");
```



1. append()
2. insert(index, "character");
3. delete
4. deleteCharAt(index)
5. Reverse();

object બની ગયા પછી તેમાં changes કરવા હશે તો String Buffer and String Builder

StringBuffer – Java 1.0 version

import java.lang.StringBuffer

StringBuffer म्हणजे Buffer हे एक प्रकारचे Buffer चे नाव आहे.

StringBuffer हे mutable आहे. म्हणजे जर का एक वेळा Reference variable ने एका object ला point करायला सांगितले तर त्या object ची value change झाली तर ती त्याच्यामध्ये change होते.

StringBuffer मध्ये आपण object ला तयार करावा लागतो.

✓ StringBuffer sb = new StringBuffer("India");

✗ StringBuffer sb = "India";

हे चालत नाही कारण आपण SCP मध्ये जागा घेत नाही. Heap memory मध्ये जागा घेते.

Heap memory clean करायचे काम Garbage Collector करतो

तो 0.06 microsecond या specific time नंतर होऊन जे object तयार झालेले पडले आहेत. त्याला clean करायचे काम करतो.

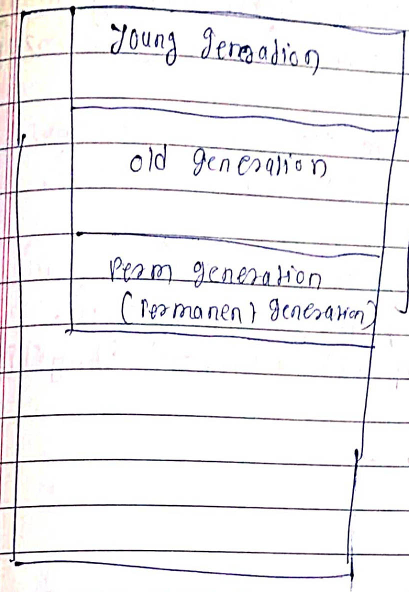
त्याचे पण 3 section मध्ये dividation केलेले आहे.

1] Young generation

2] Old generation

3] Perm generation (Permanent generation)

Heap



या तीन generation heap memory मध्ये आहे.

Garbage Collector object destroy करताना generation च्या आधारे करतो.

Young generation:

Young generation मध्ये नवीन तयार झालेले object असतात. ते थोड्या वेळासाठी memory मध्ये राहतात होऊन destroy होतात.

Old generation:

जे object खूप वेळापासून या memory मध्ये टिकून आहेत ते या old generation मध्ये असतात.

Perm generation

जे object permanent ठेवायचे आहेत ते object Permanent generation मध्ये असतात. जे object Ram मध्ये process संपली की destroy होऊन जातात.

java મધ્યે આપણા ઘણા memory allocation થયા હશે તેને destroy કરવાનું કામ jvm કરશે.

String Buffer એ class ના ઉપયોગ થી mutable String objects તૈયાર કરવામાં આવે છે.

String Buffer માં object મધ્યે આપેલી string ની પાછળ નવીન String append કરશે. append() method સાથે StringBuffer વાપરવાની રહેશે.

3] String Builder

StringBuilder એ mutable ના String તૈયાર કરશે.

~~StringBuffer એક time માં 16 character store કરશે પણ 17 મા character આવતા નથી~~

16 x 2 = 32 character સાથી બીજા તૈયાર કરીને 17 મા character store કરશે.

StringBuilder બધા object Heap memory વાપરીને તૈયાર કરવામાં આવશે.

String functions:

① length()

length() function આપણા દરેક string મધ્યે કેટલી character આવે તે ગણીને તેમાંથી કેટલા આવ્યા તે ગણતરી character એટલી value integer માં ગણી આપે તે Return કરશે. તેનો રીટર્ન Return type int છે.

String s1 = "India";

eg: int a = s1.length()
System.out.println(a);

② toString()

toString() આ method String class મધ્યે override કરેલી છે. કારણ કે તેના class ના parent object class છે.
to String method આપણને આપેલી string object તરીકે return કરશે.

String s = "Ganesh";

System.out.println(s.toString());

toString() method object class મધ્યે String class મધ્યે override કરેલી છે.

③ CharAt()

CharAt() function ~~index~~ પોતાના index વાળી જગ્યા character શોધે ને character return કરે.

હેઠળ રહેલી return type char છે.

S1.charAt(a);

④ Concat()

પહેલી string અને બીજી string ભેગી કરવા માટે concat function વાપરો append ~~કરો~~ કરવામાં

S1.concat(S2);

⑤ CompareTo()

બે string compare કરવા માટે કામ આવે આની return type boolean છે એટલે કે true અથવા false

⑥ CompareToIgnoreCase()

બે string compare કરીને જો return true હોય ~~તો~~ અથવા false
- આમાં capitalize case ~~તો~~ અને small case matter કરતા નથી.

⑦ trim()

~~ન~~ String ની starting અને ending ની spaces remove ~~કરો~~ કરવા ~~માટે~~ string ની વચ્ચે ના string માંથી space remove થતી નથી.

⑧ ~~R~~ replace()

પોતાના position ના character ને replace કરવા માટે ~~શ~~ જૂના character ને નવી character વડે replace.

⑨ equalsIgnoreCase()

Capital અને small ~~ન~~ case ignore કરે છે...

⑩ ~~A~~ indexOf()

Index ~~ન~~ character point કરવા. First occurrence ~~ન~~ થી index પહેલાના માંથી ને return કરે.

~~int f~~

public int ~~ન~~ indexOf(char c)

{

return 1;

~~3~~ else

} ~~return -1~~ return -1;

} ~~ન~~ મળ્યા નહીં તો

(11) Contains ('A');

true or false return કરશે

(12) ① SubString ('Start Character')

SubString ('my');

② SubString (Start index , end Index)

e.g. 0 , 9

0 to 8 index સુધીના character આપશે.

(13) Split()

The split() ~~function~~ method divides the string at the specified regex and returns an array of substrings.

Array

Array:-

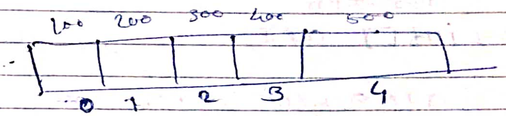
definition:-

① Array is a collection of homogeneous element under in one variable

② Array is a collection of homogeneous variable under one variable in sequential order

Java માં array C અને C++ કરતા [illegible] array ના object બનાવવા [illegible] reference ના આવે.

int a[5];



a [100]

↑

અહીં array ના reference હોય છે.

Note:-

Data member ની default value zero આવે.
Reference variable માટે null આવે.
local variable માટે default નથી.

Java માટે array ને declare કરવામાં આવે [illegible] definition [illegible].

Java માટે array define કરવાના [illegible] types [illegible].

① Primitive array — derived type છે
② non-primitive અથવા Reference array — User define types ના હોય છે.

① Primitive array

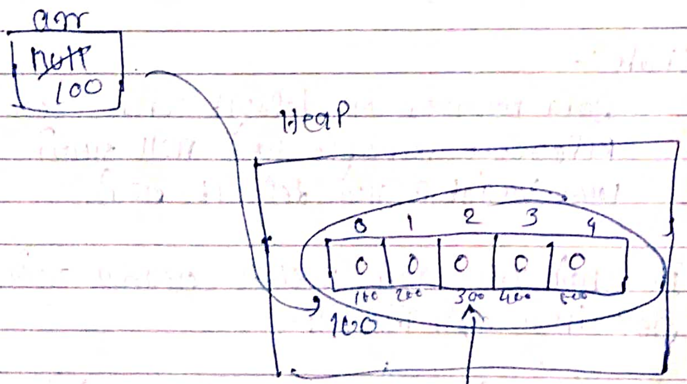
int[] arr;
int arr[];
int [] arr;
} 3 way to declare array

~~int arr~~

① way int[] arr;

arr = new int[5]; — 5 elements

Dynamic memory allocation



length [5]

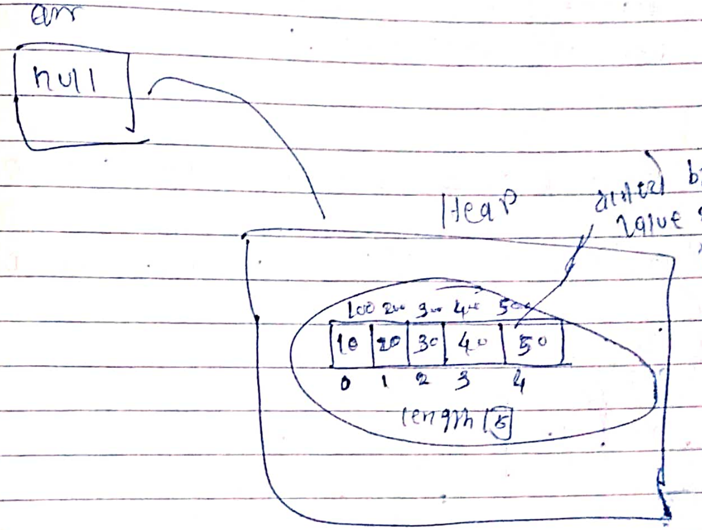
array ના memory કેટલા length variable ...
... element ...

② way int[] arr = {10, 20, 30, 40, 50};

ઉપર ની line ~~[illegible]~~ Replace કરી શકો.

int [] arr = new int[]{10, 20, 30, 40, 50};

stack



આખરે by default value zero આવશે.

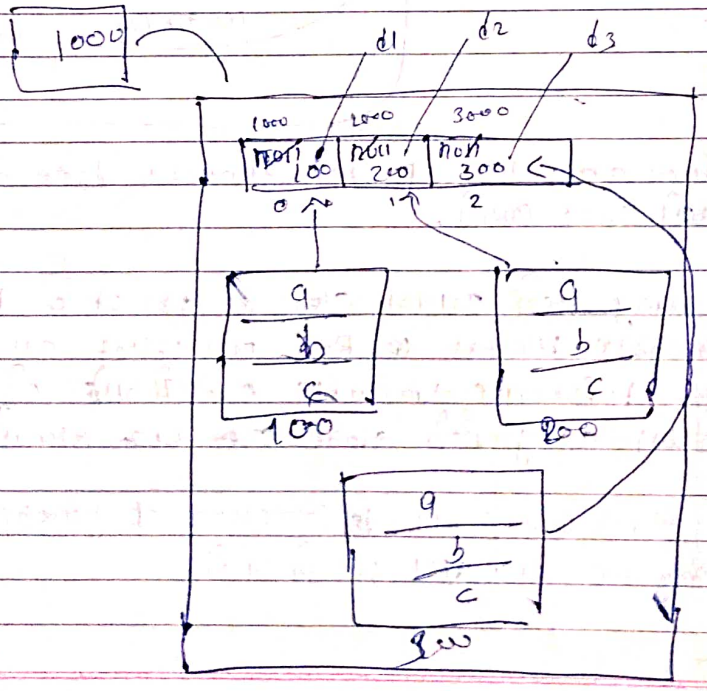
આપણે array માં homogeneous type ના variable જ store કરશો.

array માં આપણે class ના object ના Reference variable રાખ્યા છે ... ના object ના address ના Reference variable માં ... ના variable આપણે array માં store કરી શકાય.

મતલબ array એ collection of homogeneous variable છે એમ કહી શકાય.

```
Class Demo
{
  int a;
  int b;
  int c;
}
Demo d1 = new Demo();
Demo d2 = new Demo();
Demo d3 = new Demo();

Demo[] arr = { Demo d1 = new Demo(), Demo d2 =
                    new Demo(),
// arr = new Demo[3];      Demo d3 = new Demo() };
```

Stack



Orchids

Orchids are easily distinguished from other plants, as they share some very evident derived characteristics or synapomorphies. Among these are bilateral symmetry of the flower (zygomorphism), many resupinate flowers, a nearly always highly modified petal (labellum), fused stamens and carpels, and extremely small seeds. Along with the Asteraceae, they are one of the two largest families of flowering plants. The Orchidaceae have about 28,000 currently accepted species, distributed in about 763 genera. The determination of which family is larger is still under debate, because verified data on the members of such enormous families are continually in flux. Regardless, the number of orchid species is nearly equal to the number of bony fishes, more than twice the number of bird species, and about four times the number of mammal species. The family encompasses about 6–11% of all seed plants. It also includes Vanilla (the genus of the vanilla plant), the type genus Orchis, and many commonly cultivated plants such as Phalaenopsis and Cattleya. The root caps of orchids are smooth and white.

S.No.	Date	Title	Page No.	Teacher's Sign/Remarks

Array Declaration

int[] a;

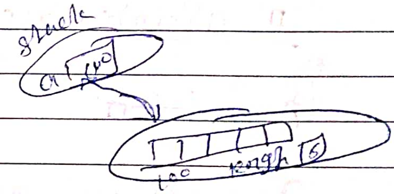
↑ value દિધેલી ચાલતા નથી નહિતર
Compile time error

કારણ,
java માં static memory allocation
નથી. બધી memory Dynamically
તૈયાર કરી જાય. using new keyword

array એ java ના static memory ~~ઉપયોગ~~
કરીને ~~@~~ allocation કરી શકતા નથી. એ java
ના Reference variable તરીકે કામ ~~કરતા~~ કરતો.

~~~~

Array definition



int[] a = new int[5];

① -7 પર value દાખલ તો Compile time
મા કોઈ પણ error હોતા નથી. પણ
Run time મા NegativeArraySizeExce
ption

~~error~~ હોય.
કારણ array ની size એ negative
માં હોઈ શકતા નહિ.

② 'A' ['A'] પણ ચાલશે તો જ્યારે
integer value માં Conversion
થયા હોય છે using unicode system
~~~~

int[] a = new int[0];

(a) 0 value રાખવાથી a Heap memory માં 0 byte memory allocate થશે. a ના Object ના address a માં રાખશે.
a. zero byte size ની allocate કરાશે.

→ જ્યારે Reference variable માં null રાખશી ત્યારે address (hashcode) દેખાશે નહિ ત્યારે address માં કોઈપણ value નથી હોતી. જ્યારે આ reference variable ના point કરશું NullPointerException આવશે નહીં.
Using Reference variable દ્વારા function ના call કરી શકાશે. a array ની length check કરી શકાશે.

~~Definition~~

Declaration and Assignment

int[] a; — Declaration

a = new int[5]; — Definition

a[0] = 10; — Assignment

Declaration + Definition + Assignment

int[] a = new int[5] = {10, 20, 30, 40, 50};

length Variable

length variable array ના object બનાવે ત્યારે જ તેના object માં JVM દ્વારા આપવામાં આવે છે. length variable final છે. તેથી આ variable માં changes કરી શકાતા નથી. કારણ final માં one time value દેવામાં આવે પછી બદલાતી નથી.

* Array ના Reference type

int[] a = new int[5];
reference type

Object
- byte[], short[], int[], long[], float[], double[], character[], Boolean[]
- Object[] ← Demo[] (implicitly type cast કરશે)

int, float ... Array Class બધાના Parent object Class છે.

જ્યારે user define કરેલા array ત્યારે Object Array Class ના under આવશે.

ઉદા.

Object = new Demo[5]; ✗

Object[] = new Demo[5]; ✓

Parent ના Reference, Child ના object

2-D Array

① int[] a;
② int []a; } 1-D Array
③ int a[];

④ int[] []a, b;

'a' & 'b' are both are two Dimension array.

⑤ int[] a, b; // ← int []a, b;

1-D array

⑥ int a[], b;

int a[] → 1-D array, b → variable

⑦ int[][] a, b;

2-D Array

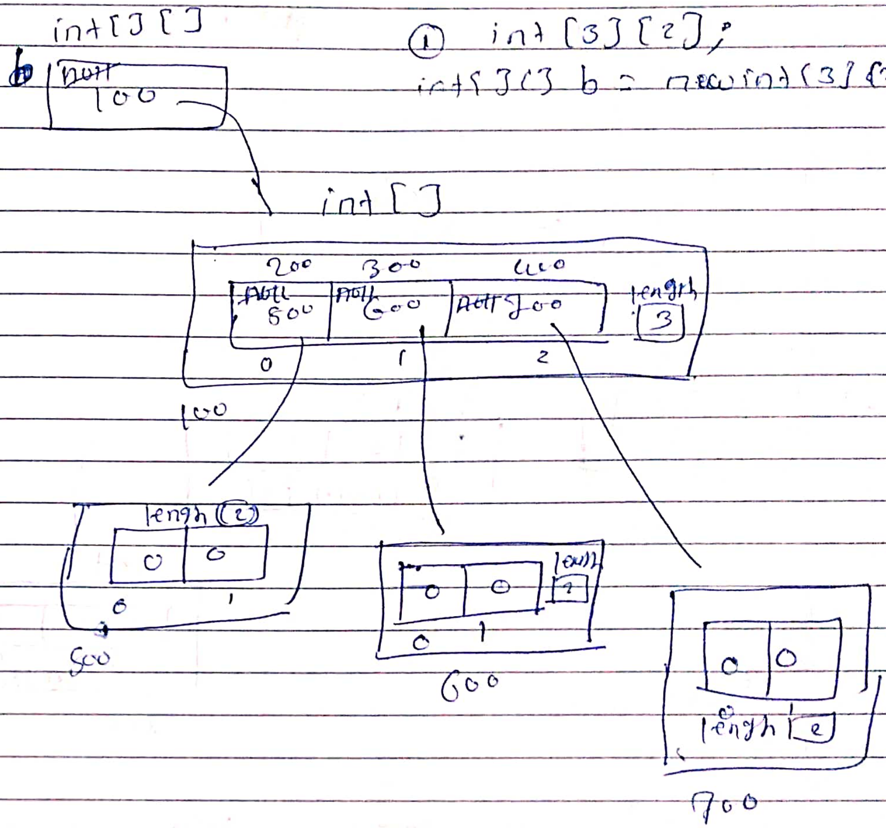
⑧ int[] a []b;

int[] a → 1-D Array, []b → 2-D Array

2-D Array Representation

① int [3][2];

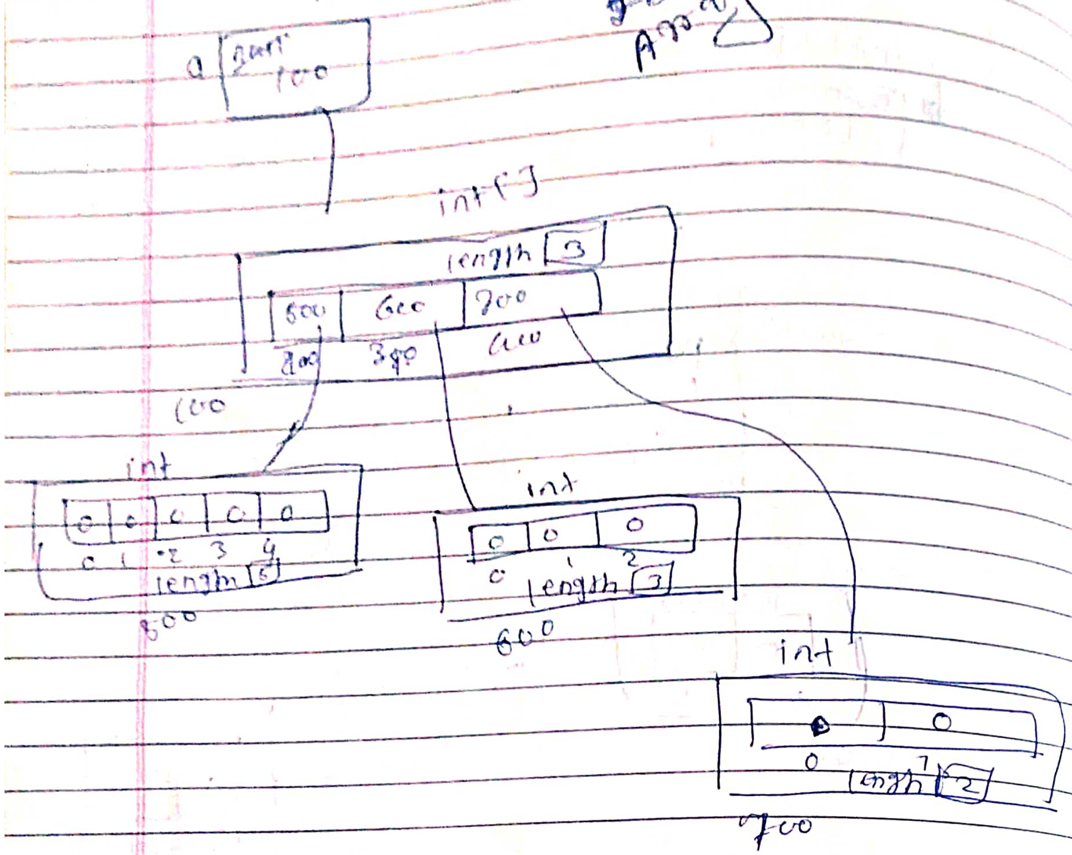
int[][] b = new int[3][2];



② int [3][]; How many element you required [illegible] object [illegible] second box [illegible] value [illegible]

a[0] = new int[5];
a[1] = new int[8];
a[2] = new int[2];

Jagged Array



java માટે 2-D array matrix form હોવો જરૂરી નથી કારણ કે hierarchy આપણે મનસ્વી રીતે કરી શકીએ. તે depend છે આપણે કઈ રીતે object બનાવીએ.

jagged array no of column fix હોતા નથી અને row ની value change કરી શકાય.

int[] a = new int[5];

Rules

Declaration વખતે square bracket ની અંદર value ન આપી શકાય. otherwise compile time error

Definition

[5] ✓

[10L] — X

[ ] byte, short, character, int

[-7] X (-7 × 4 = -28)

Program compile થશે. પણ Runtime માં negative Array Size Exception

[0] ✓ (0×4 = 0)

int a[a]

└ આ constant નથી. મેમરી ફાળવણી કરતી વખતે non-constant value આપી શકાય. Runtime માં.

java ના Arrays class utility package ના under રહેલા છે

import java.util.Arrays;

└ દરેક method static છે.

① toString

System.out.println(java.util.Arrays.toString(a));

(java.util.Arrays.sort(a));

toString method override int[][] class માટે કરેલ હોતી નથી કારણ કે predefine class છે તેથી array using for loop print કરાવવી.

Exception handling

import java.lang.Throwable

- Exception
- Error

Compile time error અને રન ટાઈમ.

In program have syntax fault then Compiler not Compile Code directly throw the error message. This error is called Compile time error

Compile time માં જો કોઈ error remain હોય તો. ત્યાં સુધી file Compile થઈને class file તૈયાર થતી નથી.

Runtime Error.

જે error Compiler throw કરતા નથી પણ Run time વખતે system Resources ના કારણે ઉત્પન્ન Runtime error Compiler throw કરતો.

આ Runtime error Program Run થતા સમયે handle કરી શકાતા નથી. eg- Stackoverflow

eg -

Stack overflow

આ પ્રકારની error ને જ્યારે આપણે ફરજિયાત પણે બનતા રોકી શકતા નથી અને error થઈ જાય ત્યારે પણ કોઈ પણ handle કરતા હોતા નથી. પરંતુ આ Program normal termination વગર abnormal terminate થતો.

Exception handling — જ્યારે error આવતા રોકી શકાય નહીં ત્યારે handle કરવામાં આવે છે.

જ્યારે આપણા Program જે Point પર normal termination વગર abnormal termination કરે તે પહેલા જે Point પર અટકે છે. તે Point ના Exception handling કહેવાય.

આવી Exception run time માં Remain હોય નહીં. જેથી તેને handle કરવી પડે. આ chapter નું નામ exception handling છે.

Exception અને Error Class ના Parent Throwable class છે: અને Throwable ના Parent Object Class છે.

Object
↑
import java.lang.Throwable

- Exeption
- Error

eg -

જ્યારે આપણે number ને divide by zero કરીએ ત્યારે Exception આવે છે. તેને handle કરવી પડે.

ArithmaticException → divide by zero ની exception છે.

Compile time Exception — Runtime Exception

~~Error~~ Check Exception — Uncheck Exception

Check Exception
Compiletime Exception

eg-

આઈ - બા મુજબ

2 Pen

જેટલા મુજબ paper લખવાની છેલ્લા આઈ હોય પેન હોય જ્યારે પર પેપર સમય time મા pen બંધ પડયા પર દુસરા વાપરવા.

o જે કેટલા Runtime મા પહેલાથી exception થવાના ચાન્સ હોય તો તેને compile time માં જ handle કરવી જરૂરી. આ Exception ને Check exception કહેવાય.

Uncheck Exception
Runtime exception.

આ ~~અનચેક~~ શાળાઓ પર paper વાળુ અસ્માન પડવા થી પડતા નથી.

o જે કોઈ Runtime મા ઉભા થતા exception હોય તો તે Compile time મા Compiler ને કહેતી નથી તો આપણા ચલાવી લઈએ.

Runtime મા કોઈપણ exception ઊભી થાય તો જે Compiler ના normal termination ~~થાય~~ ના

flow abnormal termination ના લીધે કેવી રીતે આપણે Exception હેન્ડલ કરી શકીએ તે જોઈએ.

Compile time Exception

Exception ને Compile time મા ચેક થયેલ ~~compile~~ runtime મા ઉભા થાય તો તેને compile time મા handle કરવી જરૂરી. ~~તેને~~ Compile time Exception કહેવાય. તેને Partial check exception કહેવાય.

eg-

આઈ — OS

પૈસા — Program (JVM)

2 Pen

1 પેન બંધ પડતા પર દુસરા પેન વાપરવા.

Technical example

(1) ArithmeticException

```
int a=10;
for(i=-10; i<=10; i++)
{
    System.out.println(a/i);
}
```

i=0 then Compiler throw the ArithmeticException

so we need to not divide by zero તેમ આપણા ચલાવી જરૂરી developer ના કરવી જોઈએ.

② IndexOutOfBound

↓

ArrayIndexOutOfBound | StringIndexOutOfBound.

~~access element by~~ not access element from valid index then compiler throws ~~the~~ this ArrayIndex-OutOfBound exception

not access valid String from String array at the time of Strings so their compiler throw the ~~compile time error~~ Compile time Exception at run time StringIndexOutOfBound

③ File not found

પર આપણા દ્વારા variable ની value મળ્યા variable માં રેહવે રેહવા ... તે આ variable ની value પર આ file માં dump કરવા ~~...~~ આવેલ. આણે કોઈ value dump કરવા આવે તો file database માં dump થશે.

હવે પર તો file delete કરવી આવે પરંતુ પર value દાખલ કરવા પ્રયત્ન કરતા પર file not found Run time Exception થશે.

eg- medical shop Bill

Person 1 → generate 5 Bill and Put into 5 value ~~...~~ into file

then Person 1 brother delete this file

Person 2 → generate new Bill but there generate error FileNotFound Exception

[illegible]

Java.lang.Throwable

- Exception — Partial Check Exception
  - ArithmaticException — Check exception
  - IndexOutOfBound
    - ArrayIndexOutOfBound
    - StringIndexOutOfBound
  - NullPointer Exception
  - File not found
  - EndOfFile
- Class inherit from exception ~~Runtimable~~ - આ Class માંથી uncheck Exception. આવેલ ~~...~~. આવેલ.
- Error — Partial Check
  - Assertion Error
  - Virtual machine Error
  - Stack overflow

④ **EndOfFile** – Checked Exception

જ્યારે File Read કરવામાં end આવી જાય ત્યારે Compile Runtime Exception આવતી.

⑤ null PointerException – Check Exception

જ્યારે Reference variable જ્યારે null હોય ત્યારે check કર્યા વગર પકડવામાં આવે ત્યારે null Pointer Exception ઉભો થાય છે.

Object
↓
java.lang.Throwable
↓ (Partial check Exception)

- Exception
  - Check Exception
    - Compile time પર Compiler Exception Check કરતી હોય error દેતો.
    - eg. NegativeArraySizeException
- Error
  - Assertion Exception
  - Virtual machine Exception
  - StackOverflow

Uncheck Exception

જે Exception Runtime વખતે પણ આવી શકે પરંતુ Compile time પર ચકાસણી કરવામાં આવતી નથી.

eg.

- ArithmeticException
- IndexOutOfBoundException
  - ArrayIndexOutOfBound
  - StringIndexOutOfBound
- null Pointer Exception
- EndOfFile

Compile time Exception

Compile code successfully હોવા છતાં Runtime પર ex. data type mismatch થાય ત્યારે exception હોય પણ તેની ચકાસણી compile time પર કરવામાં આવે છે તેથી Compile Time Exception કહેવાય.

Exception Practicals (Uncheck Exception)

1] Arithmetic Exception

~~for(int i:~~

Class Demo

{

Public Static void main(String args[])

{

int a=10;

for(int i=-10; i<=10; i++)

{

a=a/i;

System.out.println(a);

}

}

}

O/P:

Exception ~~ArithmeticException~~

java.lang.ArithmeticException / by zero

2] Array out of Bound Exception

Public Static void main(String args[])

{

int[]a=new int[5];

for(i=0; i<=5; i++);

{ System.out.println(a[i]);

Exception java.lang.ArrayIndexOutOfBound:5

3] ~~StringAoo~~

4] ~~negative~~

3] StringArrayOutOfBoundException

Public static void main(String args[])

{

String s = new String("sangamner");

System.out.println(s.charAt(15));

}

Exception StringArrayOutOfBoundException:

4] NegativeArraySizeException

Public Static void main(String args[])

{

int[] a = new int[-5];

}

Exception NegativeArraySizeException

~~Class.forName("Test");~~

~~જે આપણાને પ્રોગ્રામ Class ના~~

Check Exception

```
Class Demo
{
    int a;
    void add()
    { }

    Public static void main(String args[]).
    {
        Demo d;          // Demo d = null;
        d.add();
    }
}
```

error : of compile time
must initialize local variable

```
Class Demo
{
    static Demo d;      -- null value in d
    void add()
    { }

    Public static void main (String args[])
    {
        d.add();
    }
}
```

Exception NullPointerException

★ Class.forName("className")

જે આપણાને પ્રોગ્રામ class ના object બનાવ ન કરવા, કે extends ન કરવા આપણે main method માં લખવા. જો તો class load કરવા માટે તે જ્યારે static block execute કરાવવા માટે ત્યારે તે કરવામાં આવે.

Class નામવાળા class ના under forName() method આવે છે static હોય છે. તેમાં as a string parameter તરીકે class નું નામ પાસ થાય.

forName() method આપણાને class load કરાવે છે. અને JVM static Block execute કરાવે છે.

```
Class Test
{
    static
    {
        System.out.println("Test Class static Block");
    }
}
Class Demo
{
    Public static void main (String args[])
    {
        Class.forName("Test");
    }
}
```

output :- Test Class static Block.

* Exception class એ java.lang. package ના under છે.
Exception class એ બધા Exception class ના Parent class છે. Exception class બધા package માં આવે તેથી inherit કરેલો package માં આવે છે.
class જુદા-જુદા package માં આવે છે.

IOException class એ java.io. ના package ના under છે. આથી તે Exception class માંથી inherit કરેલા છે.

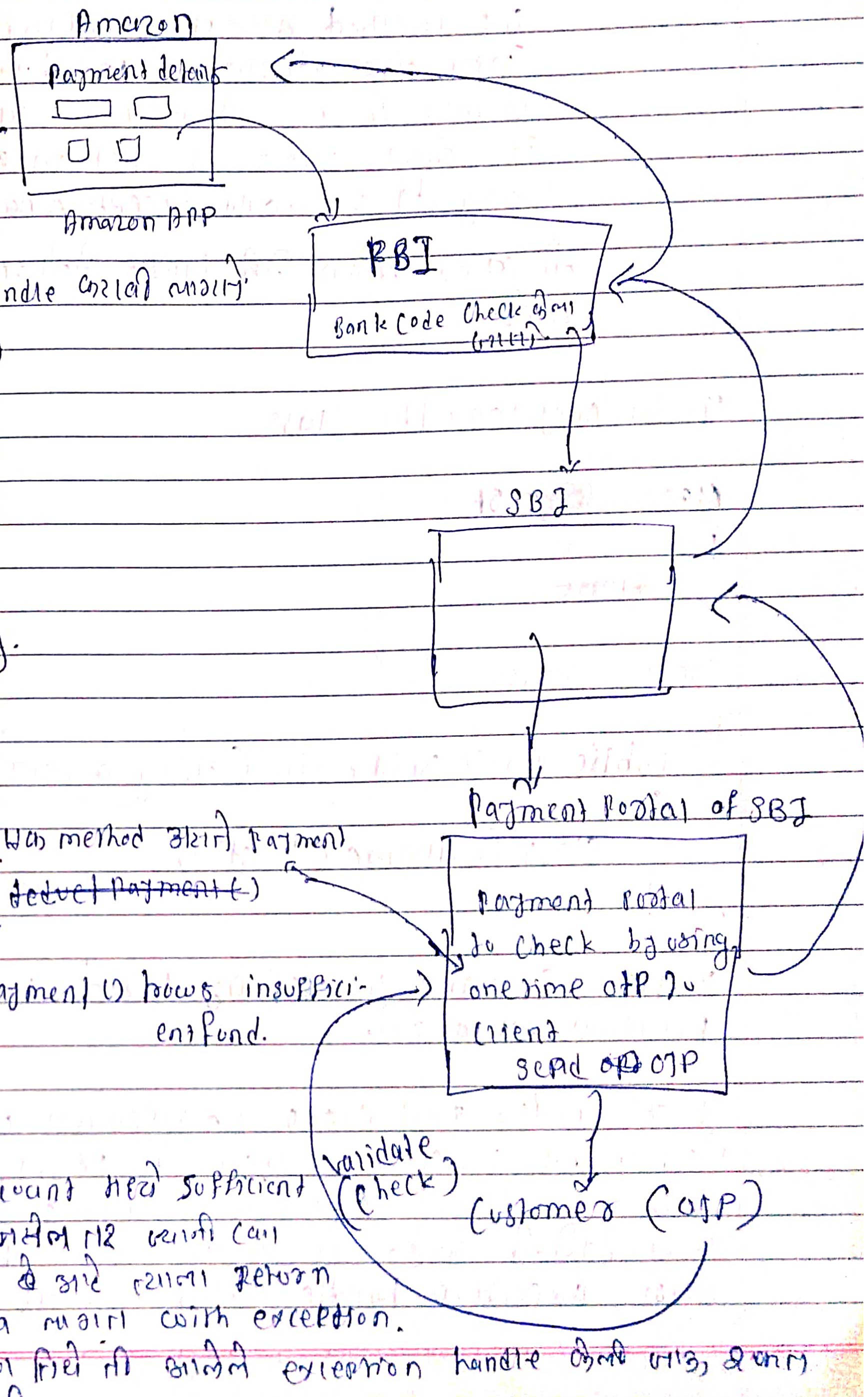
દરેક Exception class ના category wise કરેલી છે.
eg- networking Exception પર આધારિત exception class કો જુદો કરેલો છે.

IOException એ input અને output પર ખાસ exception આવતી વખતે use થાય છે.

eg-

```
class Demo
{
  public static void main(String args[]) throws
                                        Exception
                               or throws IOException
  {
  }
  java.io.BufferReader br = new java.io.BufferReader
        (java.io.System.in);
  String s = br.readline();
}
```

Example of Amazon - RBI -> SBI.



અહીં handle કરવી પડે
using
try
{
}
catch
{ }
+
finally.

Use method કરવું Payment
~~deduct Payment ()~~
deductPayment () throws insufficient Fund.
{
}
જો account માં Sufficient Fund નહીં હોય તો call કરીને return exception with exception.
જેના પછી તે exception handle કરવી પડે, નહીં

try -> catch ...
try -> multiple catch - try with ...

throws - જ્યારે method ને execution
જે method ... data proper
મળશે નહીં ત્યારે તે method જ્યાંથી
મોકલાવી ત્યાં તે method જ્યાં
call કરેલા હોય ત્યાં Return કરશે.
using throws and Exception name
થી exception ને handle કરી શકાય
નહીં.

eg. In only load the class

~~Class Test~~
~~static~~

```
Class Demo
{
    public static void main (String args[])
    {
        Class.forName ("Test");
    }
}
```

Program Compile successfully થઈ જશે પણ
Runtime પર error થશે.

જ્યારે Class loader Test class ને શોધવા જશે ત્યારે
તે class ઉપલબ્ધ નહીં હોય તો
JVM ના default Exception handling દ્વારા
execute થશે. ... Exception handle કરેલી નથી

Exception : ClassNotFound : Test (Exception type, Exception name)
(Java.lang.658)
(Java.lang.8) } Description

Description ને printStackTrace() method display કરશે.

```
try
{
}
catch
{
}
finally
```

આ Block વડે exception handle કરી શકાય throws કરી શકાય નહીં.

* NumberFormatException (Uncheck Exception)

```
Class Demo
{
    public static void main (String args[])   "Ten"
    {
        int a = Integer.parseInt(args[0]);
    }
}
```

value ઉપરના પર integer ના હોય અને String હોય. "Ten" તો NumberFormat Exception આવશે.

new throw ArithmeticException

implicitly JVM કરશે.

ClassCastException — uncheck Exception

```
class Parent
{

}

class child extends Parent
{

}

class Demo
{
  public static void main(String args[])
  {
    Parent Class Parent
    child c = (child) new Parent();
  }
}
```

↑ ClassCastException

Parent can not Converte into child.

જ્યારે throws Exception આપેલ હોય ત્યારે exception આપણે handle કરતા નથી. આપણે throws ના લીધે call કરેલ હોય ત્યારે જે ઉપરની method માં રહેલ હોય ત્યાં આપણે પરત jvm ને ફેંકી આપે.

```
Class Demo
{
  public static void main(String args[]) throws Exception
  {
    int a = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);
    int c = a / b;   → new ArithmeticException
  }
}
```

0 value.

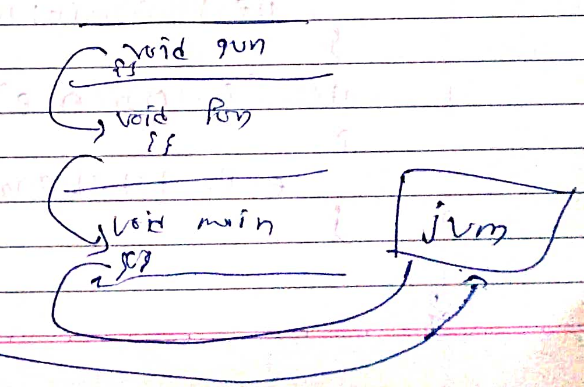
new ArithmeticException class નો object create કરવામાં આવે.

અહીં

throw new ArithmeticException

જ્યારે આ exception ના ઓબ્જેક્ટ handle કરવા માટે નથી ત્યારે implicitly jvm default exception handler ને call કરવામાં આવે. exception type, name, અને description jvm printStackTrace() આ method નો use કરીને print કરે.

StackFrames



printStackTrace() method

↑

printStackTrace એ ઉપરના થી error આવતા હોય તે print કરે.

Class loader-નું Hierarchy મા loading deligation Hierarchy હોય.

```
Class Demo
{
  Public static void main(String args[])
  {
    try
    {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int c = a/b;
    }
    Catch (ArithematicException ae)
    {
      ae.PrintStackTrace();    }  આ ત્રીણ method
      ae.toString();           }  JVM ના હોય છે.
      ae.getMessage();
    }
    Catch( ArrayIndexOutOfBound AB)
    {
      AB.PrintStackTrace();
    }
    Catch( NumberFormatException ne)
    {
      ne.PrintStackTrace();
    }
    Catch (Exception e)  — general Catch Block
    {
      e.PrintStackTrace();
    }
}
```

ArrayIndexOutOfBound
NumberFormatException
ArithematicException

→ Child class of → **Exception Class**

જ્યારે આપણે exception direct parent ના object દ્વારા catch કરીએ show કરીએ ત્યારે પણ sequence મુજબ child ના parent સુધી exception મળે.

① try નો Block with Catch Block હોય છે. એકલો try without catch ચાલતો નથી. ને catch નો એકલો without try ચાલતો નથી.
આ બંને ના Block વચ્ચે કોઈપણ extra statement ચાલતા નથી.

② type
1) one try with one catch Block
2) one try Block to multiple Catch Block
3) one try Block ~~multiple~~ one generic catch Block (સર્વસામાન્ય Exception class).

① one try Block one catch Block.
જ્યારે Exception ઉત્પન્ન generate થાય છે ત્યારે માહિતી પ્રમાણે catch Block ... class ના object implicitly throw કરવામાં આવે.

② one try Block multiple catch Block
જ્યારે Exception ... ત્યારે exception નો object implicitly throw કરવામાં આવે છે ... catch Block માં object મોકલે છે.

⑤ one try Block and multiple catch block and with generic catch Block.

જો કેટલા Exception તૈયાર થાય તેમાં આપેલ હોય catch Block માં catch કરી શકાય.
જો unexpected Exception આવે ત્યારે parent Exception Block execute થશે અને normal termination ને abnormal કરે તેવું ના થાય normal termination થશે.

• try અને catch Block સાથે આવે છે.

જો program માં try અને catch Block ના હોય ત્યારે exception તૈયાર થાય તો delegate કરે છે JVM ને. ત્યારે normal termination ને બદલે abnormal termination થાય. ત્યારબાદ તે abnormal termination રોકવા માટે try catch Block લખવું.

Exception તૈયાર થાય ત્યારે implicitly throw new exception object () તૈયાર થાય છે અને તેને catch કરવા માટે class કે parent class catch કરી શકાય.

eg-
(પહેલાથી question નહિ આવે.)

① Presentation દરમ્યાન પૂછેલા question હોય તો project presentation band નહિ કરવામાં આવે તે ઉપરાંત new question પૂછે.

② File not found - જો તે file create કરી નહિ હોય તો runtime પર exception generate થશે અને તે handle કરવું પડે.

તો exception handle ના થાય તો normal termination થવાને બદલે abnormal termination થશે.

જે line માં error થઈ હોય તેના પછીની no. of lines try Block માં લખવી અને catch block માં handle કરવી.

About try and catch exception

```
try
{
    ---------------- error
    [ ........ ]  X
}
catch ( Exception e)
{
}
```

આ line માં error આવે તો ત્યારપછીની line execute થતી નથી.

જ્યાં error આવે ત્યાં catch handle કરવી.

જો exception handle ના કરેલ હોય તો JVM કરે છે.
JVM પાસે 3 method

- printStackTrace - type, name, description
- toString - type, description
- getMessage - name of exception

આ method by default છે જે exception આપે છે.

try પછી catch ના હોય તો exception catch થશે.

Catch & try ના Rules

a) try ani catch blocks ક્યાં method અંદર લખવાનો / class level ના લખાય નહી

1) જ્યાં જ્યાં line ના code લખવા.

2) કઈ line મા Risk લાગતી હોય તે line લખવાની.

3) try cannot work without catch.

4) try Block exception આવવા પર implicitly throw (exn). object throw કરશે.

5) catch Block ના exception catch કરવા abnormal termination રોકવા.

~~.......~~

Exception occur થવા પર

① void func()

```
void func()
{
    System.out.println("1");
    try
    {
        System.out.println("2");
    }
    catch (Exception e)
    {
        System.out.println("3");
    }
    System.out.println("4");
}
```

જો આ line મા Exception આવી ગઈ તો program abnormally terminate થશે. try ani catch block execute થશે નહીં.

②

```
void func()
{
    s.o.p("1");
    try
    {
        s.o.p("2");   ← exception આવી ગઈ તો
        s.o.p("3");
    }
    catch (exception e)
    {
        s.out.p("4");
    }
    s.out.p("5");
}
```

exception આવી ગઈ તો matching catch Block શોધશે, જો મળશે તો catch કરશે, જો નહીં મળે તો JVM ના default exception handler call થશે અને printStackTrace() method ના call થશે. type of exception, name of exception, Description of exception print કરશે console પર.

③ try ani catch Block class level ના લખાય નહીં પરંતુ method level ના લખાય.

④

```
void func()
{
    try
    {
    }
    catch (exception e)
    {
        System.out.println(e);   — આ catch માં exception
    }
}
```

આવી ગઈ તો લખાય છે. try ani catch Block ના લીધે abnormal termination નહીં થાય.

* nested try catch Blocks

એક try અંદર catch Block ના દ્વારા ઘર આપવા
ઘણા try ની catch Block વિના ઘર આવતા nested
try catch Block કહેવાય.

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

(nested try catch Block)

```
void fun ()
{
try
```

```
void fun ()
{
  try
  {
    try
    {
      int size = Integer.parseInt(args[0]);
    }
    catch (NumberFormatException nfe)
    {
      s.out.println("Number Format Exception");
    }   // ઘર value mismatch થાય છે
    try
    {
      int[] arr = new int[size];
    }
    catch (NegativeArraySizeException nase)
    {
      s.o.p("negativeArraySizeException");
      // ઘર size variable ની value negative
      આપીએ તો error થશે.
    }
    try
    {
      System.out.println( arr[5] );
    }
    catch ( ArrayIndexOutOfBound aioob )
    {
      s.o.p("Array Index invalid")
    }
  }
  catch( Exception e)
  {
  }
}
```

ઘર આપવા Dependent statement હોવાને કારણે
ઘર આપવા કે કરવા શકતા ઘણા ઘર પહેલાના statement ના
error આવતા ઘર dependent statement execute થઈ શકતા
નથી. ત્યારબાદ nested try catch વાપરવા
જોઈએ exception handle કરાવતી

દરેક આપવા catch Block રાખવા, multiple try જોડી
catch Block રાખી શકાય. a exception handle કરી
શકાય.

```
void fun()
{
    try
    {
    }
    catch ()
    {
    }
    try
    {
    }
    catch()
    {
    }
    try
    {
    }
    catch()
    {
    }
    statement 1 ;
    statement 2 ;
    statement 3 ;
}
catch ()
{
}
```

અહીં ક્યાં statement માં જો exception આવી શકે તે handle કરેલી છે. ધારો કે statement 1, statement 2, statement 3 execute કરાવવા અગત્યના છે તો આ statement માં error હોય તો statement try અને catch block માં મૂકવા જોઈએ. exception handle કરવાશે.

try block માં break, continue અથવા return statement વાપરી શકાય.
જ્યારે try માં exception આવે અને matching catch block ના મળે ત્યારે execute થઈ શકે અને abnormal termination.

~~finally~~

* Finally Block

Finally Block ની આવશ્યકતા - finally block ની અંદર આપણે કોઈ resource વાપરતા હોઈએ તો close કરવાની જવાબદારી છે.

Third party sql connection close કરવાની જવાબદારી છે.
જ્યારે કોઈ file માં open કરેલા data used કરવામાં આવે ત્યારે close કરવાના હોય છે.

Finally Block માં resources cleaning and close કરવાનું કામ કરાય. આ કારણોસર finally block આપે છે.

Finally Block હંમેશા execute થાય છે. Program normal termination કે abnormal termination થાય તો પણ finally block execute થશે.

Finally block લખવો કે નહીં તે compulsory નથી. સ્વૈચ્છિક છે.

Finally block ને class level માં લખાય નહીં. ફક્ત method level માં લખાય.

Finally block try અને catch પછી લખાય.

Finally block try → finally વાપરવામાં આવે.

Finally block try catch વગર લખી શકાય નહીં. finally માટે try-catch finally ની જરૂર છે.

Finally independent લખાય નહીં.

Finally block try and Catch ની આગળ પાછળ લખાય.

try ↓ Catch ↓ finally ✓	finally ↓ try ↓ Catch ✗
try ↓ finally ↓ Catch ✗	try ↓ and finally ✓

eg -

```
~~void fun()~~
int fun()
{
    try
    {
        return 10;
    }
    catch (exception e)
    {
        return 20;
    }
    finally
    {
        return 30;
    }
}
```

```
public static void main(String args[])
{
    int a = d.fun();
    S.o.p(a);
}
```

output : 30

અહીં try માં 10 return કરાવે પણ program terminate થતા finally block execute થાય, આથી આપણે return 30 મળે.

Finally - Block resource deallocation કરવા માટે use કરવામાં આવે.

direct finally block લખાતો નથી [illegible] try block પછી લખાય catch પછી [illegible] લખાય.

* multiple try and catch block પર લખાય [illegible]

~~પર catch block લખીને વધ્યું~~

પર 1 try block પર લખીને જોઈએ તો 2-3 Error આવતા [illegible] using or (|) operator નો used કરીને એક catch ની મદદ multiple exception handle કરી શકાય.

try with multiple catch

try with one catch and its having multiple exception

nested try/catch Block. Program

```
void func()
{
    try
    {
        int size = Integer.parseInt(args[0]);
    }
    catch(NumberFormatException nfe)
    {
    }
}

void Run()
{
    try
    {
    }
    catch(ArithmeticException | NumberFormatException |
          ArrayIndexOutOfBoundException)
    {
    }
    catch(Exception e)
    {
    }
}
```

Point :-

- finally

* User defined Exception

java માં બધી exception આવે ત્યારે જ્યાં check a uncheck exception ત્યાં રહેવાવાળી predefine exception આવે. ત્યાં રહેવાવાળી exception class ના ~~class~~ parent class Exception class હોય છે.

~~Exception:- predefine exception by jvm~~
~~RuntimeException:- બધી exception માં~~

Exception:- compiler દ્વારા ચેક થાય અને runtime
(partially check) ના exception ઉભા થાય પણ compiler પકડી શકતા નથી ત્યારે runtime ના exception આવે ત્યારે બધા predefine exception આવે ત્યાં throw કરવામાં આવતા.

eg. file not found, end of file, null pointer, Database connection

કે જેમાં problem runtime ના સમય આવે ત્યારે compiler ની handle કરવાની જરૂર આવે.

RuntimeException:-

compiler ના ~~મદદથી~~ દ્વારા ચેક ન થાય પણ run time ના exception આવે ત્યારે તો class પર RuntimeException પરથી inherit કરવામાં આવે તો તે predefine exception હોય તેવી ખબર નથી. તેથી તેના પર handle કરવા માટે user define exception નહીંતર abnormal termination.

multithreading

multitasking

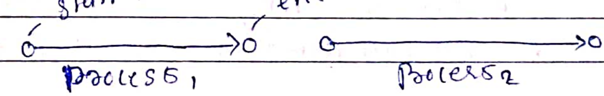
multiprocessing | multithreading

multiprocessing:
multiple process at time ચાલુ રહેવાનાં ત્યાં time sharing ~~busing~~ bases પર જે operating system manage કરે.

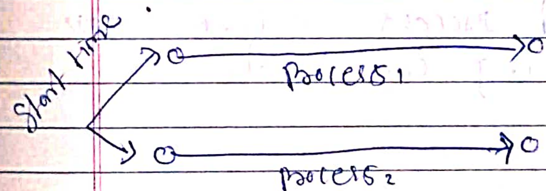
multithreading: eg- microsoft word office
autocorrection, auto completion, autospeechcheck

multithreading is a એક જ process માં under multiple ચાલુ રહેતા પણ parent process ચાલુ ન હોય તો child process ચાલુ નહીં રહે.
Child process ના parent process ઉપર depend રહેતા.

(1) sequential

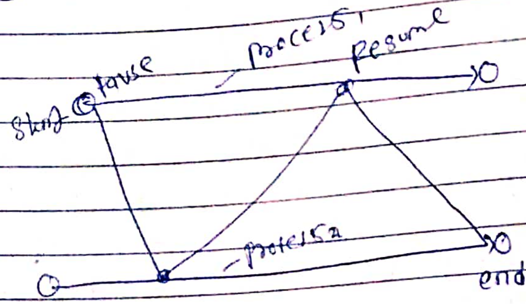


(2) Parallel execution



at time multiple process ચાલુ રહેતા sequential process execution handle by main process i.e called ~~OS~~ operating system program parallel execution execute on time sharing basis to processor.

(3) Concurrent execution



Pause-Resume-End is used multithread

પહેલા પહેલી process ચાલુ કરવામાં દુસરી process execute કરવામાં હતી.
ત્યા દેખવા Processor CPU IDEL mode થતી.

પહેલી a process execute હોવાથી છે a દેખાતા CPU IDEL mode વાળી આવતી ત્યા દેખતા તો તો IDEL time થાય તો દુસરા process ના હતો. આવાથી concurrent execution હોવાથી.
Concurrent execution only achive by the object oriented programming like C++, java

આ Process ના concurrent execution ના multithreading કહેવાય.

multithreading મતી process execute કરવામાં time saving હોતી થતી.

3

Java મતી multithreading કઈ રીતે કરવામાં?

1. Runnable Interface – Functionality મળતા. ના Interface મા.
2. Thread Class
3. Thread Group Class
4. Thread Local Class } Java.lang package મતી છે

(1) Runnable Interface

```
interface Runnable
{
    ~~Public~~
    Public abstract Void run();
}
```

– આવતી Functional Interface હોવાથી
↓
only method આવે તેને Functional Interface કહેવાય.

Runnable interface માં મતી run() method હોય છે કે a Interface a Runnable ~~use~~ ~~[illegible]~~ ના implement કરવા તો method override કરવા ~~[illegible]~~ પછી આપણી thread ~~[illegible]~~ લિંક કરવા.

run(); method thread (process) ના run કરવા માટે a use હોતા આવી.

(2) Thread Class

Thread Class મતી run() method override હોતા છે.
Thread class a predefine class છે. તો run method ના call કરવા-માટે use હોતા છે.

Class Thread ~~implemen~~ implements Runnable

```
{
    @override
    public void run()
    {

    }

    public ~~start()~~ void start()
    {
        start0();
    }
}
```

start zero() method run method ની call કરે.

(3) ThreadGroup class - ગ્રુપમાં આવતા Thread ના

(4) ThreadLocal - locally execute થતી હોય Thread.

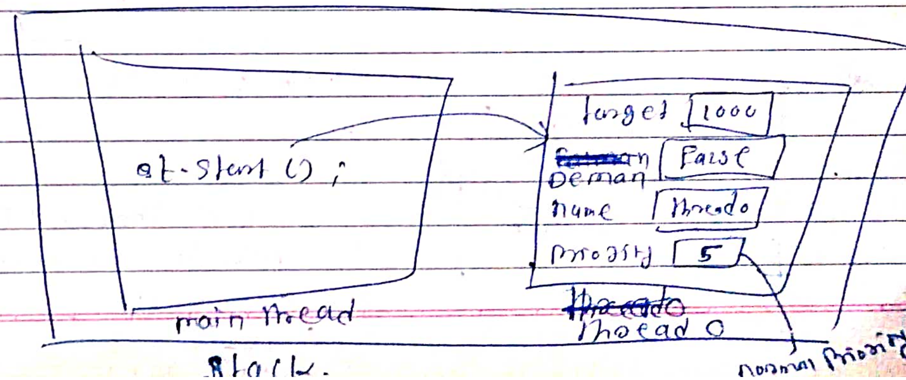
class Thread માંથી run(), start() આવ્યો. start() method માં આપણે start0() method ની call કરી. આપણે class ની રચના કરેલી છે.

run() method એ Runnable interface માંથી લીધી. એ abstract method ની Body લખેલી છે. override કરેલ.

આ class જે આપણે ઉપયોગમાં લીધેલ નથી.

(1)

```
class Thread implements Runnable
{
    public void run()
    {
        for(int i=0; i<=50; i++)
        {
            System.out.println("run" + i);
        }
    }
}

class Demo extends Thread
{
    public static void main(String args[])
    {
        Thread t = new Thread();
        t.start();

        for(int i=0; i<=50; i++)
        {
            System.out.println("main" + i);
        }
    }
}
```



① Target (null) –
target માં જે Thread ની address store કરવામાં આવે તો thread ચાલુ થાય ત્યારે store કરેલ થાય.

② Deman ~~[illegible]~~ (false)

~~true~~ ના કારણ તો thread internally implement કરવામાં આવે છે જ્યારે explicitly કરવાના હોય તો run() continuously ~~[illegible]~~ ચાલુ રહે છે.
false - ત્યારે thread આપણે explicitly લખવાનો.

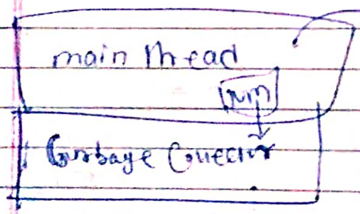
③ name (Thread0)

જેટલા thread ચાલુ કરીએ તે નામ થી Thread0, Thread1 ---- Thread n સુધી ~~[illegible]~~ (8b) મળશે.

④ priority (5)

normal a bydefault priority એ normal રહે છે.
max priority (10) a min priority (1) આવે છે.

*



stack માંથી JVM ચાલુ થાય.

main thread માટે stack માંથી JVM બનાવે છે. ચાલુ થાય
Garbage Collector object clearing

②

~~Class 11~~
~~Class~~
~~Class Thread implements Runnable~~
~~{~~
~~Public abstract void run();~~
~~}~~

~~Class Demo extends Thread~~
~~{~~

② Class Thread implements Runnable
{
Public void run()
{

}
}

Class Demo extends Thread
{
Public Static void main(String args[])
{
Demo d = new Demo();
d.start();
}
}

→ Thread Class ની method call કરી છે o using start() method.

જો આપણે start() વાળી method ના call કરીએ તો thread execute થશે.
run method execute થશે.

(3)

```
Class Demo implements Runnable
{
    Public void run ()
    {

    }

    Public Static void main (String args[])
    {
        Demo d = new Demo();
        Thread t = new Thread (d);

        t. Start();
    }
}
```

Thread Class ના દ્વારા Constructor ના આધારમાં ~~custom~~ userdefine Class ના Object પાસ કરાય. આ user defined Class ના object ની start() method ના call કરે છે. start() method by using virtual keyword ની object type ની method Call થતી વખતે run() method આપણા user-define Class ની જ ઉપયોગ થાય છે.

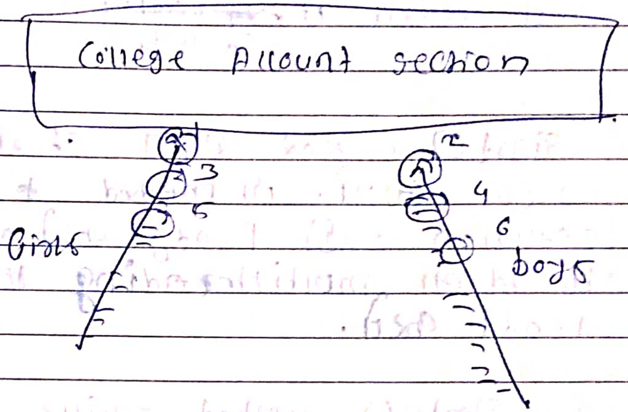
Start() method Thread Class ની Call થાય છે તે run method by using virtual keyword Object type ની Call થતી વખતે.

multithreading શા માટે જોઈએ?

Thread - પ્રોગ્રામના Process ના sequence માટે execution થાય.

multitime Thread execution (process) એ પ્રોગ્રામ ના time (ના execution) ~~આવેલ~~ કરતા તે independent કરવામાં આવે તેને multithreading કહેવામાં.

eg -



જે multithreading માટે at a time ઘણા Process ચાલુ કરવામાં. આપણે execution Concurrent કરીએ.

Scenarios

(1) ~~પ્રોગ્રામ છે~~.

(2) ~~એક સાથે Thread~~ જ

~~રુલ્સ~~

Rules:

① કોઈ પણ એક object ની start method ના call કરવા માટે,

object.start();

Start method Thread ~~execution~~ ના Ready state માં ~~explicitly~~ implicitly લાવે છે.

start() method મુજબ start() (zero) આ method ના call થાય. આ method thread ના execution માટે Ready state માં મૂકે છે અને thread ના multithreading મુજબ તે execution માટે Ready કરશે.

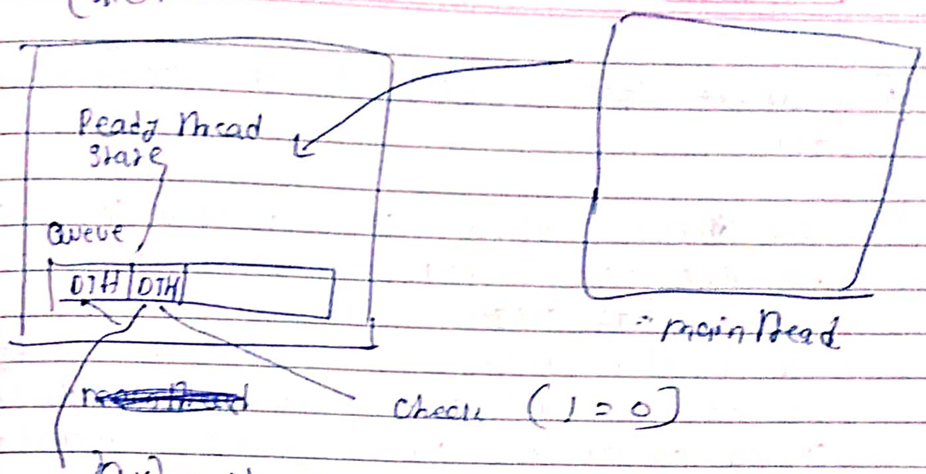
start() method વારંવાર એક object પર start() method call કરી શકાય. દરેક type ના object ને class ના run() method execute કરે.

જો class માં start method ના હોય તો parent class માંથી method call થશે. નહિતર compile time error

② એકજ type ના multiple object બનાવીને એક જ object ની start method ના call કરવા પર Runtime exception થશે.

~~Illegal~~ IllegalThreadStateException

Collection object



બીજી વાર object start કરવાથી Type ની state ના કારણે IllegalThreadStateException runtime માં generate થશે.

③ જો multiple object માં અલગ thread માટેનો logic execute કરવો હોય તો દરેક object અલગ બનાવીને તેના object.start() method ના call કરવા. એમ object અલગ અલગ ~~[illegible]~~ બનાવવા.

eg.

Demo d1 = new Demo();
— — d2 = ——;
— — d3 = ——;
— — d4 = ——;

d1.start();
d2.start();
d3.start();
d4.start();

(4) ~~જો multiple objects બનાવવા હોય તો~~
run

(4) જો run() method માં logic પ્રત્યેક object ના સરખા નથી હોય, ~~તો ... શકાતી નથી~~
~~એક class~~
પ્રત્યેક object ને અલગ type ના બનાવવા માટે run() method multiple લખી શકાતી નથી. તેથી
પ્રત્યેક object ના separate class લખવા પડશે.

```
class Thread1 extends Thread
{

}
Class Thread2 extends Thread
{
}
Class Thread3 extends Thread
{
}

class Demo
{ p.s.v.main(String args[]) {
  Thread1 t1 =new Thread1();
  Thread t2 = new Thread2();
  Thread t3 = new Thread3();

  t1.start();
  t2.start();
  t3.start();
}
}
```

(5) જો run method માં પ્રત્યેક value પસાર કરવી હોય તો

~~run()~~
run() method માં value લેવાતી નથી.
જો પસાર કરવી હોય તો Data member declare કરવા પડશે (field);

એ field માં value મૂકવા માટે Constructor લખવો પડશે. તો Parameterised Constructor બનાવી data member ને initialize કરવો.

આ data member ને run method માં this.data member વાપરી શકાય.

(6) એક Class માં multiple run() method લખી શકાતી નથી. એ ~~multiple~~ પ્રકારના type ના multiple object પર method ના Call કરી શકાય નહીં.

Thread Life Cycle

main Thread — Custom Threa

↑

ક thread આપણા તૈયાર કરવા માટે implicitly તૈયાર કરવામાં આવ્યો.

Custom thread આપણા માટે કરીશું using main thread

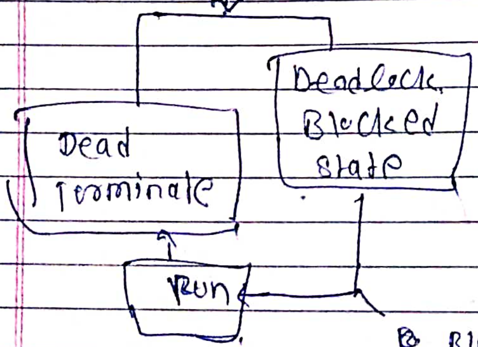
step 1) New state – new thread તૈયાર કરવામાં using thread class

↓

Ready to Run state – start method ના call કરવાથી thread Ready to run થઈ જાય છે.

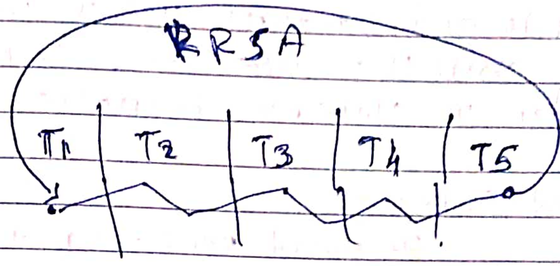
↓

Runnable – start method ના call કર્યા પછી start() method start() method ના call કરતી આ run method execute કરે.



Block state મળેલા બાદ જ આપ્યો જો run થતો આવી Dead થઈ જાય.

Round Robin Scheduling Algorithm



Concurrently T1, T2, T3, T4 time sharing basis પર કામ કરે આવી શેડ્યુલ process ના ટાઈમ માં (T5) ના પૂરા પછી process execute કરવામાં આવી જાય.

કે રાઉન્ડ Time sharing કરવી આવતા.

Time-waiting state અને waiting state

Time-waiting state

જ્યારે આપણા thread ના millisecond નક્કી કરેલ હોય ત્યારે time-waiting state કહેવાય.

waiting state :-

જ્યારે thread બીજા thread ની રાહ જોવે ત્યારે બીજા thread પૂરો થાય પછી time-waiting state મળે ત્યારે waiting state માં મળી આવશે.

new state :-

નવીન thread તૈયાર કરવા આવે ત્યારે Ready to run થતા પહેલા new thread આવે છે આવતા.

Runnable :-
જ્યારે નવીન thread તૈયાર કર્યા પછી Ready to run પોઝીશનમાં using start() method કરાવી start() ના call કરતા run પરથી state માં Runnable state કહેવાય.

Blocked State :-
જ્યારે thread run કરાવતા Deadlock occur થવાના કારણે thread block થઈ જાય. જ્યારે thread block થાય ત્યારે તે block state માં આવે.

Terminated :-
Thread નું execution પૂરું થાય ત્યારે તે terminate થઈ જાય. જ્યારે terminate થાય ત્યારે state આપણે get કરી શકીએ using getState() method.

Thread methods:

(1) sleep() :-
જ્યારે આપણે thread નું execution રોકવાનું હોય તો sleep() method નો use કરી શકાય.

તેમાં Parameter માં millisecond આપવાના eg- sleep(1000) 1000 એટલે 1 second.

sleep() method thread નું execution 0 થી second સુધી suspend કરે. time પૂરો થતા પછી ચાલુ કરશે.

sleep() method માં જ્યારે InterruptedException handle ના કરવામાં આવે તો error આવે છે.

" Exception throws કરવા જરૂરી નથી.
" દરેક class ના કાર્ય પ્રમાણે method override કરી શકાય ના class માં exception handled કરવા throws કરવાની જરૂર નથી "

(2) getName() :-
Thread0, thread1 એવું default name મળે છે પણ user define નામ આપવા માટે setName(" "); થી આપી શકાય.
↑ userdefine name

(3) getName()
Thread નું નામ get કરી શકાય.

(4) getState()
Thread ની current position ની જાણકારી state માં આપી છે તે get કરી શકાય.
new, Runnable, Terminated.

(4) setPriority()

1	5	10
MIN_PRIORITY	NORM_PRIORITY	MAX_PRIORITY

↑ By default આવે.

using setPriority() method થી આપણે thread ની priority set કરી શકાય.

d1.setPriority(10);
d2.setPriority(3);

Priority set કરવામાં એક exception છે જે તો ની exception handle કરી delegate પણ કરવી પડશે.

IllegalArgumentException ફેંક શકાશે

Argument ની value જેમાં 1 to 10 સુધી Range માંથી કરાવી સારા બહારની value લેવાથી ત્યારે IllegalArgumentException થશે.

By default main thread માં normal priority હોય છે. આપણે બનાવેલ Thread main-thread પછી બનાવવાના હોય છે. તેથી તમામ child thread પણ રહેશે. તેથી by default દરેક thread ની priority normal priority (5) હોય છે.

* Start method override કરાય કે ના ?

```
class Demo extends Thread
{
    public void run()
    {
    }

    public void start()
    {
        ------;
        ----;
        ----;
    }   super.start();

public static void main(String args[])
{

}
}
```

જ્યારે multiple thread માટે જ્યારે common logic વાપરવાનું હોય ત્યારે start method Thread class માંથી આપણા class માં override કરવાથી આપણે common logic લખી શકીએ.

ત્યાર બાદ Parent class Thread ની method call કરવા માટે using super keyword.

Awt & Swing:.

~~java.lang.awt.*;~~
~~java.lang.swing~~

import java.awt.*;
import javax.swing.*;

awt - Abstract window toolkit
swing:

awt ના કોઈ drawback આવવાથી swing ટેકનીક develop કરી.

awt ને extended કરીને javax પેકેજ બનાવી. જેમાં પ્રત્યેક class J ના letter થી ચાલુ હોય.

awt અને swing એ graphic Development of java. છે.

API - Application Program Interface એટલે package, class, Interface આપવામાં.

આ API મહત્વ class કરી લિખિત માહિતી developer ના જીવને જુરવાના કામોમાંથી બચાવવા આવે છે.

import javax.swing.JFrame;

JFrame ના class ના ઉપયોગ કરીને આપણી frame બનાવી શકાય છે.

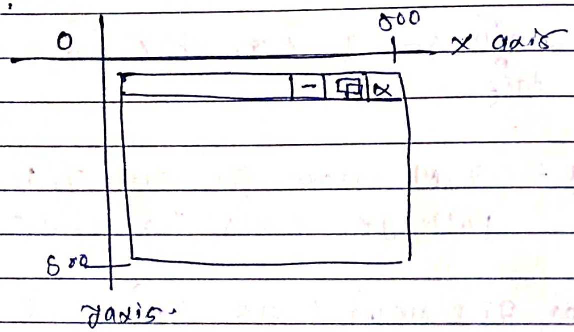
JFrame ના class ના extend કરવા નહીંતર Composition સ્ટ્રેટેજી object બનાવ કરવા વાપરવું જોઈએ.

JFrame માં 90% method અને variable (Field) static આવેલ.

Methods. JFrame frame = new JFrame();

1) frame.setVisible(true);
by default visibility false હોય છે. તેને દેખાડવા માટે true as a boolean Parameter પસાર કરવો પડે.

2) frame.setSize(800, 800)
આમાં બે Integer Parameter પસાર કરવામાં આવે. આ value graph Paper પર 4th quadrant માં આવશે.



3) frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

in JFrame class have exit-on-close variable have static variable. its used to close the Frame and they automatically terminate the java Program.

4) frame.getContentPane().setBackground(Color.BLUE);

Color.BLUE
↓
Class this class in awt package.

5) frame.setLocationRelativeTo(null);
આ frame ને સ્ક્રીનની center માં આપોઆપ ગોઠવી દેશે.

6) import java.awt.Dimension;

Dimension fsize = new Dimension(200, 200);

Dimension class ના under getSize() method આવે છે. જેનો ઉપયોગ કરીને frame ની width અને height get કરી શકીએ.
fsize.getSize();

Add on Button event listener

```
import javax.swing.ActionEvent;   // class
import javax.swing.ActionListener;   // Interface
```

btn_button.AddActionListener(this);

addActionListener () એ callBack method છે. જે જ્યારે Event fire કરવામાં આવે ત્યારે call થાય.
આ AddActionListener () method madhun actionPerformed() આ method મા call થાય.

actionPerformed () method ActionListener ના Interface માં છે જે આપણા class માં override કરવી પડે છે. તેથી ActionListener Interface implement કરવામાં આવે છે.

- Prototype

```
public void actionPerformed(ActionEvent ae)
{

}
```

આ class માં override કરવા ની જરૂર નથી override કરવું પડે.

Adapter class.

```
Interface Demo
{
    void ~~int~~ add ();
    ~~void~~ float sendvalue();
    void getPersentage();
}

class DemoAdapter implements Demo
{
    ~~void~~
    ~~Public add()~~
    ~~{~~
    ~~}~~
    ~~void Pc~~

class DemoAdapter implements Demo
{
    Public void add()
    {
    }

Public float sendvalue();
    {

    }
Public void getPersentage ()
    {

    }
}
```

```
Class Test extends DemoAdapter
{
    //override the method which you need

}
```

~~એટલે~~ Adapter class એટલે એવા class જેમણે method ની Empty implementation કરેલ હોય છે. કારણ interface માં આવેલ બધા method ના Body લખવા પડતા interface implements કરવાથી. તેથી આપણને બધા method ના body લખવાની જરૂર નહીં આપણને એક Extra class બનાવવા પડતા તેથી Adapter class બનાવ્યા.

Adapter class માં Interface implements કરેલ હોય છે. તેથી બધા method ના implementation કરેલ. આપણે child class માં method override કરવા પડતા.

Interface
↓
Adapter class
↓
Adapter Child Class.

key Event.

```
import java.awt.event.keyEvent;
import java.awt.keyAdapter;
import java.keyListener;

txt_first_number = new JTextField();
txt_first_number.setBounds(10,20,100,20);
                          x  y  width height
txt_first_number.addkeyListener(new keyDemo());
~~from~~

class keyDemo extends keyAdapter
{
    Public void keyRealeased() (keyEvent ke)
    {
        System.out.Println("key Realeased");
    }

    Public void keyPressed() (keyEvent ke)
    {
        System.out.Println("key ~~Realeased~~ Pressed");
    }
}
```

keyEvent એ class છે જે event ~~કોઈ પણ~~ કોઈ પણ component પર થાય ત્યારે તેની બધી Information get કરી શકાય.

keyAdapter:-

key Adapter is a class છે. આ class અંદર પણ interface implements કરેલા છે. આ interface અંદર ત્રણ method keyListener આ interface અંદર છે. ① keyReleased, ② keyPressed ③ key typed

keyAdapter આ class અંદર આ method માં implementation કરેલી છે. પણ તેમાં not available code of any line.

જેથી આ class આપણા userdefined class ની extend કરીએ છીએ. તો આપણે કરવા જેવું [illegible].

~~[illegible] multiple class [illegible] class~~

multiple Inheritence Java અંદર ચાલતું નથી માટે DemoFrame પહેલાં extends કરેલ હોવાથી બીજો userdefined class એ inner class તરીકે બનાવીને અહીં extends keyAdapter પહેલા કરવો પડશે.

~~textnum~~ txt_number_first.addkeyListener(new keyDemo())

reference variable txt_number_first છે. addkeyListener એ method ~~key~~ ActionListener ની છે. આપણે keyDemo user defined class છે.

addkeyListener() એ method નો parameter તરીકે class નો object આપેલો ત્યારથી એ call થશે. ત્યારે આપણે key Demo class અંદર ~~[illegible]~~

~~[illegible] parent [illegible]~~

keyDemo class extends key Adapter આપણે ત્યારથી પણ interface implements કરેલા છે.

keyListener વિશે ની method check કરવા માટે ની જરૂર નથી. પણ આ class અંદર implement કરી છે તેથી ~~[illegible]~~ [illegible] java અંદર પ્રત્યેક method virtual છે [illegible] child અંદર implement કરી શકાય. ત્યારે method call થશે.

```
interface keyListener
{
    void keyRealeased();
    void keyPreassed();
    void typed();
}

class keyAdapter implements keyListener
{
    void keyReleased()
    {}

    void keyPreassed()
    {}

    void typed()
    {}
}

class keyDemo extends keyAdapter
{
    // need which methed to child class
       implement. in tohechild
       by using parent class method.
}
```