

① How Computer works?

like about RAM, Networking, Internet works

Technical discussion

1. CPU \rightarrow "Brain of Computer"
- 4. CPU consists of microprocessor and controller
 - 4. Microprocessor consists of Transistors


 offset \rightarrow Address bus (A)
 data \rightarrow Data bus (D)

\rightarrow Transistors understand only
(only 1's)

o.1 \rightarrow machine level language

- Instructions all at in one place called program
to perform specific task

\rightarrow we have Machine level language (o.1)

In MLL we can't write program to do task
It's difficult to remember as a human's

- \rightarrow So, then Assembly level language came into picture which consists of alphabets and registers
Same this (AHL) also faces many issues

\rightarrow then after High level language (HLL) came into picture, which consists of English like words and simple symbols like (+, -, *, %)

MLL AHL HLL \rightarrow (Human understandable)

o.1

1.0

ADD

SUB

MUL

DIV

+, -, *, %,
print, scan, Math this
came in HLL

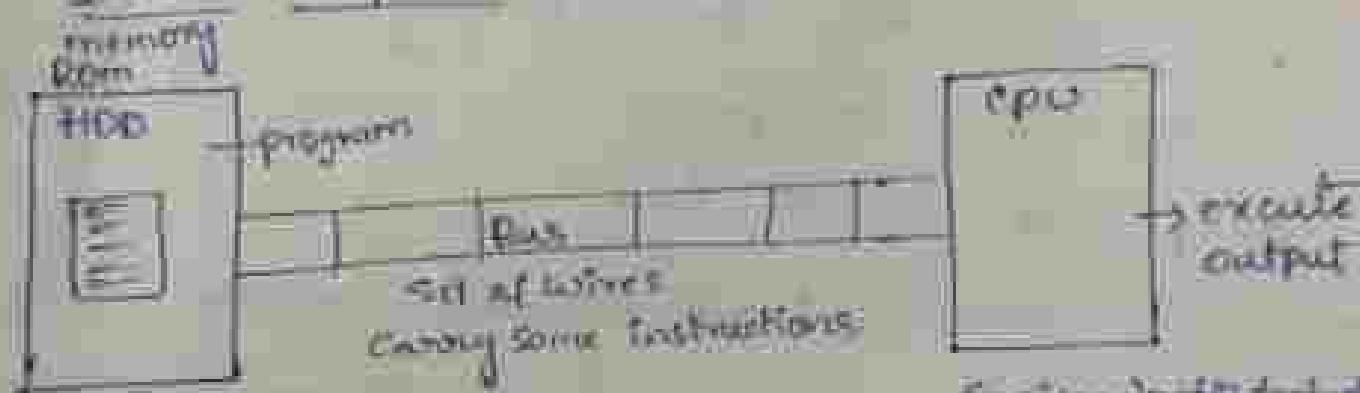
Computer understands only machine level language which is 0s and 1s

So, to convert Assembly level language to machine level language we use Assembler like mediator

and to convert High level language to machine level language we use Compiler.

Examples for High level language : C, C++, Java, Python

Inside Computer working

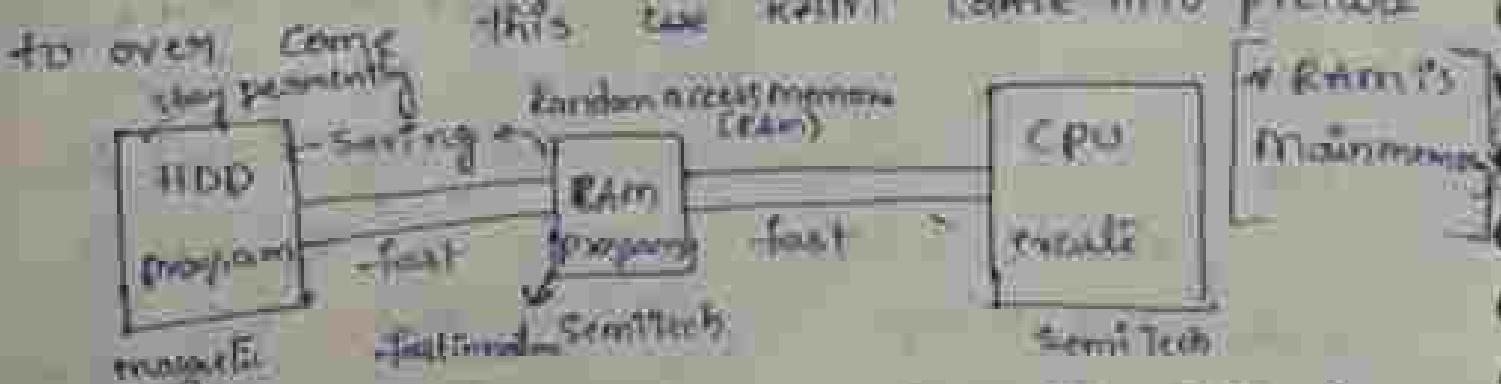


magnetic
technology

Slow in nature

So speed will mismatch b/w both

this too Ram come into picture



when we typing the code it will store in
RAM only

Disadvantage of RAM is it is volatile.

continuous power supply needed.

* If power supply shut down during typing the code, entire code will be gone. even when we open. So to this we have to make "SAVE" (ctrl+s) everytime then it will store in HDD (hard disk). In that hard disk program stays permanently until unless we delete.

But it will take time in saving program from HDD.

In HDD program storage we call source file and in RAM we call it as Byte file and in CPU we call it as Register.

* Cache memory closer to CPU if we opening again again it will store in cache memory.

SSD → Solid State Disk Better Version of HDD
- if use semi conductor technology (flash storage technology)

② Object file (.obj file) and executable file (-.exe)
.obj is incomplete file
↓
complete file

* Source file → Compiler → obj file which is incomplete file
we have odd libraries → linking through linker
and loading through loader → mix -obj file and libraries.

- .exe is complete which includes all the execute to get output.

Technical discussion

Java

C/C++ → Trending in 1990's

↳ 1991

↳ Sun MicroSystem → Co-founder: James Gosling

→ Easy to understand, object-oriented, portable platform independent (by) WORA

→ 1995 → Alpha & beta Version → trial Version freely downloadable, Open Source

→ Java 1.0 Version released

Name: Oak, Green, Java

2001 → Oracle acquired Sun MicroSystem

→ Object oriented

→ platform independent / WORA

→ programming language → when Internet came Java came.

Portable platform independent WORA

platform → microprocessor + Operating System

mp + OS

trial + hardware

Software platform → Operating System

platform dependency / non-portable \rightarrow C is platform dependent because of the architecture.



OS: Computer

because both application

OS: Computer



OS: Computer

OS: Computer



OS: Computer

Result/Op: No

Result/Op: Yes Result/Op: Yes

Program App / Software \rightarrow Same

In C program if both are same platform

m11 code will work gives you output.

because it is platform dependent.

actually code never transport because anyone will copy

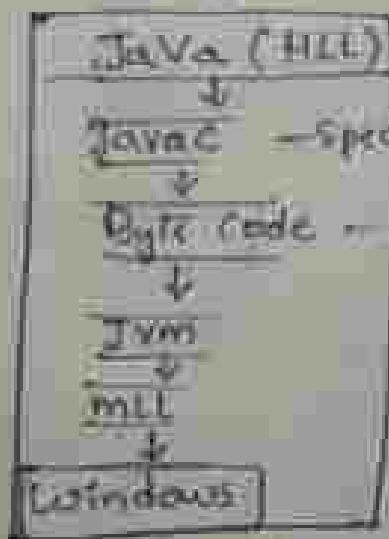
So coming to Java language

Java \rightarrow platform independent \rightarrow (write once run anywhere)

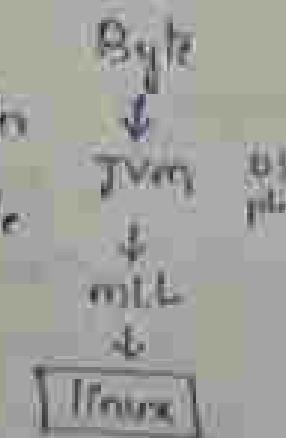
because it has special advantage

(WORA)

it will give you byte code called .class file after compiling java file which is saved



Result: Same



OS: Computer



OS: platform

We can share .class file (byte code) to any other platform we will get output.

.class file (byte code) is Intermediate code.

Human can't understand only JVM will understand because JVM is platform independent.

JVM is same in all platforms.

JDK \rightarrow (Compiler, Library, JVM) containing in JDK

JVM \rightarrow Java Virtual Machine

Java Compiler \rightarrow javac

Byte code (Independent) like JVM / (ML is dependent)

Execution
Java program

Class Launch &

\rightarrow save it as (Launch.java) file
class name should be same as save name

Source file

\downarrow
javac \rightarrow compiler \rightarrow Byte Code \rightarrow JVM \rightarrow Launchclass

\rightarrow machine level language \rightarrow platform \rightarrow O/p one

Compiler

Translate entire code at once

ex. C/C++

Interpreter

translate each line /
translate line by line

ex. Python, C

Java is both Compiler & Interpreter.

Famous version of long term support (LTS)

Java 7

Java 8 (Inclusion using)

Java 11

Java 17

why in C after compiling we will get
null but not in Java

Because C architecture is designed to go null
and Java architecture is designed to get byte code
is not to build internet applications.
Java is to build internet applications.

means after Compile.

class file \rightarrow In this file byte code is there
war file \rightarrow Collection of many class file
jar file \rightarrow Collection of many class file

war file \rightarrow web application archive

jar \rightarrow Java archive

what is purpose of giving arguments
at the time of execution

Code \rightarrow 3 phase before releasing the code
to users

Developer \rightarrow few Input associated with developer

Testing \rightarrow \uparrow \uparrow \uparrow \rightarrow Tester

Production \rightarrow \uparrow \uparrow \uparrow \rightarrow QA

Editors → Typing code

IDE → Typing code, Compile, debug, Test
Integrated Development Environment (IDE)

Java is verbose code → means we need to mention everything like class, main method.

→ What is method?

Methods are like functions

we write methods to perform tasks

Method → name, parameters, body, return type

e.g.: `Void play() {
 // body
}`

everything included in curly braces.

→ Main Method

Starting point for program to start execution from here.

If we not have main method JVM will not start execution.

Why main method?

JVM has coded such way that JVM will start only if main method is there.
to access without creation of object parameters.

`public static void main (String [] args)`

↓
To make visible

return
type

name of it is starting point to execute program

Return → forces the method to return value to the calling method of particular type execution of a method and can be used to return value from a method.

main → name of Jvm will search for this name only to start execution of program. void → return type (Java main() will not return anything)

public → To increase visibility

Static → Can be accessed without object creation.

String args[] → To receive command line argument

Command line → From command prompt instructing something arguments → information

OS → Javac Launch .java

↓
Jvm → java launch → command to execute it in static line

Command line
to execute

Class launch {
public static void main (String args[]) {

 System.out.println("Hello");

 System.out.println(args[0]);

}

 ↓ array
 ↓ index
 ↓ index 0

 ↓ ↓ what ever in this point
 ↓ ↓ that one

 ↓ if we don't have it will point

String → we need to give String only. Jvm is designed that way and we pass only argument in command line like "Launch" 10 10.5 it will be converted into String.

Syntax for Java:

```
public static void main (String [] args); } this can be any name.  
public static void main (String args[]); }  
public static public void main (String args[]); } All valid  
public static void main (String .. args); }
```

Variable = To store data

Static program language

Java, C, C++

int a = 10;

Before compile file

we should specify what type
of data.

Dynamical program

a = 10;

type of data no need
to mention

JDK → Compiler + JRE (JIT compiler) + library
JDK is required for developers as they write the
code and run the code JRE is required including
as they only run the code.

1. Ops (basic introduction).

2. Identifier | Variables - rules to write an identifier

3. Reserved words.

4. Data types and its chart.

oops

it stands for object orientation principles.

Object : Real time instance of an entity

Every object in real time will have 2 parts

① what it has → having

② what it does → doing

e.g. Car

what it has → name, noOfWheels, Speed

what it does → move, break, accelerate, willPress
(Variable name ex- noOfWheels)

class Car {

String brandName;

Int noOfWheels;

... classnames - it should
start with capital letter
start each word with capital
Ex: noOfWheels

- Statement ending

S.o.p("Hello") ; ; ; → we can write how many,
Compiler will remove those;

Identifiers / Variables → It is a name in Java
program.

it can be classname, methodname, Variablename.

Rules for Writing an Identifier

→ The only characters in Java Identifiers

are → a to z, A to Z, 0 to 9, _ (underline), #

→ if we use any other characters "it would
result in - Invalid (Compile time error)

- iii) Identifiers are not to start with digits
 - iv) Java identifiers are case sensitive
 - v) There is no length limit on Java identifier but still it is a good practice to keep length of identifier not more than 15 characters
 - vi) Reserved words can't use as identifiers
 - vii) pre-defined classes are user identifiers Even though predefined class names can be used but good practice we can't use
 - * Reserve words are normally in "lower case"

Review Board

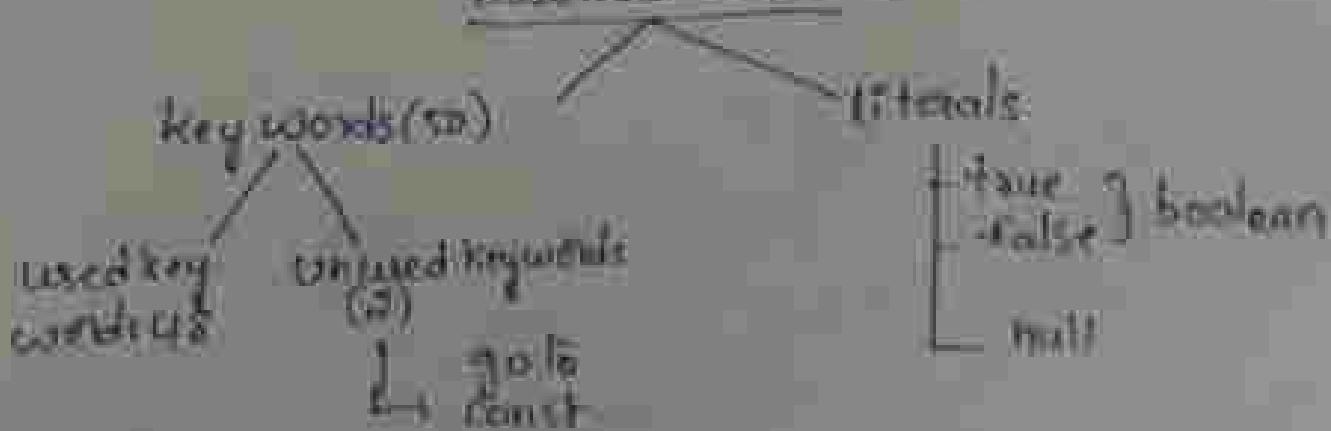
Keywords
 It is a built-in words/keywords, which has already a predefined meaning to it's reserve words.

Data Types

primitive data types

- a. Numeric Value's
 - b. Character Value's
 - c. Boolean Value's

Reserved words (53)



Literal → Any constant value which can be assigned to a Variable.

`int data = 10;` is Literal.

Data types : (Size and different values can be stored in Variables)

Every Variable has a data type.

Primitive data types : That specify the type and size of data.
primitive → pre-defined by language and is named in reserved keyword.

a. Numeric Values

↳ to store number

i) Integral data type (to store whole numbers)

① byte (1 byte = 8 bits)

Size of byte is → 8 bits

min. Value → -128

(→ to occupy more space we can take byte datatype which has more space)

max. Value → 127

byte marks = 255, Valid

byte marks = 125, Invalid

② Short = 16 bits (2 bytes) (when we perform the operation compiler is designed to store data in higher data types of specific type.)

min. Value = -32768

max. Value = 32767

Java give preference to

int datatype as it has

higher precedence order.

③ int = 32 bits (4 bytes)

min. Value = -2147483648

max. Value = 2147483647

→ it is commonly used

By default if we specify any literal of number type compiler will treat it as int only but we can also keep it in short or long

i) long = 64 bits (8 bytes)

min value = -9223372036854775808

max value = 9223372036854775807

when we work with long data type data would come to Java program in terms of 64's.

when int is not enough to hold bigger values then we can use long data type long data = 1234567890L

By default compiler will treat like integer but if we need to store in long data type at last of literal we should mention with capital "L" (on) small

i) To store real numbers

as float

Size = 32 bits (4 bytes) (why it will treat double)

min value = 1.4E-45

max value = 3.4028235E38

Q?

float number = 10.5f;

By default we will specify any real number

By default we will specify any real number compiler treat as double to specify to compiler compiler treat as float suffix it with 'f'(on) 'f'.

Double

Size = 64 bits (8 bytes)

Min Value = 4.9×10^{-324}

Max Value = $1.7976931348623157 \times 10^{308}$

Ex: `double d = 10.5d;`
(or)

`double d = 10.5;`

To map primitive data as Object in Java - from
DK 1.5 Concept of wrapper class.

Data types → follows specific format to store in
cls of it's
to perform operations.

byte = Byte (class)

short = Short (class)

double = Double (class)

long = Long (class)

int = Integer (class)

float = Float (class)

char = Character (class)

Now char → to store any character or special
character (ASCII → 65536 character wie these).

Size = 2Byte (16 bits) → It follows UTF (Unicode).

Syntax : `char a = 'A';`

Java is not pure OOPS because we use primitive data types.

By using wrapper class we can make 100% OOPS.

```
Ex: int a = 25;  
    int b = 2;  
    float c = 2.5f;  
    System.out.println(a/b);
```

Output: 12
Truncation/Rounding to zero

If we divide float by int output also int
means we are storing int in the
function means we are storing int in the
value output (also) same in int

Type Casting = Changing the data from one
data type to another data type.

Two types
a) implicit conversion
b) explicit conversion

a) Implicit: means internally changing
by Java compiler from lower data to higher
data type

```
byte a = 10;  
double b;  
b = a;  
System.out.println(b);
```

Output: 10.0

byte short int long float double
Implicit (small to large)

Explicit : It is done by programmer to convert
larger data type to smaller data type

byte
int $a = 500$,

byte b;

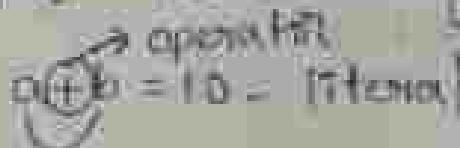
$a = (byte) b$

\hookrightarrow we are using explicit word
is op(b);

Output : - 45 loss of precision on data
will happen.

Operators

operator : To do any operation.


operator
 $a + b = 10 = \text{integer}$

operands

Incrementation

To increment value
($a++$)

increase existing value
by 1

$(a = a + 1)$ we can write it
as $a++$, $+a$

post increment
post decrement

Decrementation

To decrement value
($a--$)

decrease existing by 1

$(a = a - 1)$ we can write it as

$a--$, $-a$

post decrement
pre decrement

Post Increment

```
int a=5;
int b;
b=a++;
a [5]
b [6]
```

5.0. p(a);

5.0. p(b);

Output: 6 9

Note: post of pre increment gives u same value as pre increment

But when we are going to assign value another variable then it will change

Ex: post increment

```
int a=5;
int b;
b=a++;
a [5]
b [6]
```

In this first use value of 'a' and assign to 'b', then increment in 'a'

Ex: pre-increment

```
int a=5;
int b;
b=a++;
a [5]
```

In this first increment value of 'a' then that value assign to 'b'

```
Output: 6
a [6]
b [5]
```

decrement post

```
int a=5;
int b;
b=a--;
a [5]
b [4]
```

Note: post of pre

gives u same

after decrement

we are going to

another variable

then it will

change

post decrement

```
int a=5;
int b;
b=a--;
a [5]
b [5]
```

5.0. p(b);

In this first use value of 'a' and assign to 'b' then decre-
ment in 'a'

Ex: pre-decrement

```
int a=5;
int b;
b=-a;
a [-5]
b [4]
```

In this first decrement value of 'a' then that value assign to 'b'

```
Output: a [4]
b [4]
```

```

int a=5, b=20;
int b;
b= a++ + ++a + ++a;
System.out.println(b);
System.out.println(b);

```

```

int a=5;
int b;
b= a++ + ++a + ++a + a--;
System.out.println(a);
System.out.println(b);

```

Comments → JVM will not execute
 for single line comment (//)
 for double line comment (/*-----*/)

Operations → Fundamentals of Java

Arithmetic operations

→ Student

+, -, *, /, %

add, sub and div module (remainder)

This are normal very simple operations

int a = 10;

int b = 20;

int c = a + b; // 30

int d = a - b; // -10

int e = a * b; // 200

int f = a / b; // 0.5

int g = a % b; // 0

10) $20 / 1 = 20$

$$\begin{array}{r} 20 \\ \times 1 \\ \hline 20 \end{array}$$

Rem

Logical operators

AND, ||, ! = NOT

↓

AND

OR

0 → False
1 → True

not (Opposite)

0 → True

1 → False

AND

OR

0 0 = False

0 0 = False

1 1 = True

1 1 = True

0 1 = False

0 1 = True

1 0 = False

1 0 = True

AND → If all one True then it will True If one is

False then it will False as result

OR → All expressions are true then it will true

and if one false if there then it will true

all one False then it will False

NOT : If quite opposite If expression is True

then it will False if expression is False then it will True

Relational Operator

<, >, \geq , \leq , ==, !=

Composition, Relate \rightarrow we can do this things

$a > 10$,

$b = 20$; and if two one-true it will give-true
 $a \geq b$, \rightarrow In this if will check one if true it will true

$a < b$; \rightarrow if one condition is satisfy it will give true and if two conditions are not satisfy it will give false
yet if will give true

$a != b$ \rightarrow To compare two values we are using !=

ex: $10 != 10$

To assign a value to Variable we will use =

ex: $a = 10$,

Assignment Operator

(=) \rightarrow Single assignment

\hookrightarrow to assign value to Variable we are using

$a = 10$; Right to left.

Chained Assignment

int a, b, c, d;

$a = b = c = d = 10$;

Chained assignment.

a(10), b(10), c(10), d(10)

If all are Variables in same data type then
Only we can do chained assignment.

If all are Variables in same data type then
Only we can do chained assignment.

int abc = 5; Variable can hold one value

int abc = 10; at a time if we assign
new value then old value will
go off

Compound assign (or) shorthand assign

int a=10;

$a+=20 \rightarrow a = a+20 \Rightarrow a = 10+20 \Rightarrow 30$

$a-=20 \rightarrow a = a-20 \Rightarrow 10-20 \Rightarrow -10$

$a/=20 \rightarrow a = a/20 \Rightarrow 10/20 \Rightarrow 0$

$a\% = 20 \rightarrow a \% 20 \Rightarrow 10 \% 20 \Rightarrow 0$

add is different

$a = a+1 \rightarrow$ it will increase by only 1 value

Uniqueness

one operand is enough to do operations

Ex: int a=10
 one operand
 Assign operation.

Ex: Assignment
 of increment
 decrement

binary

more than one operand is involved in the operation

for Relational logical

Arithmetic operation

$$a=10$$

$$b=20$$

$$a+b$$

Operations need

Conditional

if we are doing any task based on condition

int a=10;

int b=5;

if-else (Only one condition)

{ }

(fully bracketed)

if (a>b){

int res=a-b;

System.out.println(res);

else

{ int res=a+b;

System.out.println(res);

[out put = 5]

if (Condition) {if true}

{ body = execute }

else { if false }

else { body = execute }

}

else if (more than one condition)

int a=10;

int b=20;

if (a>b)

System.out.println(a+b);

else if (a==b){

System.out.println(a+b);

else { System.out.println("a<b"); }

we can write one more "else if" in "if else"

how much we can write conditions.

Nested if is in one if another if condition
(Called Nested if)

```
int a=10;  
int b=3;  
if (a>b) {  
    if (a==10){  
        s.o.p("a=b");  
    }  
    else {  
        s.o.p ("Not doing");  
    }  
}
```

if (condition) {
 s.o.p("x");
}

else if (condition) {
 body ;
}

else {
 s.o.p ("");
}

in if else we won't
another if condition else if

By using logical operator of conditional operator

```
int a=10;  
int b=20;  
int c=30;  
if (a>b && a<c){  
    s.o.p ("a is least");  
}  
else if (b>a && b<c){  
    s.o.p ("b is High");  
}  
else {  
    s.o.p ("c is high");  
}
```

execute only if true
it will skip if false

execute and check
(practice)

Ternary question result is condition checking

(Condition) ? T : F colon

If condition is true before colon (:) colon

It will execute (T part will execute) colon

If condition is false after colon (:) it colon

will execute (F part will execute) colon

Ex: ① int a = 10; colon will execute

int b = 20; so Auto colon will execute

int c = (a > b) ? a : b so ans is b

System.out.println(c);

Ex: ② int a = 10; colon will execute

int b = 20; so before colon will execute

int c = 30; so before colon will execute

int res = (a < b) ? a : b; colon will execute

(Variable name) before colon also can have

to

State result

in before colon also can have another Ternary operation

(a < c) ? a : c)

(10 < 20) ? 10 : 30)

True

So before colon will execute

ans 10

Switch Case

Switch (Condition) {

 ↳ it will match with below case that will execute.

 Case 1:
 System.out.println("Hello");

 Case 2:
 System.out.println("Hello");

 ↳ If case 100 isn't satisfied

 }

 }
 int number = 100;

Switch (number) {

 ↳ It will print "Hello" because this is match go out of switch.

 ① Case 100 :

 System.out.println("not equal to 100");

 ② Case 50:

 System.out.println("not equal to 100");

 ③ Case 10 :

 System.out.println("not equal to 100");

 ↳ Please see case 100 is in ① It will check that case 100. If case 100 is not match then it will execute one more case. So to avoid this until case unequal to 100. So to avoid this we have to write (break;) after each case.

At least we can write default :

 System.out.println("un");

 ↳ If none of the case is not match default case will execute once check in IDE - Code in all possible ways.

Just for compiling purpose introduce

Percentage \rightarrow Total obtained by out of marks & Students
float percentage \rightarrow Total * 100 / 200

Swapping without third Variable

$$\begin{array}{l} b = b - a; \quad a = -b \\ a = a + b; \quad b = a \\ b = - (b - a); \end{array} \quad \begin{array}{l} a = -6 \\ b = 8 \\ a = -6 + 8 \\ a = 8 \\ b = - (a - 8) \\ b = -6 \end{array}$$

② $F \text{ or Fahrenheit} = (\text{Celsius} * 1.8) + 32$

leap year \rightarrow $100/4 = 0.83$ $100/400 = 0$
no. of year

③ To print char Value of char

s.o.p((int)'h')

④ char a = 'A'

int b ;

b = a;

it will print Accd Value of 'A'

when we are using Integer-type
Char-type to Integer

it will give a Int Value only

Loops

System.out.println(); after printing it will go next line. Ex: Vijay -> new line

System.out.print(); after printing it will remain in same line. Ex: Vijay -> same line

Loops : To Repeat any activity (or) same activity to repeat. 3 types
1) for
2) while
3) do while

For Loop

Three things are mandatory to execute for loop like initialization, condition, update).

Initialization ①

(1. steps)



Condition check (No) →

↓ (Yes)

(Stop)

Body of loop ②



Update ③ (increment (or) decrement)

- * until unless condition is true it will execute Body of loop and keep on update the value and condition check. if it will stop. Condition false

To print 5 (*) stars

For (Initialization; Condition; Update)

(Syntax)

Body of result

Ex:- `for (int i=0; i<5; i++) {` } Output

System.out.println(" * ");

0 *
1 *
2 *
3 *
4 *

• It is stacking from (0 to 4) so 5 numbers

0 1 2 3 4

int n=5 increasing

for (int i=0; i<n; i++) {

 | we can write n here

 | instead of value of number
 | we have to initialize n
 | first already

If you want to change number of stars
Just in above initialization of 'n' we can
make replace 5 with 9

Ex:- `int n=9`

Output: it will give

(Same as) `int n=9;`

10 stars

for (int i=0; ~~loop~~ i<n; i++) {

 | System.out.println(" * ");

Total 10 *'s

To print that start horizontal
we have to remove `ln` in `println()`
So that it will give you output as
follows

while (as long as)

```
int n=5;
```

```
int i=0;
```

```
while (i<n){
```

```
System.out.println("i");
```

```
    i++;
```

```
}
```

→ Entry Condition Check loop

→ we can execute based on
condition no need of update

→ Initialization like (for)

if you want output of
series like this

remove `ln` in `println()`
in program

here first initialize value
and (as long as) `while (i<n)` checking condition
if condition is ok after control will to body
execute and update (check)

$i=0$ | ~~1~~ as true - so point (1)

$n=5$ | ~~1~~ as true - so point (2)

like it will continue

once $5=5$ - fault so nothing print
terminates loop

do while (exit condition check loop)

```
int n=5
```

```
int i=0
```

```
do {
```

```
System.out.println("i");
```

```
    i++;
```

```
} while (i<n);
```

Irrespective of condition
at least once it will execute
for second time to repeat
it will check condition
But first time it will execute
without condition

Unlike for, while, only first condition requires here do-while - no condition required first time for second time it's required condition

Nested loop = means inside loop one more loop
we can write

Ex: Write a program to print

*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*

So treat rows as  columns as



rows
(i)

like no. of rows

j=0 to 4

for (int j=0; j<5; j++)

i=0 to 4

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

i=0 to 4

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

columns

(j)

We are starting here from 0 so ~~to (0 to 4)~~

program = ①

int n=5;

```
for (int i=0; i<n; i++) {  
    for (int j=0; j<n; j++) {  
        System.out.print ("*");  
    }  
    System.out.println();
```

Output:
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

→ the character continue next line

Program ① Break down the Debugging

```
int n=5;  
for (int i=0; i<n; i++)  
{  
    for (int j=0; j<n; j++)  
    {  
        System.out.print("*");  
    }  
    System.out.println();  
}
```

- first we are initialize n value to 5 (0 to 4)
- we are starting from '0'
- i [0] → 0<5 - true So control will go into body
in the body we have another loop j
- J [0] → 0<5 - true so control will go into body
In the body we have printing '*' after output
printing '*' j value is going to update (i.e. 0 to 4)
- J [1] → 1<5 - same above will process
- J [2] → 2<5 - same above will process
- until unless j value to 4 loop will execute
when j value 5 (5<5) false so terminate
- and again control go to & update i value
→ i [1] 1 < 5 - true so again above will take place
- Same for 2 i=2
i=3 Same problem
i=4
when i=5 5<5 - false loop will terminate
everything will stop

Program ③

Write a program to print pattern

```

int n=5;
for (int i=0; i<n; i++)
{
    for (int j=0; j<n; j++)
    {
        if (i==0 || j==0 || i==n-1 || j==n-1)
            System.out.print("*");
        else
            System.out.print(" ");
    }
    System.out.println();
}

```

→ i=0, j=0, i=4, j=4

i=1, j=0, i=4, j=3

i=2, j=0, i=4, j=2

i=3, j=0, i=4, j=1

i=4, j=0, i=4, j=0

i=0, j=1, i=4, j=4

= n-1

= 5-1

= 4

So both are same

$$n = \frac{5-1}{2} = \frac{4}{2}$$

$$\frac{5-1}{2} = 2$$

→ We initialize n=5, because it's 5 rows, 5 columns

i= 0 → refer to program ① breakdown

j= 0

here we are using conditions based on where we want star '*' and Space ' '.

As per program ③ our condition is

i==0 || j==0 || i== (n-1) || j== (n-1) → here we want

remaining we want ' ' Space

||= operator - if one true everything true

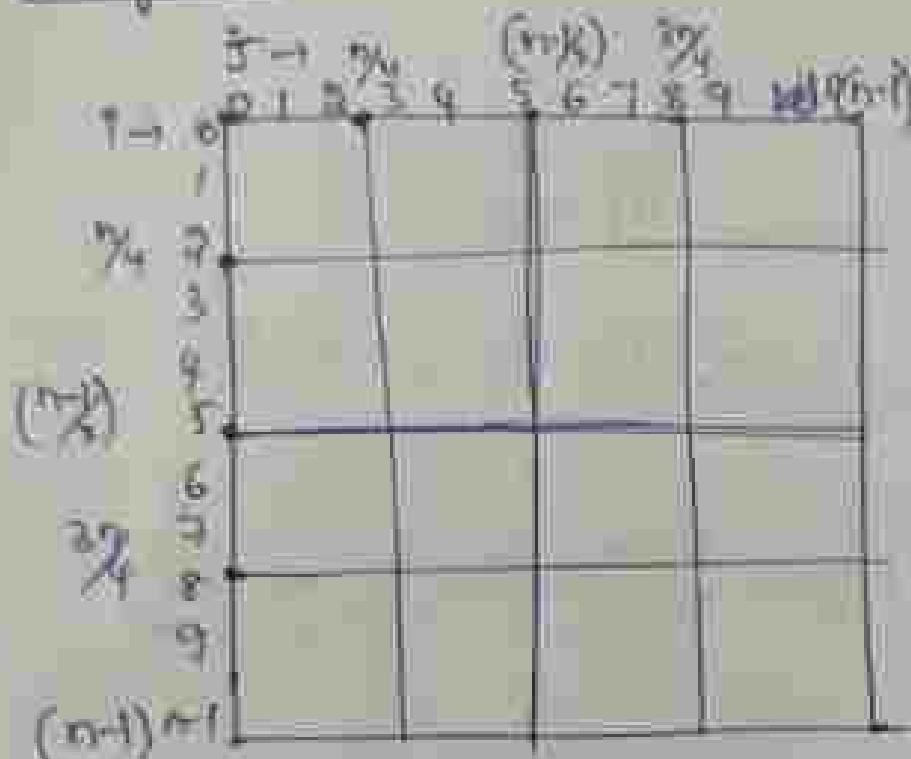
&& operator - if only both true it will execute

→ one & if one false everything false

Palindrome programs

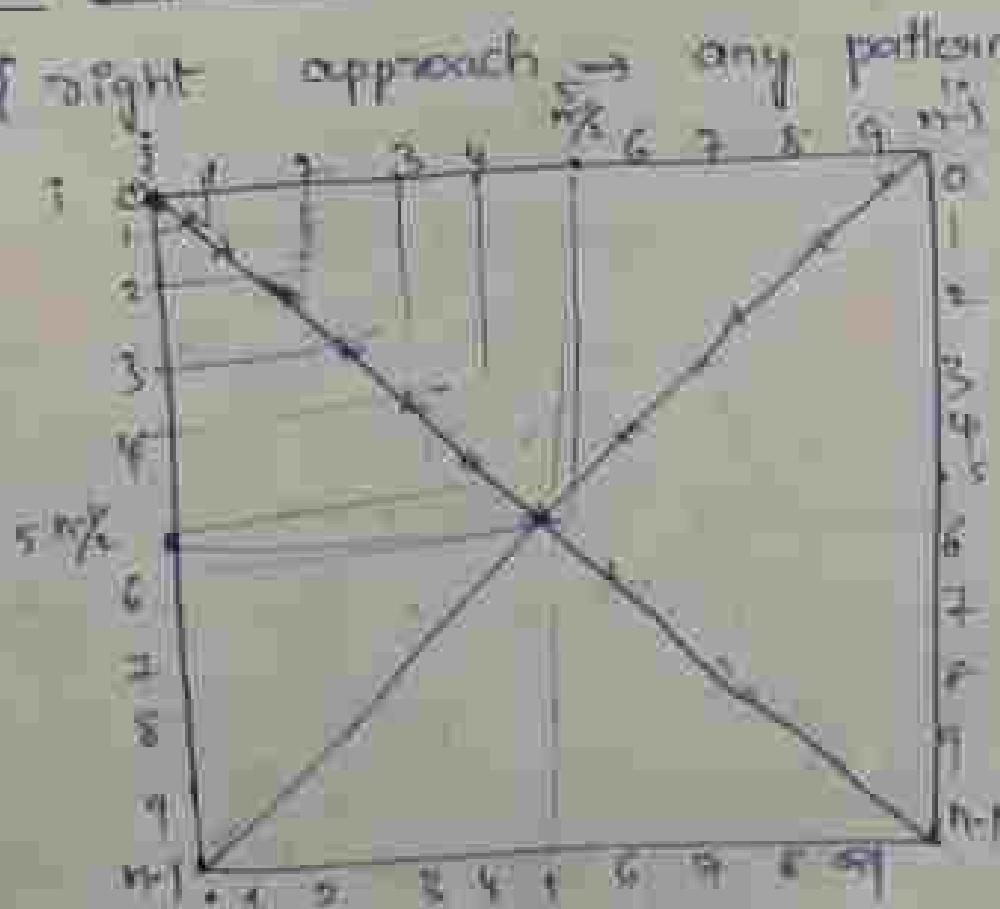
here I am not writing loop overall just conditions
but() with where we want 'W' and '-' spaces
and class

Diagram



for Alphabet

→ {right approach → any rule pattern }

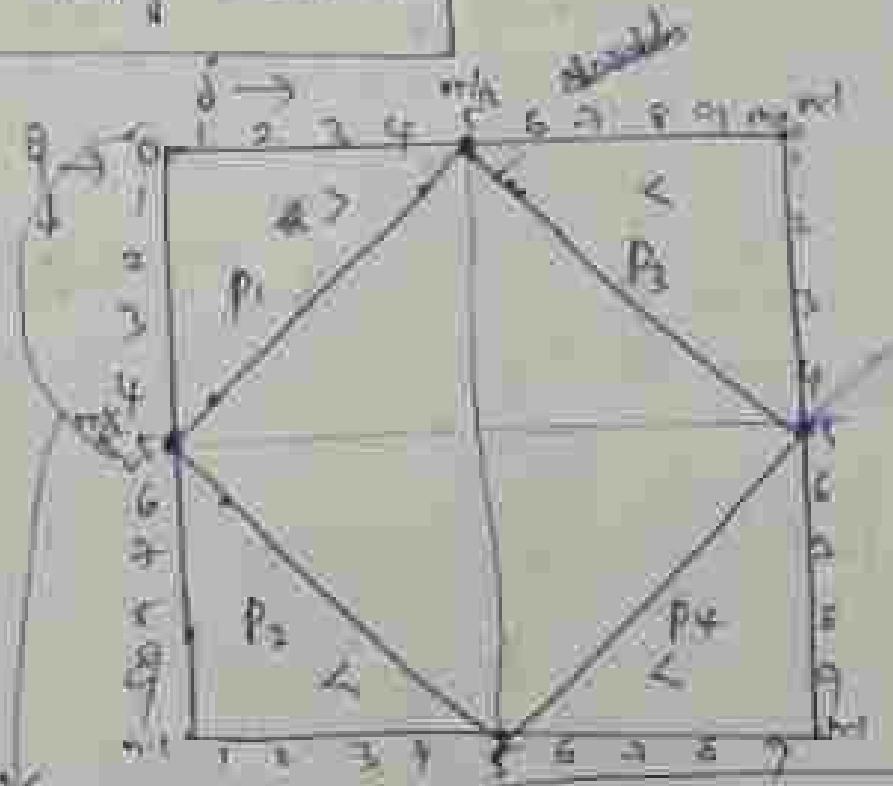


THE BODWYS

— Columbia —

→ $\text{d} = 1$

$$d_2 \circ \varphi + \psi = \eta - \psi$$



$$P_1 = \left| 1 + \left(\frac{z - i\omega_0}{2} \right)^2 \right|$$

$$0+5=5$$

0+5

55

So - point

$$P_i \ni j-i = \frac{100}{2}$$

$$P_4 \Rightarrow i+j = (n-1) + \frac{(n+1)}{2}$$

$$f + \frac{a}{j} = z \left(b + \left(\frac{a}{j} \right) \right)$$

Pattern program :-

```
int n=5;  
for (int i=0; i<n; i++) {  
    for (int j=0; j<n; j++) {  
        System.out.print(i);  
    }  
}
```

to print vertical same numbers we have to
print j.
to print horizontal same numbers we have to
print i.

Output :-

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

12345

Ops (concepts of class and objects)

(object oriented programming)

By using oops concept we can solve any
real world real time problem

(everything java principle)

Note :

- Software means collection of programs
- programs means set of instructions
- To write program set of instructions we
need programming language. In Java we are learning

① Class → Copy of object/blue print of object
Object → physical existence of any element we say
as object Ex: book, car, laptop etc ...
real time example : Book My show - class
object : person, ticket, show etc ...

• In OOPS while solving the problem we
need to first mark the objects

- Every object should have 2 parts
- i) class - point / fields / attributes (static information
of variables)
 - ii) class - point / behaviour (represent them as
methods)

Has point → indicates what it can hold
Does point → indicates what it can do

Ex: Student

1) Has point (Variables/identifiers)

int id;

String name;

2) Does point (method)

void play() {

logic

}

void study() {

logic

}

3) To represent an object, first we need have blue print of an object

what is blue print and how to represent it in java to represent a blue print we have ~~variable~~ key word "class".

Ex: Class Student {

int Sid;

String name;

void play();

void study();

}

Conventions followed by Java developers while writing a class is:

a. class name should be in " Pascal convention"

eg: Buffalo^{upper} Student^{upper} Gender^{upper}

b. Variable name should be in " CamelCase"

eg: name, Gender Buffo^{upper} Gender^{upper}
lower lower (our only compound word Capital)

c. Method name should be in " camelCase"

eg: play, toUpper(), lower()
upper upper
and word should start with Capital letter.

4) We use "new" keyword to create an object

for blue print (class)

ClassName Variable = new ClassName();

↓
datatype

(It is also called Reference

datatype means user
only defining datatype)

↓
Object
Creation

↓
Creating object
for className
by using new
keyword

method area	Object Heap area (dynamic memory available)	Stack area (local Variable & method)	be memory consistency problem of local variable which needs to be deallocated rather method code code of other language which is required for java would be available here
-------------	---	---	--

(x) (y)

2.8.6 Runtime to execute Java program

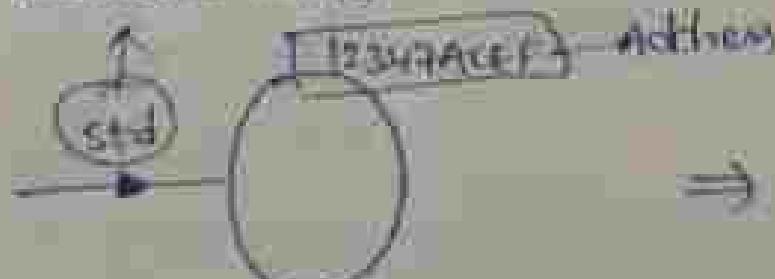
→ OS allocates this space to execute Java program

new ⇒ it is a signal to (JVM) to create some space for the object in the heap area

Tell the className, we inform the className, JVM creates object and sends the "hash code" to user

User should collect hash code through "reference variable"

Variable name



⇒ Student std = new
Student();

Object (Student)
(heap area)

5 Every object should always be in constant interaction

↳ Discuss later

6 Local object doesn't exit

Types of Variables

int a=10; (primitive data type)

Student std = new Student(); (reference data type)

primitive datatype / primitive Variables

primitive Variables can be used to represent primitive values

e.g. int x=10;

primitive = predefined by language and it is using some "Keywords" like byte, int etc.,

Reference Variables

Reference Variables can be used to refer object

e.g. Student std = new Student();

Instance Variables

If a Variables declare inside class out side method is known as instance Variable

If the value of the Variables changes from Object to Object then Such Variables are called "Instance Variables."

Ex:-

Student S₁ = new Student(); // id=10, name= Sachin

Student S₂ = new Student(); // id=11, name= dhoni

To know how many object are there, we can find by using `new`

ANSWERED - address



default values

`Int=0`

`float=0.0`

`String=null`

`boolean=false`

`Student (keapara)`
(Given)

`Char=' '`

to collect
garbage

Note:- Scope of Instance Variable would be available only when we have reference pointing to object. If the object reference becomes null, then can't access "Instance Variables".

Local Variables

Inside a method if any Variables are declared is known as local Variables memory would be allocated in Stack area.

local Variables default will not given by JVM
Values should give by programmer otherwise it
will leads to (Compile time error)

Static - Variable

If we use static key word with instance
Variable

Static int age;



JVM will allocate memory at class loading
time in heap area without object creation

Methods :- Stack area and heap area

→ any task heavy work
methods come to picture

→ ① name

ex: return type name (parameters)

② Input (parameters)

↑

③ body

Activity/Body

④ return type

↑

Methods we can access 2 types

⑤ with memory level.

→ where we are using doesn't matter to variable
where we declared it is matters.

rule 0 → It should have
name, parameters, body, return type

Method 1: Normal

(mandatory task / to account
that costs $\mathcal{O}(n)$

to stack space)

class Calculation

{
int a, b, c;

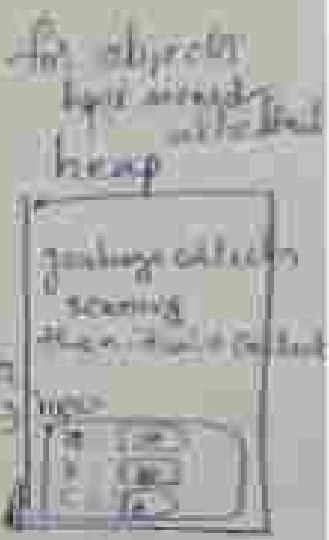
void add()

{
a = 0;

b = 20;

c = a + b;

System.out.println(c);
}
}



class LaunchCalc

{
public static void main(String[] args)

Calculation calc = new Calculation();

calc.add();

control with calc
add method

once method ends
it will delete

not enough space delete

Method 1

→ Two important data come one them
Stack and heap area

→ After compiling byte code given to
JVM

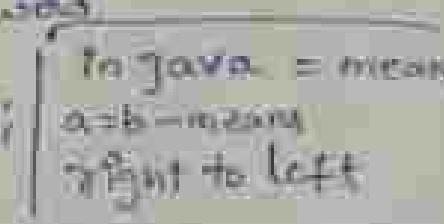
→ JVM to execute the body of
Main method. (JVM is calling main method)



Main method record will be created. That record
technically we call it as Stack frame.

First line in main method Object creation
is there so in heap area object will create
memory for all instance variables will allocate

right point over object creation with
Calculator cal = new Calculator();



Now left side point reference Variable cal is
there memory will be allocated in Stack frame
main method record. Right to left that address
of instance Variable referring to reference Variable

Next line add method is there then it will
go to add method. And again in stack area
One more method record will be created

what ever body it is it will execute
After task gets over that record will delete.
Control will come back again where
it went from add method

→ If there is no any task in main method
So main method record also will delete
What reference Variable also deletion condition
is referring to object.
So then in heap garbage collector keeps on scanning
Is there any thing if it finds any object which
is not referring then it will collect, memory
deallocated. (expiring, lifetime)
method & parameters

class Calculation

{

int res;

void add (int a, int b)

{

res = a+b;

s.o. p (res);

class LaunchCalc

public static void main (String [] args)

{

Calculation calc = new Calculation();

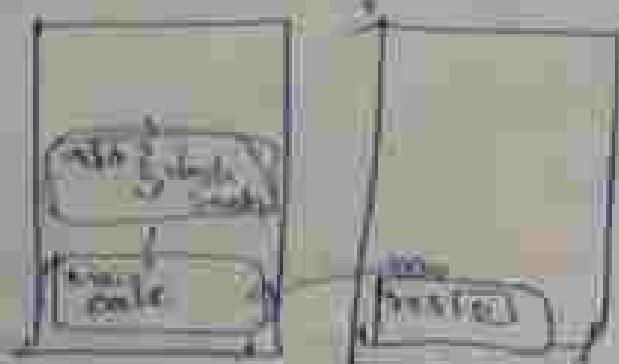
calc.add (10, 20);

scanning lifetime

When - Object is passing

values - parameter is passing

→ In method we are passing
accepting something
So when someone calling
method
who is called that fellow
should pass the parameters
which type method is
accepting.
like method is - example



Method 3 returning

→ class calculator

↓ int res, a, b;

int add() {

↑ a=10;

b=20;

res = a+b;

return res; } return

return [res] → syntax error is int

class Calculator {

public static void main (String [] args) {

Calculator calc = new Calculator();

return

int c = calc.add();

System.out,

System.out.println(c);

Method 4

both passing arguments
returning something also

class Calculator {

int a, b;

int add (int a, int b)

int res = a+b;

return res;

if this method expecting
anything and returning
something in int form
you when we call method
it will return when create
variable to store in input
↓
we can use it in any place

who is calling on the
it is not expecting any parameter
so result variable

giving you something
for calculation (adding)
except you it you
want take only one it

public class Calculator {

 public static void main (String [] args) {

 Calculator calc = new Calculator();

 int res = calc.add (10, 20);

 System.out.println (res);

}

}

(Garbage collector will deallocate memory in heap area.)

Method

Overloading :

(one is polymorphism)

⇒ Methods → stack area and heap area

method overloading

class Calculator

{

 int add (int a, int b) {

 return a+b;

}

 int add (int a, int b, int c) {

 return a+b+c;

}

 float add (float a, float b) {

 return a+b;

}

refers to provision of writing more than one method with same name and with different parameters with in

Same class

method overloading

(false polymorphism)

(one method

one functionality)

```
float add (float a, float b) {
```

```
    return a+b;
```

```
}
```

```
float add (int a, float b, float c) {
```

```
    return a+b+c;
```

```
}
```

```
double add (int a, int b, double c) {
```

```
    return a+b+c;
```

```
}
```

```
double add ( double a, double b, double c) {
```

```
    return a+b+c;
```

```
}
```

```
3
```

public class LaunchMode

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

```
    Calculator calc = new Calculator();
```

```
    int a=10, b=20, c=30
```

```
    float m=10.5f, n=20.5f, o=30.5f
```

```
    double x=10.5, y=20.5, z=30.5;
```

Calculator() method overriding

```
System.out.println (calc.add (a, b));
```

```
System.out.println (calc.add (m, n));
```

```
System.out.println (calc.add (x, y, z));
```

The moment calculated early binding
All the odd methods called compile time
Tweak expect two inputs polymorphism.

Based on
1) no of parameters
2) data type of parameter
3) either of data type parameter
compile time polymorphism

(to reduce complexity) (inbuilt methods also)

Compiler resolve the conflict
a) Number of par
overloading :-
nothing is overloaded
just illusion

name and parameters matters

Return type has no role to play, its only
method name & parameters.

method overloading with numeric type promotion
Implicit type casting

Can we overload main method in Java?

Yes, we can overload main method

however JVM will call such a main method
which accept (String[] args) — JVM will take up
second and third.

array → why arrays treated as objects
what
how to create use "new" keyword.

→ Variable → to store data int/

Want to store 5 integer values

int a; } "each variable store only value"

int b; } 5 Variables that are different

int c; Variable names

int d; so what if we want to

int e; store 20, 100 values with

different variable names "hot spot"

It's difficult to create & write, it's a lengthy process.

→ convenient (Traditional way to store data is to create Variable)

→ So Arrays came into picture.

Collection

of

data

→ If Array is indexed based data structure
to store large volume of data using single
variable name.

In Java Array store only ^{single} homogeneous
data

→ Array is in heap area.
because Java treat Array is object.

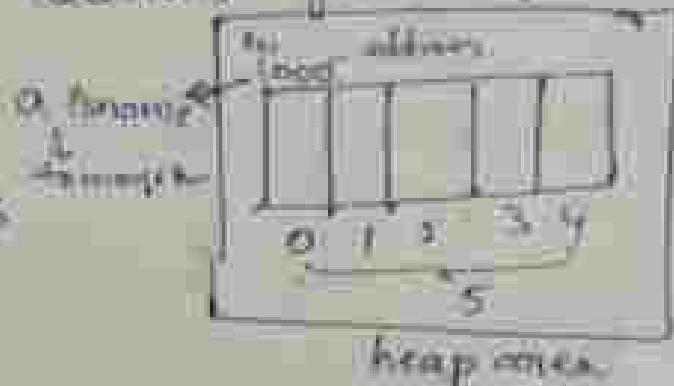
int arr = new int[5];

↓ ↑
datatype name Integers Integer type values

or its elements
of integers

The moment you say this to memory
locations given to you

10 → Students → marks
↓
int



int arr = new int[5];

or it's array of int

arr[0] = 100

arr[1] = 101



in static memory is a local

variable

10

System.out.println("arr[0]");

(Array can have two types (Regular → Jagged))

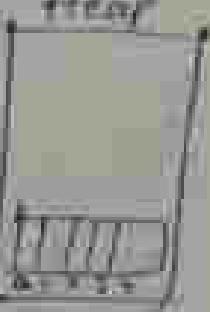
Case 2 → To store marks of 5 students

1D Array

Students →
5
int type
marks

int arr = new int[5]

or 5 elements of type integers



Case 2 To store marks of 3 classes, each with 3 students

3D array [3][regular array] 3D

3D array I have classes students

Regular -

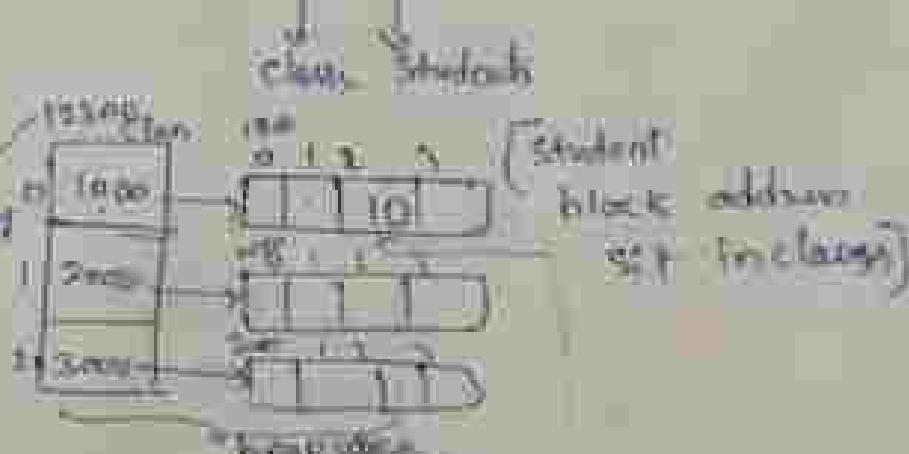


3D - array creation

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

memory map

as



as [0][0][0] = 10; → static data

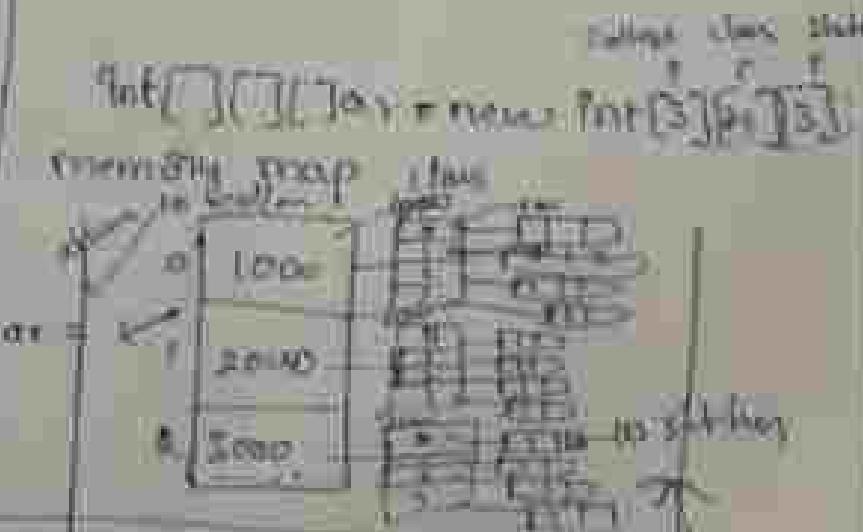
case 2

3 College 4 classes 3 students

3D - 3dimension array regular same data

College	classes	Students
0	obj[3]	3
1	obj[3]	3
2	obj[3]	3

as [2][2][2] = 10;



Case 3:

class	student	3D-Jagged Array
0	5	
1	3	jagged = random
2	4	not similar

to create

int arr = new int[3][]

↓

This much you write array
will not create

empty

the matrix have similar
keep writing

So

arr[0] = new int[5];

arr[1] = new int[3];

arr[2] = new int[4];

arr[0][4] = 10;

3D - joined array

String class

points

int arr[3][3] arr = new int[3][3][3];

it's not sufficient

arr[0] = new int[3][3];

arr[1] = new int[3][3];

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

arr[0][0] = new int[3];

arr[0][0][0] = new int[3];

arr[0][0][0][0] = new int[3];

arr[0][0][0][0][0] = new int[3];

arr[0][0][0][0][0][0] = new int[3];

→ Array Introduction

→ User Input from Console

```
class Scanner { → Import java.util package
```

```
nextInt(); } - many methods like these  
nextInt();
```

key board

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

Something into program

```
int a = scan.nextInt();
```

static

(arr.length to find array
length)

10 Array creation

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

```
class launch
```

```
{
```

```
public static void main (String [] args)
```

```
{
```

```
int [] arr = new int [5];
```

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

```
for (int i=0; i<5; i++) {
```

System.out.println ("please enter marks of student
at") + i + 1 + " : " + scan.nextInt();

```
System.out.println ("The marks of student");
```

```
for (int i=0; i<5; i++) {
```

```
System.out.println (arr[i] + " ");
```

Q) Array (Regular) by using for loop

class Student

e

i

a

o

11

22

33

44

Scanner sc = new Scanner (System.in);

int cls = sc.nextInt();

int std = sc.nextInt();

int a[][] = new int [cls][std];

for (int i=0; i<a.length; i++)

for (int j=0; j<a[0].length; j++)

scop ("Enter class "+i+" student "+j);

a[i][j] = sc.nextInt();

b

for (int i=0; i<a.length; i++) {

for (int j=0; j<a[0].length; j++) {

scop (a[i][j] + " ");

c

— — —

Same as in Jagged array also but
we will not define student while creating
array.

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

a[0] = new int [3];

a[1] = new int [2];

remaining same process

30 Jagged Array

int arr[3][4] = new int[2][3][4];

a[0] = new int[2][3];

a[1] = new int[3][4];

a[2][0] = new int[4];

a[2][1] = new int[3];

a[2][2] = new int[2];

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

a[2][4] = new int[4];

class

o

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

How Array Store in Java?

RAM is a Collection of Objects bytes.

Buffer Over Run

RAM



Here 3 locations are given if we access for 4th The values will be overwritten if called BufferOverRun for c.

In Java no BufferOverRun.

Because arrays are declared with a boundaries.

int a[] = new int[3];

a[0] = 10;

a[1] = 20;

a[2] = 30;

a[3] = 40;

Array Index out of boundary -
Runtime error / Exception

Creating array of objects

Class Fan

```
int cost;  
String brand;  
int noOfWings;
```

public class

```
public class Fan {
```

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

Fan [] f = new Fan [3];

values

class was created Creating int type 23.04.2019

f[0] = new	Fan();	
f[1] = new	Fan();	
f[2] = new	Fan();	

f[0].brand = "Glen"

f[0].cost = 10,000

Reverse

→ int [] arr = { 10,20,30,40 };

for (int i = arr.length-1; i>=0; i--)

{
 System.out.print(" ");

}

Disadvantages of array

- It can store only homogeneous type of data
- Length of array is fixed in size
(It cannot grow or shrink)
- int [] a = new int [3]; 
existing array will not increase
- another location will create
- Array demands contiguous memory locations
int a [] = new int [3];

1 byte for 2

= 12 bytes

Array will not make use of dispersed location

It uses contiguous memory location

It will count 0,1,2,3 like if in 4th place some value is there then it will again count 0,1,2 like after 3rd location. So this is disadvantage.

Another Way of Creating Array

int [] a = { 10, 20, 30 }; we create like this



for each loop

for (int ~~variable~~ name; ~~initialization~~ without start and end; ~~increment~~ named)

 s.o. print("variable");

Automatically it will take

}

10 20 30 - output

Objects \rightarrow Heap

copywriting



Strong code + IDE \Rightarrow Spring }

Position dependent
variable
loop

\rightarrow Enhanced for loop \rightarrow for each {

for (datatype Variable : ArrayName)

{

 s.o.p (variable)

}

Traversing / Iteration while travelling getting values

iteration = Variable - body

ab- iteration

but don't access them
as it will trigger bounds
- error in illegal direction

int [] a = {10, 20, 30, 40, 50, 60, 70, 80, 90};

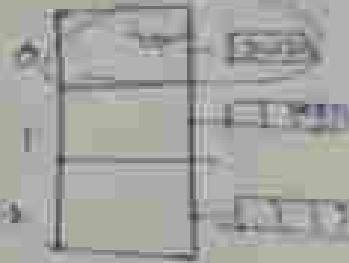
for (int element : a) {

 for (int arr : a) {

 s.o.p (arr);

}

iteration control



}

first one time do iteration

and and and and next time

Int C1, C2, ✓
int* a = new int[10]; ✓

int(6) arr; ✗ not valid

int arr[10]; ✓
int* arr[10]; ✓

int arr[10];
int* arr; ✓
int arr[10];
int arr[10];

int arr[10]; ✓

✓ is also valid
✓ is not valid

General: int C1(a, b, c);

a = new int[4]; ✓ Valid
a = new int[5];

int C1(x, y, z);
above code is valid

int m, n, o;
m=10; ✓ Valid
n=20;
o=40;

int C1(a, b, c);
✓

after, not) pass not allowed

size = long, float, double
int C1 a = new int[5];
↓

(new runtime) ✓ by short, int, char

Exception in Array

Compiler-Syntax

class Derived;

Derived d = new Derived();

✓

int() a = new int();

if class is not there. Can't create object.

Every type of array will have specific class
about: class

That class is not for programmers.

→ Arrays - So many methods are there

↳ copy

→ fill

↳ binarySearch

}

Utility class - Arrays

Static methods - without creation of objects
call method

Arrays.fill(a, s);

Setting

Arrays.set(a, i, v);

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

```
int sum = 0;
```

```
temp
```

```
for (int i = 0; i < a.length; i++) {
```

sum += a[i];

or

sum += a[i];

```
int [] a = {25, 35, 45, 75, 10};
```

```
int max = a[0];
```

```
for (int i = 1; i < a.length; i++)
```

```
{
```

if (a[i] > max)

{

max = a[i];

}

System.out.println(max);

1st Nov

Linear Search is used to search a key element from multiple elements. Linear Search is less fast than because it is slower than binary search and it is not

Searching

Linear Search

Sequential Search

Binary Search

1. Step 1: Traversing the array,
2. matching key element with array elements
3. if key found return index
4. if key found not found

whether an element is present or not in collection

using a linear search algorithm

if you want traversal

→ traversing / travelling

public class LinearSearch {

public static void main (String [] args)

{

int [] arr = {10, 20, 30, 60, 10, 50, 90};

boolean flag = false;

Scanner Scan = new Scanner (System.in);

System.out.println ("Enter the key to be searched");

int key = Scan.nextInt();

for (int i=0; i<arr.length; i++)

{ if (key == arr[i])

System.out.println ("Key" + key + "found at index" + i);

flag = true;

break;

{ if (flag == false)

System.out.println ("Key not found");

Binary Search -- Array should be sorted

1st Step - Array must be sorted

public class BBS {

```
public static void main(String[] args) {
    int[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90};
    Scanner Scan = new Scanner(System.in);
    int Key = Scan.nextInt();
    int low = 0;
    int high = arr.length - 1;
    while (low <= high) {
        int mid =  $\frac{low+high}{2}$ ;
        if (key == arr[mid]) {
            System.out.println("Key " + key + " Found at index " + mid);
            break;
        } else if (key < arr[mid]) {
            high = mid - 1;
        } else if (key > arr[mid]) {
            low = mid + 1;
        }
    }
    if (low > high) {
        System.out.println("Key not found");
    }
}
```

Bubble sort

public class LaunchRS

public static void main (String[] args)

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

for (int i=0, len=a.length; i++)

{

for (int j=i+1; j<a.length-1; j++)

{

if (a[j]<a[j-1])

{

int tempVar = a[j]

a[j] = a[j-1]

a[j-1] = tempVar,

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

}

logic for bubble sort

Strings in Java

- A String Variable Contains a collection of characters surrounded by double quotes
- If it is classname, String is basically an inbuilt class for which object can be created (java.lang).

State of String (java.lang.String)

[package]

Java

↳ lang

↳ String

String = Collection of characters

String = "Vijay"; — ① type

String s₁ = "Vijay"; ②

String s₂ = new String ("Vijay"); ③ type

Java.lang.String

→ String is refers to an object in java present in package called java.lang.String(c)

→ String refers to Collection of characters

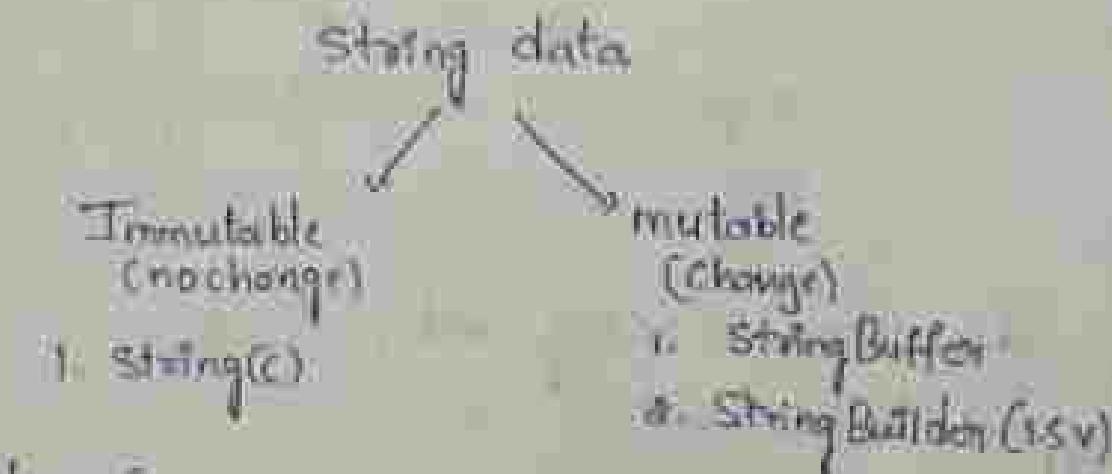
eg:- String s = "Sachin";

System.out.println(s); //Sachin

String s = new String ("Sachin");

System.out.println(s); //Sachin.

In Java, String Object is by default immutable, meaning once the object is created we cannot change the value of the object. If we try to change then those changes will be reflected on the new object not on the existing object.



class {

// Instances Variables

// methods

}

Case 1 :

String s = "Sachin";

s.concat(" tendulkar "); // new object got
(created with modification
so
Immutabile)

System.out.println(s);

Output : Sachin

(Vs)

StringBuilder sb = new StringBuilder("Sachin");

sb.append(" tendulkar "); // for the same object

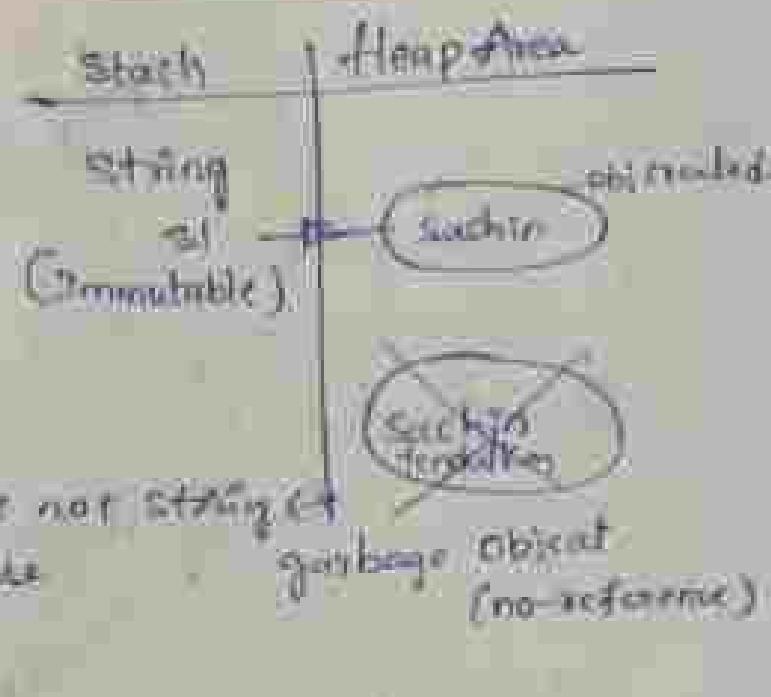
System.out.println(sb);

(modification so mutable)

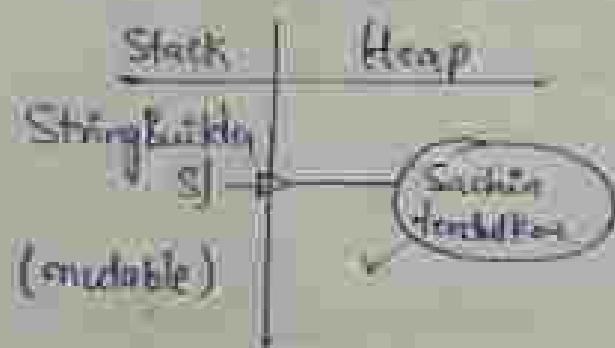
Output : Sachintendulkar

Case 1 - Memory map

```
String s1 = "Sachin";  
s1.concat(" tendulkar");  
System.out.println(s1);  
//Sachin
```



```
StringBuilder s1 = new StringBuilder("Sachin");  
s1.append(" tendulkar");  
System.out.println(s1); //Sachin tendulkar
```



Case 2

```
String s1 = new String("Sachin");
```

```
String s2 = new String("sachin");
```

```
System.out.println(s1==s2); //false
```

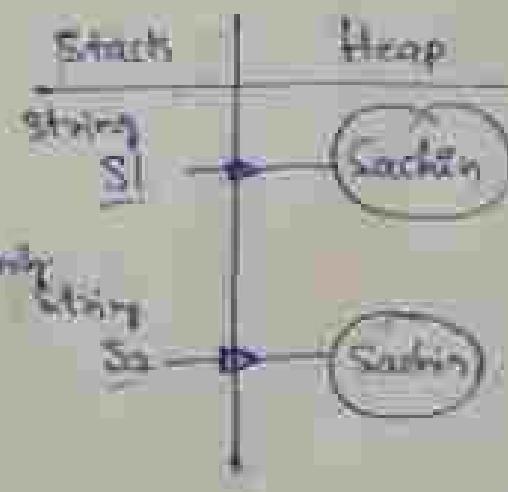
```
System.out.println(s1.equals(s2)); //true
```

⇒ `String Class.equals` method will compare the content of the object if same return true otherwise return false
(vs)

StringBuilder Sb1 = new StringBuilder ("Sachin");
 StringBuilder Sb2 = new StringBuilder ("sachin");
 System.out.println (Sb1 == Sb2); // false
 System.out.println (Sb1.equals (Sb2)); // false
 \Rightarrow StringBuilder class equals method do reference
 Comparison if different object returns false, even
 if different object returns false.

Case 8: memory map

String S1 = "Sachin";
 String S2 = new String ("Sachin");
 System.out.println (S1 == S2); // false
 System.out.println (S1.equals (S2));



String class.equals()

will compare the content present inside the string
`"Sachin".equals ("sachin")`
true

StringBuilder S1 = new StringBuilder ("Sachin");
 StringBuilder S2 = new StringBuilder ("Sachin");



StringBuilder class.equals() compare the content (addition object)
 not the content of StringBuilder.
false.

Case 3 : String s = new String("Sachin");

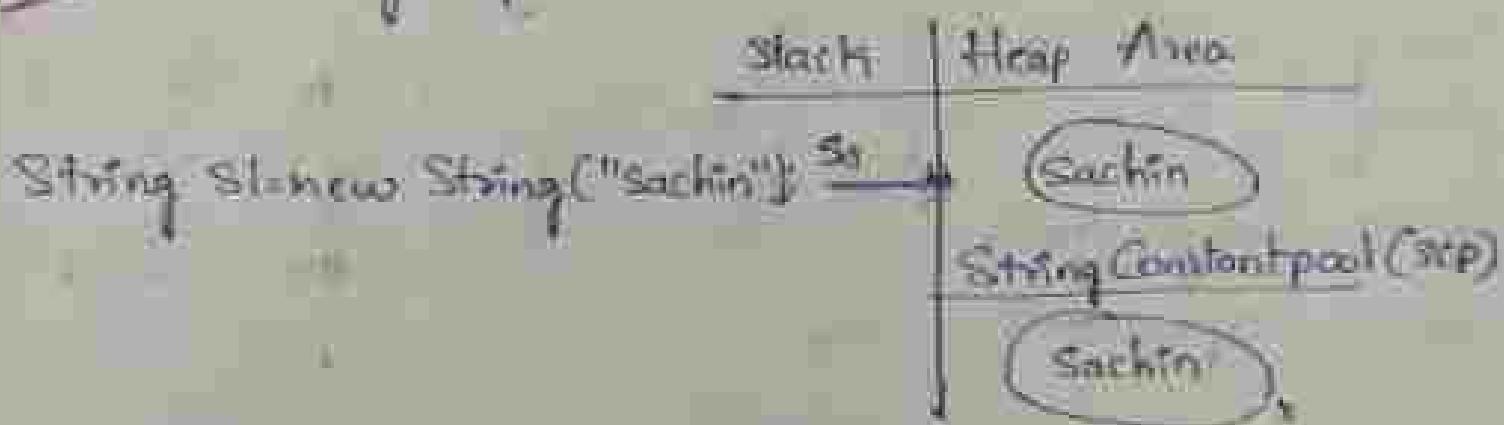
In this case 2 objects will be created one in the heap and the other one in the String Constant pool, the reference will always point to heap (s)

String s = "Sachin";

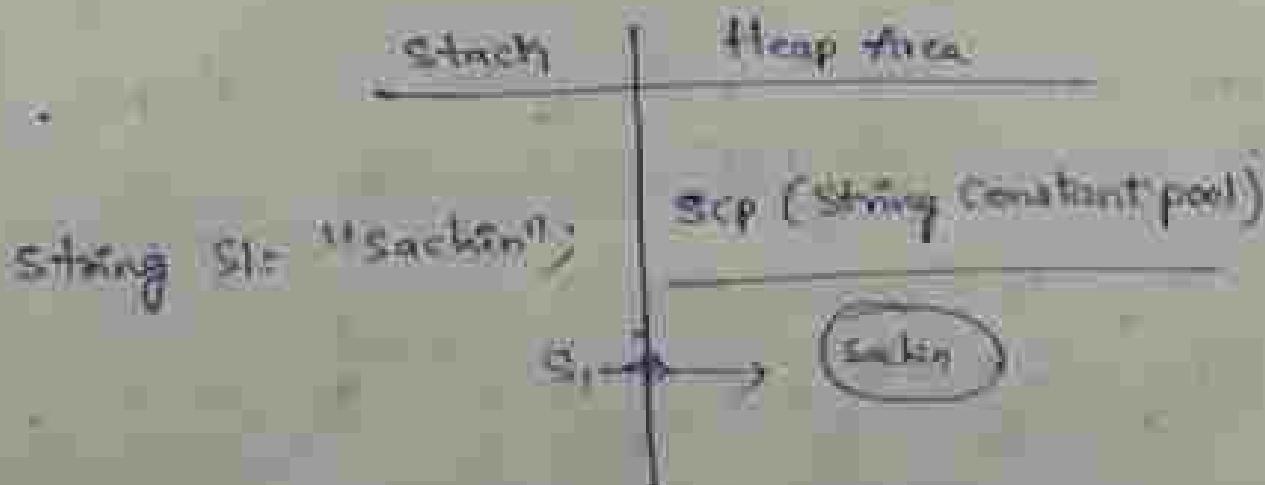
In the case only one object will be created into the Sop and it will be referred by our reference.

case 3

Memory map



Even though it is a garbage object Garbage Collected will clean this data present in Sop



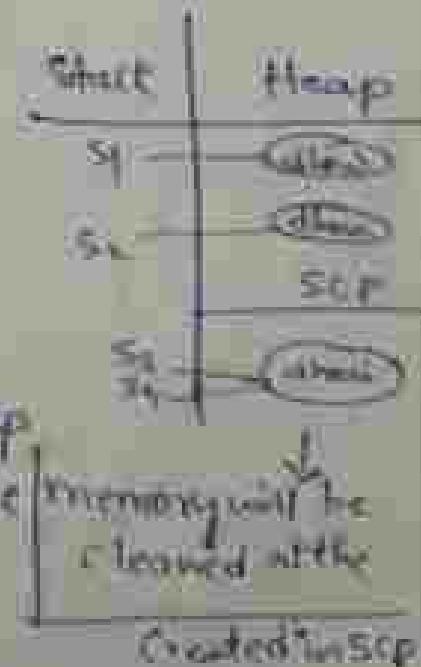
Note : Object Creation in SCP is always optional. Jvm will check if any object already created with required content or not. If it is already there then it will reuse the existing object instead of creating the new object. If it is not available only then new object will be created. So we say in SCP no chance of creating objects with some content. In SCP duplicates will not be allowed.

Garbage collector cannot access SCP area. Even though object does not have any reference still object is not eligible for garbage collection. All SCP objects will be destroyed only at the time of Jvm shutdowns.

Ex:

```
String S1 = new String ("dhonei");
String S2 = new String ("dhonei");
String S3 = ("dhonei");
String S4 = ("dhonei");
S1.equals(S2); // True
S1.equals(S4); // True
```

Two objects are created in the heap with data as S1, S2. One object is created in SCP with data as S3, S4.



Case 4

String S = new String ("Sachin");

S.concat ("tendulkar");

S = S.concat ("IND");

S = "Sachintendulkar";

S = op(S);

Output = Sachintendulkar.

Concat method will create object in Scp

A method call will be by JVM.

Because of runtime operation if new object is created
it will create in Heap area not in Scp.

Direct literals are created in Scp. Because
of runtime operation if object is required to create
compulsory that object should be placed on heap
but not in Scp.

String S1 = new String ("Sachin");

S1.concat ("tendulkar");

S1 = S1 + "IND";

String S2 = S1.concat ("IND");

S = op (S1);

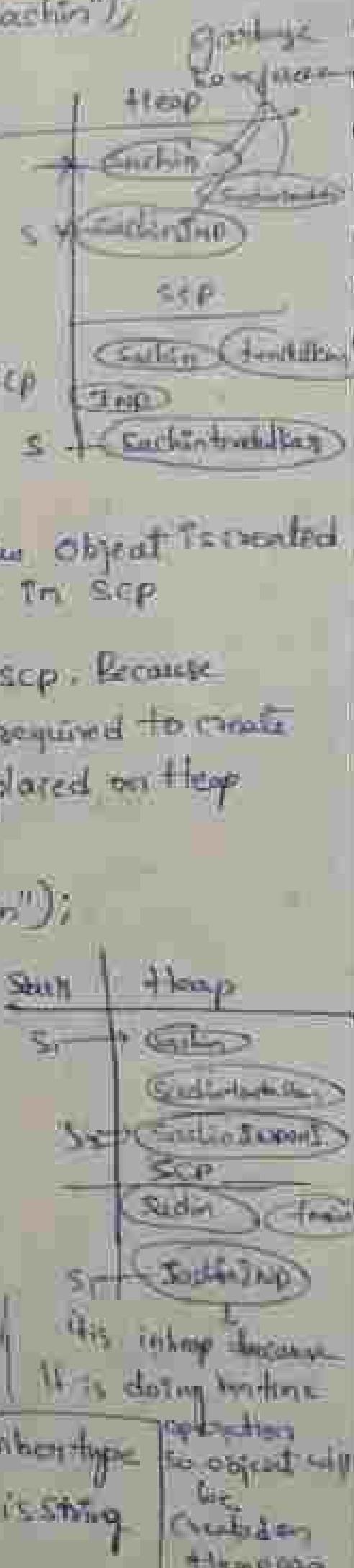
S = op (S2);

S1 = S1 + "IND";

String + "IND"

+ = addition both operands are number type

edit -> Concatenation if one operand is String



String s₁ = new String ("Sachin");

String s₂ = "Sachin";

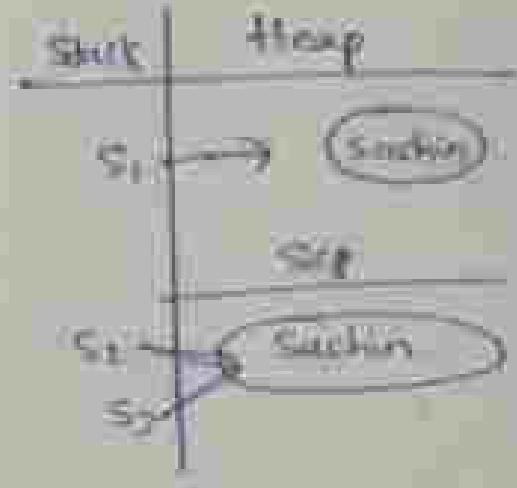
s₁ == s₂;

false

String s₃ = "Sachin";

s₁ == s₃;

true



Interning → Using Heap object reference, if we want to get corresponding Scp object otherwise need to use intern() method.

String s₁ = new String ("Sachin");

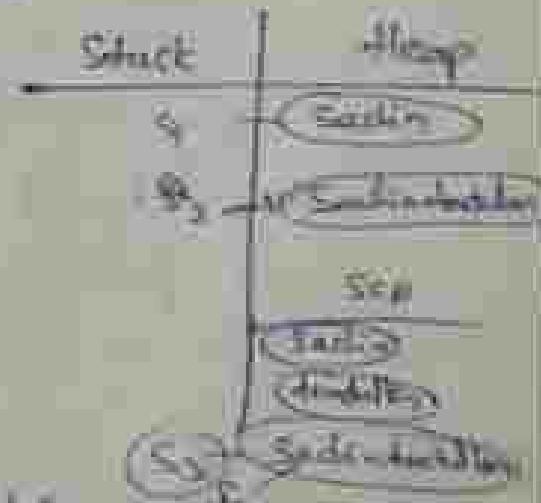
String s₂ = s₁.concat (" tendulkar");

String s₃ = s₂.intern();

String s₄ = "Sachin tendulkar";

(s₃ == s₄)

true



using heap object reference → You can get reference on Scp object and if object does not exists then intern() will create new object in Scp and it returns.

Importance of Scp

Advantages of Scp Application

Name	<input type="text"/>
F. Name	<input type="text"/>
DOB	<input type="text"/>
City	<input type="text"/>
Phone	<input type="text"/>
<input type="button" value="Submit"/>	

Unmutable

Scp Area



- i) In C/C++ program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content it reduces performance of the system and affects memory utilization.
- ii) we can create only one copy and use twice the same object for every requirement. This approach involves performance and memory utilization we can achieve this by using "Scp".
- iii) In Scp several reference pointers to same object are main disadvantages in this approach is by using one reference if we are performing any change, the remaining references will be impacted to overcome this problem Sun microsystem implemented Unmutable concept for String object.

7) According to this we can create an object we can't perform any change in existing object if we trying to perform any change a new object will be created hence immutability is main disadvantage of sep.

API level (Application programming Interface);
Some one wrote the code how he will give class file and users will use and like the benefit. Entire java we are learning as API only
if some class name & some method name is known.

Methods of a string (Important)

1. public char charAt (int index)
2. public String concat (String str)
3. " boolean equals (Object o)
4. " boolean equalsIgnoreCase (String s)
5. " String substring (int begin)
6. " " (int begin, int end)
7. " int length ()
8. " " String replace (char old ; char new)
9. public String toLower Case ()
10. " " " upperCase ()
11. " String trim ()
12. " int index of (char ch)
13. " int lastIndex of (char ch) .

String class constructor:

String s = new String(); \Rightarrow Create an empty String object

String s = new String("String literal");

\hookrightarrow Creates an object with string literals on heap.

String s = new String(StringBuffer sb); \Rightarrow

Creates an equivalent String object from StringBuffer.

String s = new String(Character ch); \Rightarrow Creates an equivalent String object for char array

String s = new String(Character b); \Rightarrow Creates an equivalent object for byte array

\hookrightarrow ch() ch = { 'S', 'o', 'v', 'a' }

String s = string(ch);

s.o.p(s); Java

Methods of String :-

① public char charAt(index)

\hookrightarrow String s = "Sachin";

s.o.p(s.charAt(0)); its

s.o.p(s.charAt(0)); if String Array out of bound

s.o.p(s.charAt(10)); if String throw ^{exception} out of bounds exception.

② public String concat (String str)

eg: String s = "Sachin";

s. o/p (s.concat (" tendulkar "));

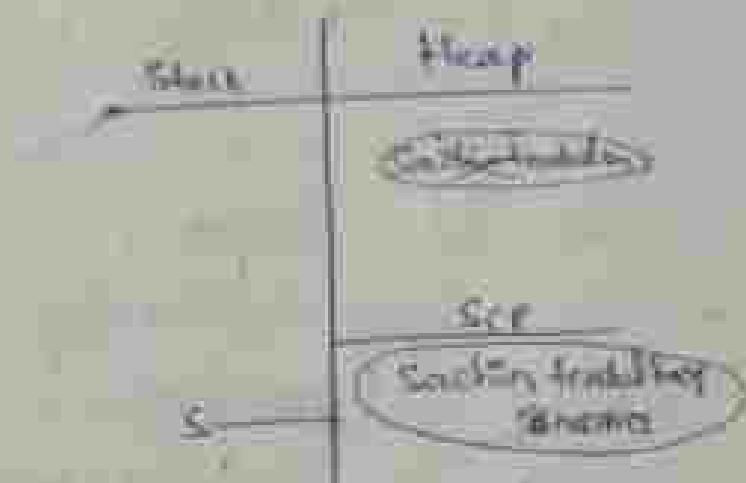
str = "IND";

so s4 "IND";

so p(s);

1) Sachintendulkar

2) Sachin tendulkar@IND



③ public boolean equals (object o)

It is used for Content Comparison. In String class, this method is overridden to check the content of object.

④ public boolean equalsIgnoreCase (String s)

It is used Content Comparison without Comparison the case.

eg: String s = "Java";

s. o/p (s.equals ("Java"));

s. o/p (s.equalsIgnoreCase (s.equals ("Java")));

false

True

⑤ public String substring (int begin)

It gives the string from begin index to the end of string.

String s = "mission";

s. o/p (s.substring (2)); // o/p: "ission" - Search from 2 to end.

④ public string substring (int begin, int end);

It gives the string from begin index to end-1 to string
String s = "Vinay";

S.o.p (s.substring (2,6)); // two Scanning towards
will happen

⑤ public int length();

It give length of string

String s = "Vinay";

S.o.p (s.length()); // 6

⑥ public string replace (old char, new char);

String s = "Vinay";

S.o.p (s.replace ('e', 'a')) // Vinay

String s = "ababab";

S.o.p (s.replace ('a', 'b')) // bbbbbb

⑦ public string to lowerCase();

⑧ public string to UpperCase();

String s = "VINAY";

S.o.p (s.toUpperCase()), // Vtay

S.o.p (s.toLowercase()), // Vinay

After the printing the JVM have control then no gc
will collect that object

11) public String trim()

To remove the blank spaces present at the beginning and end of string but not the blank spaces present at middle.

e.g. String name = "Viney Kumar";

s.o.p (name.trim()); // Viney Kumar

String name = "Viney Kumar";

s.o.p (length()); // 13

s.t.s.o.p (name.trim()); // VineyKumar - runtime operation

s.o.p (name.length()); 13

public int indexOf(char ch)

it returns the index of 1st occurrence of specified character if the specified character is not available then it returns -1

String s = "Sachinrajan";

s.o.p (s.indexOf('a'));

s.o.p (s.indexOf('z'));

public int lastIndexOf(char ch)

it returns the index of last occurrence of specified character if the specified character is not there then it returns -1

String s = "Sachinrajan";

s.o.p (s.lastIndexOf('a'));

s.o.p (s.lastIndexOf('z'));

public String toString();

When ever we print any reference, by default
JVM will call toString() on the reference to print value.

public Student {

String name = "Sachin";

int id = 10;

Local variable
is being
accessed

Student std = new Student();

s.o.p (std); // student @ hexadecinal

s.o.p (std.toString()); // student @ string

String s1 = "Sachin";

String s2 = s1.toString();

s.o.p (s1 == s2); true



String s1 = new String ("Sachin");

String s2 = s1.toString();

 s3 = s1.toUpperCase();

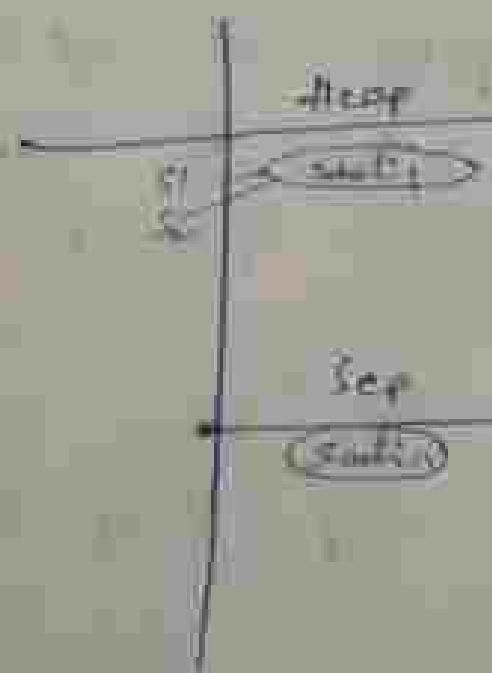
 s4 = s1.toLowerCase();

 s5 = s1.toUpperCase();

 s6 = s1.toLowerCase();

s.o.p (s1 == s6) false

s.o.p (s3 == s5); true



Mutable → we can change and changes will be reflected in same memory.

```
final String Buffer sb = new String Buffer ("Stackin");
sb.append (" tendulkar");
System.out.println (sb);
```

final String Buffer sb = new String Buffer ("tendulkar");
By using keyword we cannot use same reference to another object but we can change the content in final reference bcoz String Buffer is mutable.

final Vs Immutability

- final is a modifier application for Variable where as immutability is applicable only to objects.
- If reference Variable is declared as final it means we cannot perform reassignment for reference Variable. It does not mean we cannot perform any changes in that object.
- By declaring a reference Variable as final we can't get immutability nature.
- final and immutability is different concept

```
eg: final String Builder sb = new String Builder ("");
sb.append ("tendulkar");
System.out.println (sb);
```

sb = new String Builder ("tendulkar");
Cannot assign a value to final Variable

Note : - final variable (read-only), final object (immutable)
non-mutable variable (read-only)

StringBuilder, StringBuffer and all wrapper classes.

ByteBuf muttable

mutable \rightarrow can be changed

immutable \rightarrow can't be changed

String Buffer : if the content will change frequently then it is not recommended to go for string object because for every new change new object will be created

2. To handle this type of requirement we have String Buffer/StringBuilder concept

Construction :

```
StringBuffer sb = new StringBuffer();
s.o.p (sb.length()); //s
```

```
s.o.p (sb.capacity()); //s
```

```
s.b.append ("abedfghijklmnp");
```

```
s.o.p (sb.length()); //s
```

```
s.o.p (sb.capacity()); //s
```

```
sb.append ("q");
```

```
s.o.p (sb.length()); //s
```

```
s.o.p (sb.capacity()); // (old capacity + 1)  $\Rightarrow$  new capacity
```

StringBuffer sb = new StringBuffer();
Creates an empty StringBuffer object than default
Initial Capacity is 16

Once StringBuffer reaches the maximum capacity a new
String will be created
new Capacity = (Current Capacity + 16) \rightarrow ^{Capacity}

2. StringBuffer sb = new StringBuffer(^{Initial} int capacity);
Creates an empty String with Specified initial Capacity

Ex-1. StringBuffer sb = new StringBuffer(17);
s.o.p (sb.capacity()); \rightarrow 17

3. StringBuffer sb = new StringBuffer (String s);
Creates a StringBuffer object for given String with
the Capacity = s.length() + 16;

Ex-2. StringBuffer sb = new StringBuffer ("Sachin");
s.o.p (sb.length()); \rightarrow 6 String length
s.o.p (sb.capacity()); \rightarrow 22 6 + 16 = 22
Default value
of capacity for String
buffer

Methods

```
public StringBuffer charAt();  
String sb = StringBuffer ("Viney");  
sb.charAt(3);  $\rightarrow$  V  
sb.charAt(10);  $\rightarrow$  null
```

2) `public StringBuffer setCharAt();`
`StringBuffer sb = new StringBuffer("Vivay");`
`sb.setCharAt(5, 'A');`
`s.o.p(sb); // VivayA`

3) `public StringBuffer append (String s)`

4) `public StringBuffer append (int i)`

5) `public StringBuffer append (boolean b)`

6) `public StringBuffer append (double d)`

7) `public StringBuffer append (float f)`

8) `public StringBuffer append (int index, Object obj)`

9) `public StringBuffer append (Object obj)`

Ex- `StringBuffer sb = new StringBuffer();`

`sb.append("Pi Value is :- ");`

`sb.append(3.1415);`

`s.o.p(sb); Pi Value is :- 3.1415`

Append method is overloaded method
method name is same but change in
argument type

int set (overloaded method)

public StringBuffer insert (int index, String s)

5) `public StringBuffer insert (int index, Object obj)`

6) `public StringBuffer insert (int index, long l)`

7) `public StringBuffer insert (int index, double d)`

8) `public StringBuffer insert (int index, float f)`

9) `public StringBuffer insert (int index, Object obj)`

To insert the string at specified position we use
insert method

Ex:- String Buffer sb=new String Buffer ("Viney");
sb.insert (3, 'i');
S.o.p (sb);

Method delete

public StringBuffer delete (int begin, int end);

it deletes the character from specified index to end-1

public StringBuffer delete (char c (int index);

it deletes the character at specific index

Ex:- StringBuffer sb=new StringBuffer ("Viney");
sb.delete (2, 5);

sb.delete (char At);

S.o.p (sb); // Vny

reverse (it is used to reverse given string)

StringBuffer sb=new StringBuffer ("Viney");

sb.reverse ();

S.o.p (sb); // yeniv

length (it is used to consider only specified no
of character and remove all remain char)

StringBuffer sb=new String Buffer ("VineyKumar");

sb.setLength (6);

S.o.p (sb);

Trim to size :- used to allocate the extra allocated free memory such that capacity and size are equal.

StringBuffer sb = new StringBuffer(1000);

sb.append("Gagan");

sb.append("Gagan");

sb.append("Gagan");

sb.trimToSize();

System.out.println(sb);

public void ensureCapacity(int capacity)

It is used to increase capacity dynamically based on our requirements.

e.g. StringBuffer sb = new StringBuffer();

sb.append("Gagan");

sb.ensureCapacity(1000);

sb.append("Gagan");

StringBuffer

→ pointer which
→ points to memory
→ using object



StringBuffer(1.0V)

Capable

Counting
Semaphore

Thread Safe

StringBuffer



StringBuffer
(V.V)

Non-Synchronous

Not Thread Safe

String Buffer :-

Every method present in String Buffer is Synchronized so at a time only one thread can allowed to operate on String Buffer Object. It could create performance problem. To overcome this problem we should go for String Builder.

String Builder (1.5v)

String Builder is same as String Buffer (1.0v) with few differences.

String Builder

No methods are Synchronized

At a time more than one thread can operate it is not thread safe.

Threads are not required to wait so performance is high. Introduced in Java 1.5 version.

String → the opt of the content is fixed and it won't change frequently.

String Buffer → we opt if content changes frequently but thread safe is required.

String Builder → we opt if content changes frequently but thread safety is not required.

Method Chaining

most of the methods in String, StringBuilder, StringBuffer return the same type only hence after applying method on the result we can call another method which form method chaining.

e.g:- StringBuffer sb = new StringBuiller();
sb.append("Sachin").insert(6, "Tendulkar").reverse()
append("IND").delete(0, 4).reverse();
S.o.p (sb);

toString() = method.

class Student {

String name = "Sachin"; } - User defined class
int id = 10;

main() {

Student std = new Student(); // User defined object
S.o.p (std); // (Sachin10) (Sachin@Home10)

Sachin & 10

String name = new String("Akash");
System.out.println(name); // Akash

when we trying to print Variable
JVM will call `toString()` call back
method.

to String return type

public String toString()
return " ";

3

when ever we print any reference, by default JVM
will call `toString()` on the object only ~~with saying~~

toString() \longleftrightarrow return
using to String method

student toString method will called.

if we don't override `toString` method()

it will call Object class `toString` method.

Object class `toString()` will never print value
of annotated with `Instances Variable`.

to print the `Instances Variables` we have
to call `toString` method and override that
method.

return type is `String`.

else

it will not print & value.

Ex Student object name with ~~long~~ ^{long} ~~final~~ ^{final} ~~value~~ ^{value}.

Object oriented programming

- 1) class → Encapsulation
- 2) Objects → Inheritance
- 3) Methods → polymorphism
- 4) Methods → Abstraction

- 1) privacy (or) Security → Data hiding (or) Binding {Encapsulation}
- 2) Code Reusability → (Inheritance)
- 3) Code flexibility → (polymorphism)
- 4) Implementation hiding → feature visibility → Abstraction

① Encapsulation

- Data hiding
- Data hiding
- providing Security
- providing controlled access to datamembers

(private, setter, getter, this keyword, shadowing)

```
class Student {
```

```
    int age;
    String name;
    String city;
```

Instance Variable / Data Member /
fields / properties

private → To provide security - for instance
Variables, i.e. can access in same class
But outside class we can't access
Instantiation → Creation of object.

Setter → if a method is doing activity of receiving
data from outside and setting it to its data member

getter → if a method is doing activity of return
value. anyone wants to use a value from outside
of private class we can use Get method name

data binding: For exam Variable we have
method. And Variable, age - one working together
In a class which has all data members are
private is called Bean.

Shadowing problem → whenever there is a name
conflict b/w instance Variable and local Variable
is called Shadowing problem.

To solve this problem we can use (the) Key word.

e.g. Class Student {

private int age;

private String Name;

private String City;

```

Void SetName (String Name) {
    this.name = Name;
}

String getName() {
    return name;
}

Void SetAge (Int age) {
    this.age = age;
}

Int getAge() {
    return age;
}

```

```
Void SetCity (String City) {
```

```
    this.city = city;
```

```
String getCity() {
```

```
    return city;
```

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

```
    St.setName ("Vijay");
```

```
    St.getName ();
```

```
    System.out.println (St.getCity());
```

Output: Vijay

→ right click
 → go to source
 → generate setters and getters
 by selecting all
 → it will generate setters and
 getters by automatically

Getters & Syntax

get property name if it should not have any parameters e.g. int getAge() {
method signature
followed by variable name}

Setters : Set property name if it can have parameters

```
Void setAge(int age),  
this.age = age;
```

public : within entire project we can access any variable by using public keyword it increase accessibility

```
public boolean isMarried() {  
return married; } } - boolean typegregation
```

this → which will hold reference of currently executing object

Class → Create many objects

this : refers to current object in a method(s)
Constructor The most common use of this keyword is to eliminate the confusion b/w class attributes and parameters with same name.

Class Student {

 int age;

 public void SetAge (int age)

 this.age = age;

 }

 parameter

(int) local variable

 parameter (int) local variable

 } instance variable

Common Setter we have have, not Common Getter
because return type is different i.e.

Common setter :-

public void SetData (String name, int age, String city) {

 this.name = name;

 this.age = age;

 this.city = city;

}

Constructor :- class name and method name is same

Class Student {

 int age;

 String name;

 public Student (String name, int age) {

 this.age = age;

 this.name = name;

}

 }

 public void main (String [] args) {

 Student1 std = new Student1 ("Rohit", 12);

Constructor will not have return type. Constructor will be called while creating object.

Student1 std = new Student();

↳ Constructor call.

Constructor need to Initialization not of

Object Creation

When object creation is happening JVM will called Constructor if any present in class.

If constructor method is not there in class by default JVM have constructor method and JVM will automatically include default constructor with 0 parameters.

Student1 std = new Student("Vinay");

Student2 std2 = new Student();

Whenever there is a called to constructor if programmer is not specified any constructor that time JVM will include default constructor.

And if programmer specified any one constructor in program in that case JVM will not include constructor.

Overloading : Same constructor name but different parameters.

Super(); → it will call to parent class

Constructor Chaining → Calling one constructor to another constructor.

Stack overflow

disp()

disp()

3

Stack



all stack area got full then it leads to
(Stack overflow)

If we call method in same method is called Stack overflow

```
Ex: void disp() {
    disp();
}
```

Class Dog

private age;

String name;

public Dog

 this ("Snow", 999);

Dog (String name, int age) {

 this.name = name;

 this.age = age;

}

public int getName() {

 return name;

}

```
public String getName() {
    return string;
```

3

main() {

Dog ob = new Dog();

ob.getName();

ob.getAge();

Dog ob = new Dog("Vivek");

ob.getName();

ob.getAge();

Constructor : Same name as class name.

Constructor parameters will work like method parameters will work.

Constructor is called during object creation
(or)

instantiated (creation of object).

return statement is invalid.

Constructor will not have return type explicitly.

Inside constructor first statement Super() or this()

Super() → Call parent constructor.

this() → calls constructor of same class.

Constructor Chaining → one constructor calling another constructor.

Both this() & Super() in same constructor.

Can write this() or Super() → apart from first statement. (⊗) Not Valid.

Purpose : If some statement has to be executed the moment we create object we can write inside constructor.

this() → inside a one constructor =>
Requirement 1

→ To another body of constructor.

this keyword
refer to current
object

this()
method
it will call same class
constructor

constructor overloading :- more than one constructor
but different parameters

Methods
call explicitly by
calling name
return type
explicitly
Inside method return
statement is allowed

Constructor
when object is created
constructor is called
no explicit return type
no return statement

Static keyword :- non-access modifier used
for methods and attributes.

Static keyword

class
Static Variables
Static Block
Static method

non static Variables (or) instance
Variables in same Java Block

Virtual method (m) non-static method -

If we create static Variables, we can use inside static block and static method and we can't use in non-static block, non-static method if we create non-static Variables (instance Variables) we can use in non-static Java block ~~and we can use in non-static Java block and non-static method~~ But we can't use in static method and static block.

Instance Variables memory allocated inside object first we need to create object. Inside object normally for instance Variables are allocated we can say object dependent also we can't use in static method or static block because static method, block are object independent

Class Demo

Static int a;

Static int b;

```
static {  
    a=10;  
    b=20;  
}
```

```
Static void disp()
{ o.p(a);
  o.p(b); }
```

```
int x;
```

```
int y;
```

```
x= 45;
```

```
y= 50;
```

```
----- Demo ()
```

```
void odd()
{ o.p(x);
  o.p(y); }
```

```
main()
{ Demo d;
```

```
d.disp(); }
```

we can call static method without object
to object dependent

```
Demo d= new Demo();
d.odd(); }
```

In a class if all elements are three then

- 1) Static Variable → ①
- 2) static Block → ②
- 3) static Method → ③
- 4) non-static block → ④
- 5) non-static Variables → ⑤

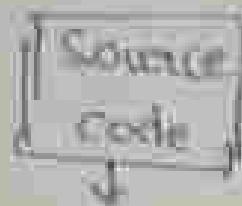
It will start execute as follows:

① → ② → ③ → ④ → ⑤

④ Constructor → ⑥

⑤ Non - static method → ⑦

→ Java program



Java



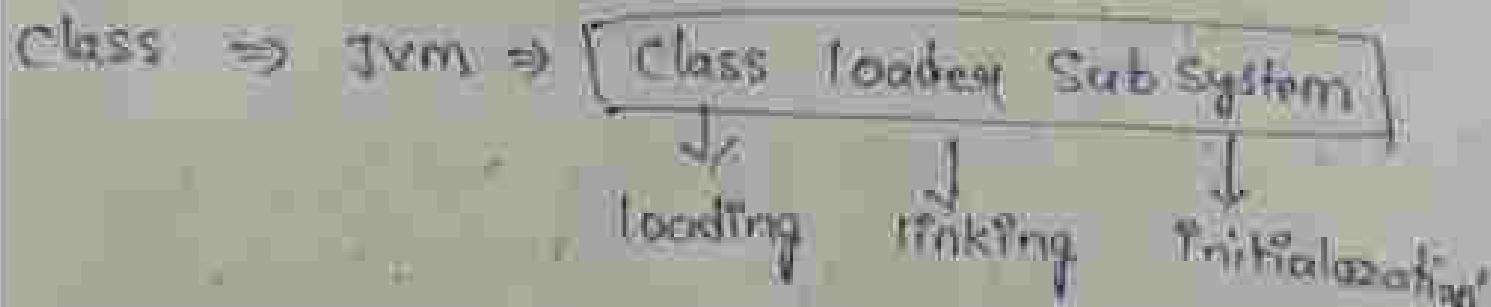
JVM → VM has to execute



JVM → ① Class Loader Subsystem

② Data Areas

③ Execution will happen (Interpreted or Compiled)



From method area a byte code file will take to class loader system.

loading → To load the code class in general

Boot Strap Loader → It will load all API Classes
& all in built classes like Scanner class (or) String class

Extension of Application Loader → It will load all
classes which will written by programmer

Linking :-

Verification :- Byte code verifier it will check
No one is trying to harm all byte codes one
connect it will check

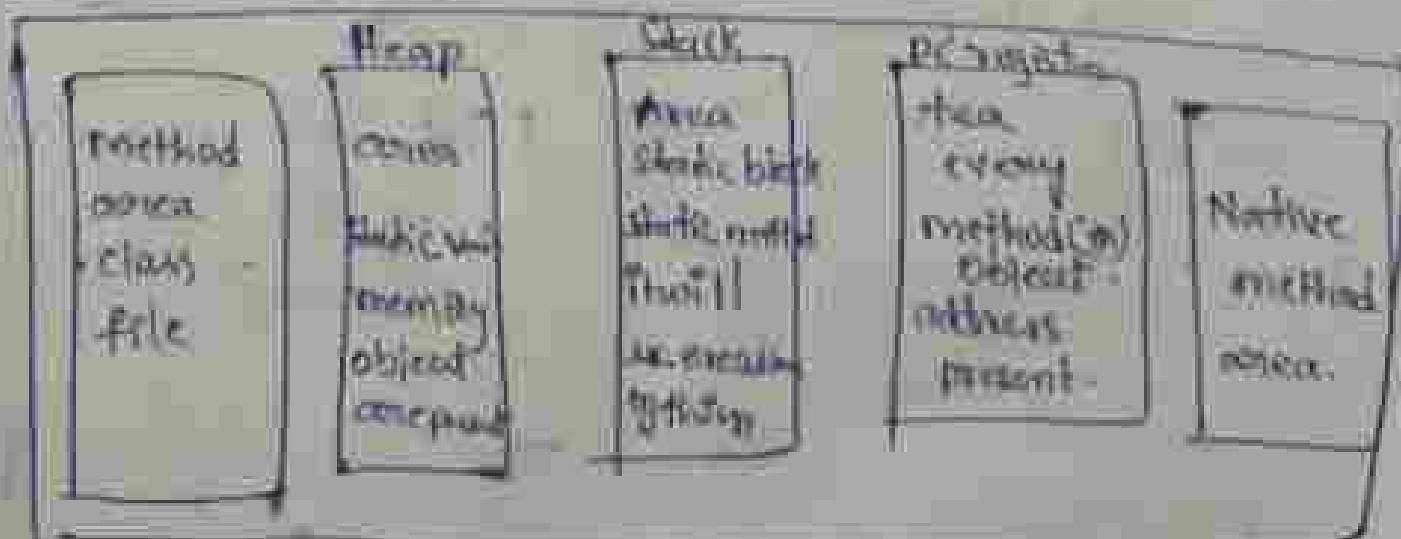
Preparation :- It will check one there any VIP(OR)
Static Variables one there if any static Variables
are there in heap memory static Variables allocated
default Values also given.

Initialization :-

Static Block is executed

like `a=10, b=20;`

Final Data Areas



After execution will start. JVM will look in interpreter to convert it. It's JIT will do only at specific time if some method is there which is been called multiple times. If suppose odd() is there two are calling so many times again and again, no need to convert in OS's JIT.

1st JIT will check odd() is converted before or not. If it is converted, it will just use. 2nd it will avoid duplicates. Interpreter will execute line by line.

Garbage Collection → if any object with no reference it will collect.

class Demo {

```
    static int a;           ← ①
    static int b;           ← ②
```

static {

```
    System.out.println("Static block");
```

```
    a=10, b=20;           ← ③
```

static void disp() {

```
        System.out.println(a); ④
```

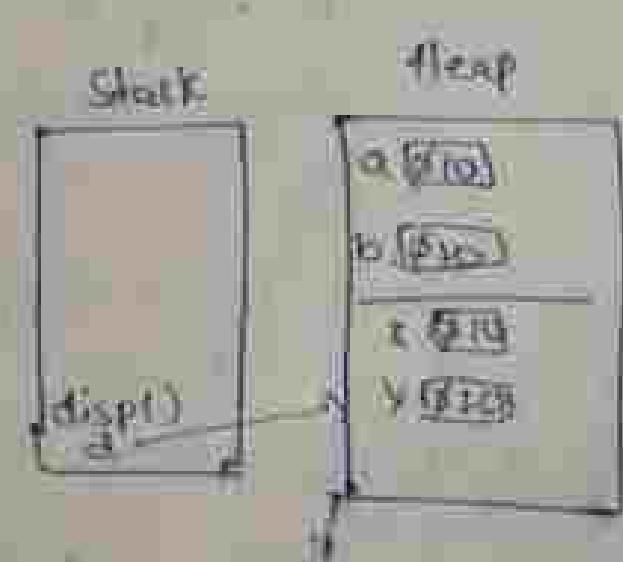
```
        System.out.println(b); ⑤
```

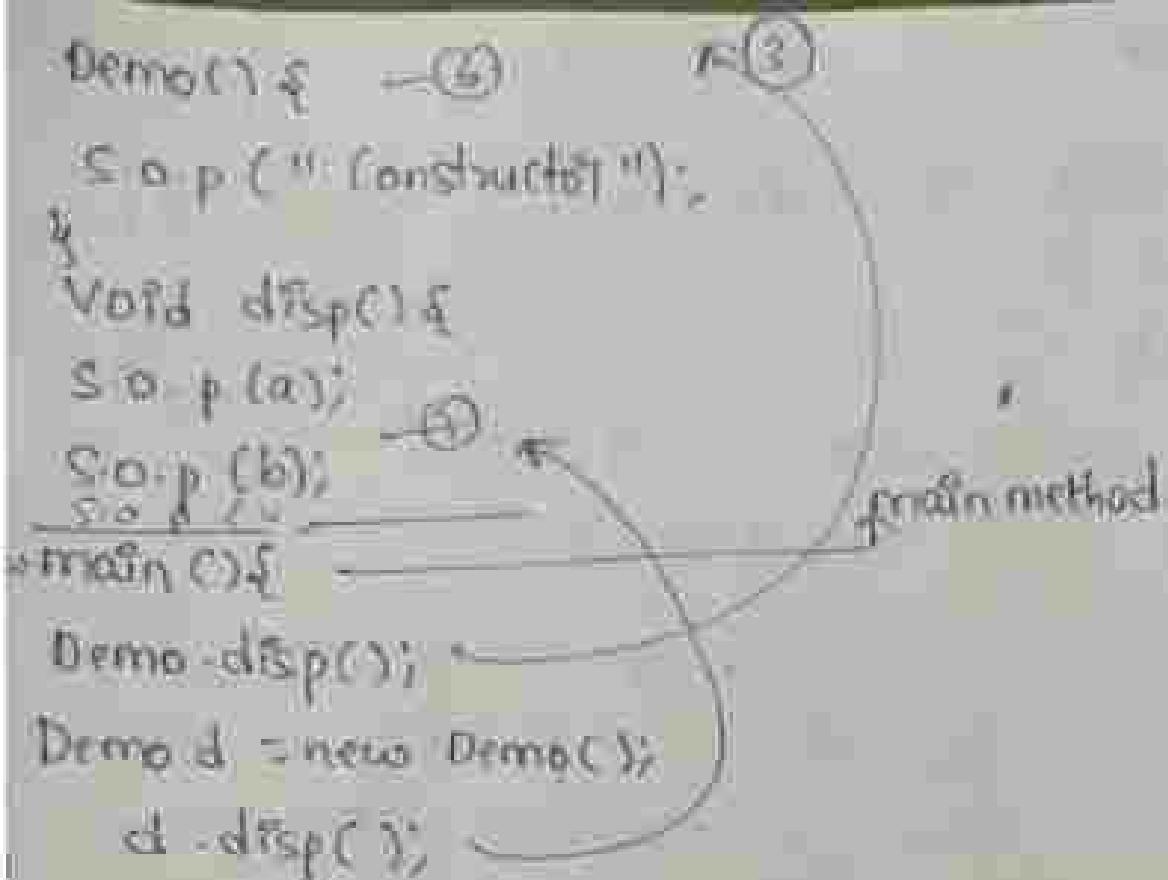
}

int x; int y; ← ⑥

x=10, y=20; ← ⑦

```
    System.out.println("Non Static block");
```





Demo d = new Demo(); while Creation object

3 things happen

1. Finalization of instance Variable inside object
2. Java block executed
3. Construction (include constructor)

Execution :-

- 1) Static Variables \Rightarrow Heap area \Rightarrow during Class loading
- 2) Static block \Rightarrow the initialization Static Variable \Rightarrow Class loading
- 3) Static method \Rightarrow main method
 \Downarrow
 Static method should call
 from main method
 without creating object

Why? ~~WAP~~ to Count objects

→ static Variable memory will be allocated once.

Class Demo

```
int a;  
int b;  
int count;  
Demo()  
{  
    count++;  
}  
main()  
{  
    Demo d1 = new Demo();  
    Demo d2 = new Demo();  
    Demo d3 = new Demo();  
}
```

Some copy will be for
all the objects

```
100 hundred obj. Count++ during  
by chance one passing wrong count  
so final java block  
& count++
```

Static Variables → static keyword used to
Create Variable.

→ memory allocated during class
loading.

→ memory allocated in heap area.

→ memory allocated only once in heap

→ One copy of static Variables
used by all objects.

→ Using class we can call static Variable.

→ It is also called as class Variable.

- Object independent
- accessed inside static & non static.
- Initialize static Variable (purpose)

Static Block vs main method

↳ First Static Block will execute

After main method will execute

Static method can be
using class name

by using object reference
can be called.

Object independent

Non static method

object creation is needed
by using object reference
can be called object Dependent

Diff b/w static Variable Vs instance Variable
class forms

private float pa;

private float td;

private float st;

private static float pi =

Static (class variable)

it is same for all forms

so in memory same memory

get

pi = 3.14;

void input()

Scanner scan = new Scanner (System.in);

System.out.println("Kindly enter principal amount");

pa = scan.nextInt();

So, plz kindly enter time duration).

td = Scan.nextInt();

Void Compute()

{

SI = SI * td/100;

}

Void display()

System.out.println(SI);

}

main CSE

Scanner fl = new Scanner();

Scanner fl2 = new Scanner();

fl. next();

fl. display();

fl. Compute();

fl2. next();

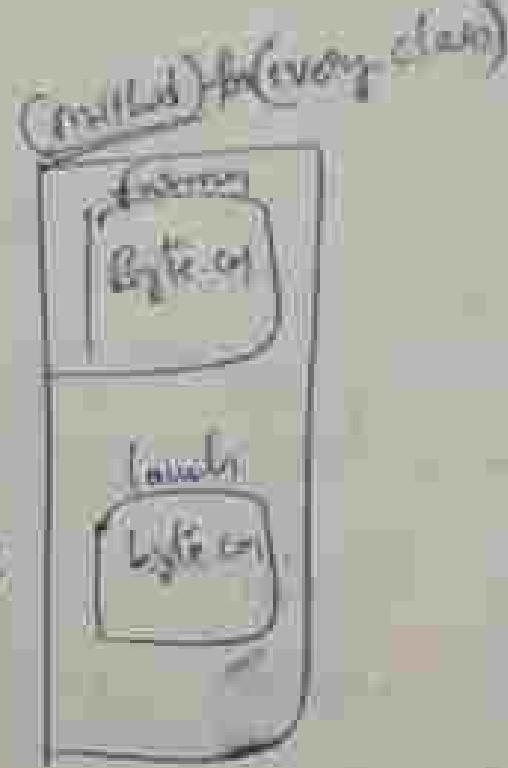
fl2. display();

fl2. Compute();

Whenever common copy of data has to shared among all objects of class, they are not specific to any object then that data is used to share in static.

Static method also called as (generic method)

non-static method also called as (specific method)



Inheritance + Code Reusability

Relationship :-

base class - sub class

- i) IS-A \rightarrow Inheritance \rightarrow parent - child relationship
- ii) Has-A \rightarrow Dependency injection (Aggregation / Composition)
one class acquiring properties and behaviour
to another class \rightarrow Inheritance
- iii) Un-related Class



we developed relation, by using extends keyword

```
Class Demo1 { // parent / Base / existing
    String name = "Vinay";
}
```

```
int age = 22;
```

```
Void disp() {
```

```
    S.o.p (" Demo1 " + age + name);
```

```
Class Demo2 extends Demo1 {  
    // child class / derived sub
```

```
main ( ) {
```

```
    Demo1 d = new Demo1();
```

```
    d.disp();
```

Important Rules : (Inheritance / In-¹ part)

- i) Single Inheritance is allowed (one class extends another class).

27

Class Demo

String name = "Vimay"

void dup()

Step (Home)

Class. Demo 2 extends Demo 1

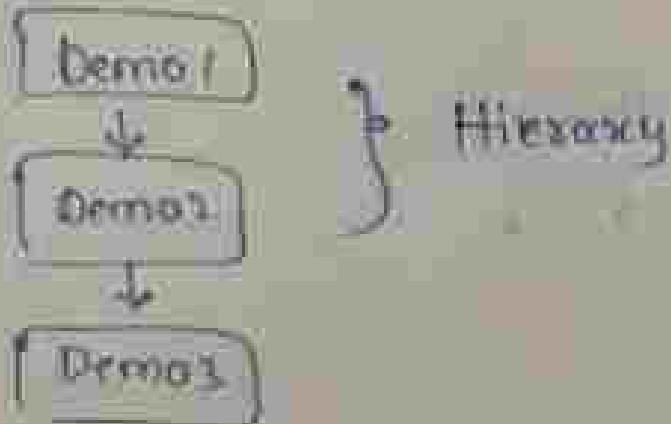
71

MathGuru

```
Demos = new Demos();
```

d·dispel

- ii) Object class is parent of all classes
 - iii) Multilevel Inheritance is allowed



Unit 2 Class, Demons & Methods, Object class

Serving name: "Vinay";

void **click**

S. o. p. (Singer)

Properties and methodology

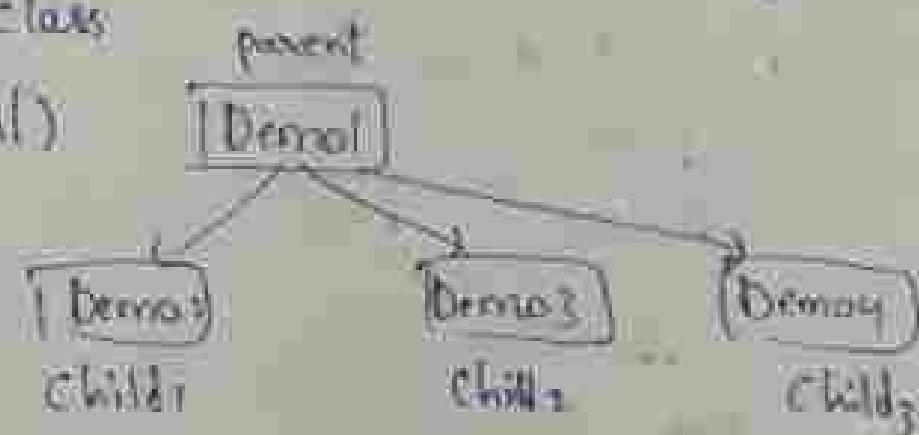
Class Demo2 extends Demo1

Class Demo3 extends Demo2

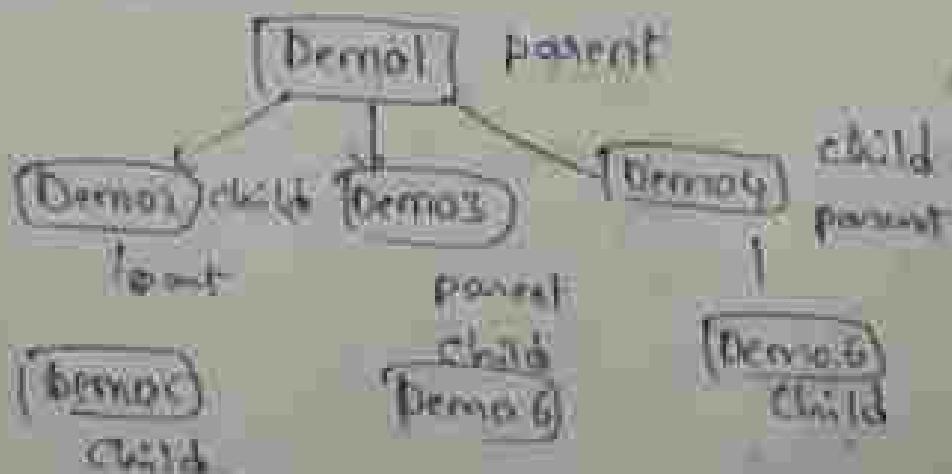
main() {

 Demo3 d1 = new Demo3();
 d1.display();

iv) One parent class can have any number of child classes
(Hierarchical)



v) Hybrid mixture of hierarchical & multilevel
is called hybrid



Class Demo1 {

String name = "Vinay";

Void disp() {

S.o.p (name);

}

Class Demo2 extends Demo1 {

}

Class Demo3 extends Demo1 {

vii) Multiple Inheritance is not allowed

(one child can't have two parents (father))

parent

Demol

Dem 2 parent

Dem 3

child

(parent) Object

child

Demol

(parent)

child

Demor

(parent)

Demol

(child)

To achieve multiple inheritance we can use interfaces.

vii) Cyclic inheritance is not allowed

Demol



Demol2



Sometimes Demol will say he is parent of ~~not~~ and Demol2 → will say Tom parent ~~of~~

Ans

Class parent extends Person

Class parent2 extends parent

Class parent3 extends parent2

viii) private members of a class doesn't participate in inheritance. Using (to preserve encapsulation) class parent is

```
private String name;
```

```
void disp()
```

```
{  
    System.out.println("parent");  
}
```

```
}
```

private members not allowed to avoid direct access

(encapsulation following)

Class child extends parent is

```
void disp()
```

```
{  
    System.out.println("child");  
}
```

name="Hyder" → error because private

Constructor will not participate in inheritance.
Because constructor will participate while object creation.

But parent class constructor will get called.

Because of super() present in child class

Class Demo1

String name;

Demo1()

System.out.println("I");

Void disp()

System.out.println("name");

Class Demo2 extends Demo1

Demo2()

Super(); \Rightarrow To call parent class

of constructor

(by default it is present)

constructor

main()

Demo2 d = new Demo2();

d.disp();

Constructor extends (Super), this() in inheritance.

class parent {

int a, b;

parent() {

a=10;

b=20;

System.out.println("parent Constructor");

parent(int a, int b) {

this.a=a;

this.b=b;

System.out.println("parent constructor");

class child extends parent {

int x, y;

child() {

Super(); // Call the parent constructor

x=100;

y=200;

System.out.println("constructor by default");

child(int x, int y) {

this.x=x;

this.y=y;

Void disp()

S.o.p (a);

S.o.p (b);

S.o.p (c);

S.o.p (d);

}

main()

Child d = new Child();
d.disp();

ii) Constructor with parameter in inheritance.

Class Demo1

int a,b;

Demol1();

a=10;

b=20;

S.o.p ("parent constructor");

?

Class Demo2 extends Demo1

int m,j;

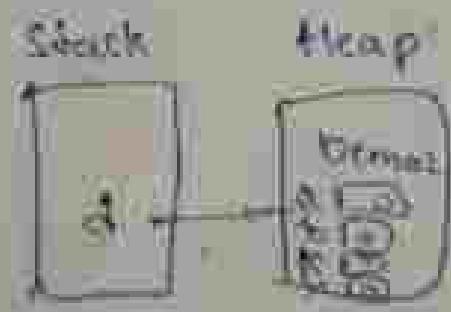
Demo2();

Super();

m=100;

j=200;

S.o.p ("child constructor");



void disp() {

 System.out.println("a");

 System.out.println("b");

 System.out.println("c");

 System.out.println("d");

 System.out.println("e");

 System.out.println("f");

 System.out.println("g");

Output:

parent constructor

child constructor

10

20

100

200

Parameters in Constructor example.

Class Demo1 {

 int a, b;

 Demo1 (int a, int b) {

 this.a = a;

 this.b = b;

 } // parent constructor

Class Demo2 extends Demo1 {

 int x, y;

 Demo2 (int x, int y) {

 this.a = x;

 super (x, y);

 this.b = y;

Comp("child construction")

Void Δ \rightarrow S

S \rightarrow P(1)

S \rightarrow P(2)

S \rightarrow P(3)

S \rightarrow P(4)

Result(?)

Object 2 Δ = New Object (100, 100)
(Δ = AspL)

What is?

Focus \Rightarrow Skills

Inheritance: process of one class (parent) acquiring properties and behaviours of another class (child). (hierarchy) (Extends keyword)
Is-a relationship \Rightarrow parent-child relationship



\Rightarrow Learn - personal
 \rightarrow Education
 \rightarrow Car
 \rightarrow Home

Common class
 has common
 in parent class
 (Code reusability)

Access-modifiers and Access Specifications

Access Specifying - by type

1) public

2) protected

3) default

4) private

→ apply

class

method

constructors

variable

void disp()



Project → multiple functionality private

→ by addition

→ subtraction

→ multiplication

→ division

when we are doing project we will divide into multiple project parts (or modules)

folders we call it has "package"

packages → shr dedicated after inheritance

	within a class	outside class within package	outside package (interrelation)
public	✓		
protected	✓	✓	✓
default	✓	✓	✗ within same package other
private	✓	✗	✗

public strongest

protected

visibility increasing

default

private weakest

→ All the methods and constructs **public** is highly recommended.



child is inherited
methods

what ever are inherited from parent
we can't change its modify (use them as it is)

in child as per requirement we can change.

Such a behaviour with gets inherited from parent
in child we are overriding them we called
it as overriding methods.

→ Specialized methods = `child's Smiling()`
parent not having. but child having.)

Symbolic

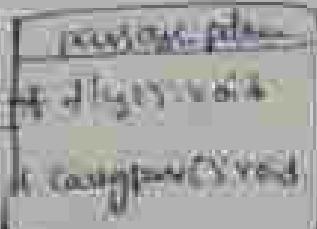
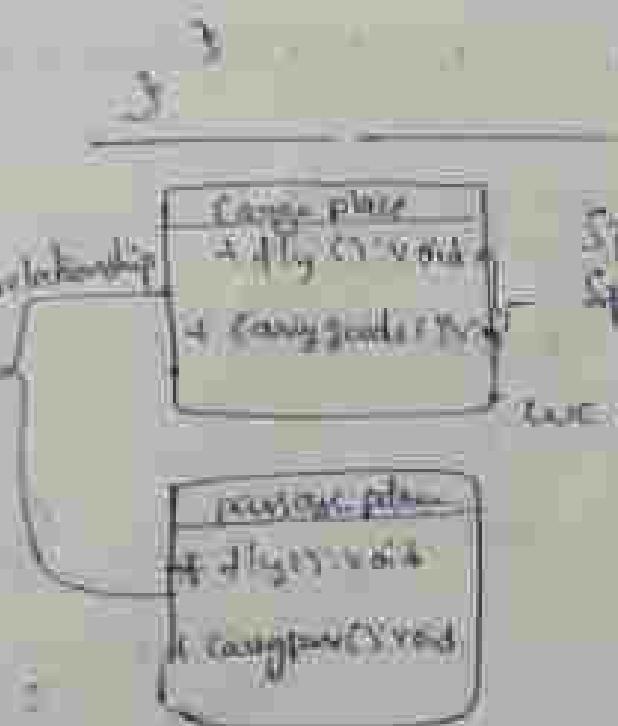
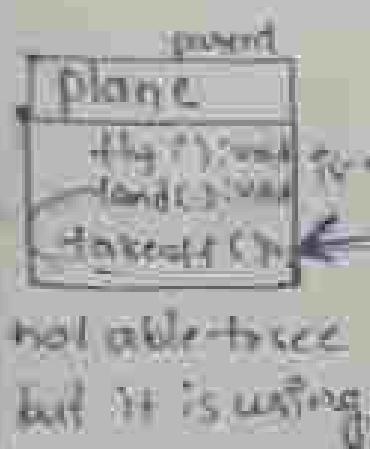
Consider modeling language

+ → public , # protected , ~ default , - private



class

Class plane of
private int cont;
private string color;
public void fly();
protected land();



class plane 5

public void takeoff()

{

System.out.println ("plane is taking off");

}

public void fly()

{

System.out.println ("plane is flying");

}

public void landing()

{

System.out.println ("plane is landing");

}

class Cargo plane extends plane

right now
the package class

public visibility = extending

Super ("Cargo plane" has nothing written)

class passenger plane extends plane

class good

→ who is parent of all classes

→ Object Class

→ Parent → Child → Inherited → overriding → Specialized

Rules to override method

① we cannot reduce visibility of overridden method

But we can increase

public work

child
visibility

In Java so many built-in class

we should invoke visibility of access specifiers

② Return type of overridden method must be same as that of overriding method.

Return type we cannot change

possibility of change of Return type

③ Return type of overridden method in child class can be different as that of parent. If it is co-variant return type (return type is a ~~parent~~ ~~sub-class~~ relationship), then it is parent.

④ Parameters of overridden method must be same as that of parent else it will be considered as Specified method. Consider method overloading.

Super() → Inside constructor
 → it will call parent class constructor

Super; → keyword

This; → Refer to current object runtime
this(); → calling constructor of same class

class Demo1

int age = 6;

class Demo2 extends Demo1

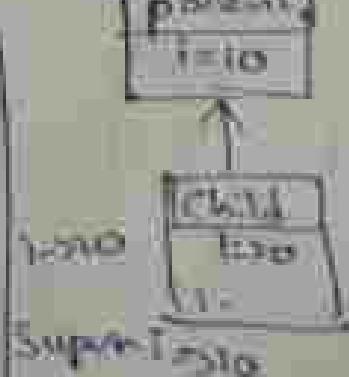
{ int age = 5; } 20 - word

void

l.o.p (super.age);

= Super.age; 20

int public class Demo1
 {
 }
 class Demo2 extends Demo1
 {
 public void l.o.p()
 }
 Specified method



Specified method
 super() is called
 super() is called

When ever we want to invoke instance Variable of parent class with super keyword we will use

Super;

→ Setting instance Variable of parent class

(Super Variable ;)

Super();

and calling parent class constructor

by default constructor having Super method

→ final Keyword

(class to create class)

final → class

(new object creation)

→ method

→ Variable

→ Abstract

final class → will not inherit, not nothing

final method → will get inherit but method will not overriding.

final Variable → acts as constant we can't change.

final double pi = 3.14;

Polymorphism, Abstraction

↓
abstract

↓

interfaces

functional interfaces

Lambda Expressions

to A - Inheritance
to A - Abstraction

Polymorphism → polymorphic family

(many forms)

Polymorphism = means = forms

(one = many)

Compile-time polymorphism - method overloading

→ no. of parameters

→ same datatype

datatype of parameters

(run-time polymorphism)

→ Run-time polymorphism

(code size
reduce)

→ overridden True (run)

↳ method overriding

Advantage code size reduces and flexibility

Parent

Child
Implementation

Child
Implementation

Class parent | ✓

Conjugated polyisobutylene

```
public void myCode
```

See Fig. "parent (crying)" by overriding method

1

class child friends parent

```
public void copy() — overridden method
```

Sop ("child cries at low voice");

10

class c bldg extends parent 14

Public Work (cont.)

so plan ("children at high value");

1

modo 63

Child Abuse and Child Neglect

child1 := new child2();

Geography

Digitized by

100

1000

Child's Voice at the Voice

Classification and Organization

whenever we create object reference type

same of that object type. ~~It will be now child~~

But in one case it can change references

Impairment type and

lose coupling, parallel processing, high efficiency

Parent class
Deficit:

(One copy) → multiple polymorphism

Shared copy → same line

sharing many lines

Deficit:

Same line

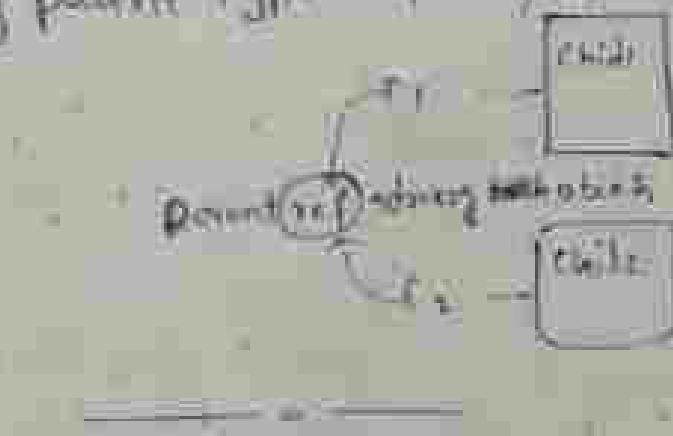
multiple lines

category →

Rule

→ to achieve polymorphism is when ever you create object that type of object must be parent type.

(Creating parent type reference)



at one time only
one address it will hold
new value. Current
value will go.

Program.

Class plane

public void takeOff()

 s.o.p("plane is taking off")

public void landing()

 s.o.p("plane is landing")

public void fly()

 s.o.p("plane is flying")

Class cargoPlane extends plane

public void takeOff()

 s.o.p("cargo is planning")

Class passengerPlane

public void takeOff()

 s.o.p("passenger is planning")

Class flight

public void plane(Plane plane)

plane.fly()

plane.land()

plane.Booking()

parent reference

to parent type

Abstraction

- Can be achieve by ① using abstract keyword (Implementation 100% may be not).
② But using Interface you can achieve 100% abstraction.
- ↳ ~~↳~~ hiding actual implementation ~~showcasing~~ only the feature
- ↳ ~~↳~~ even ~~we~~ don't know implementation but we getting feature
- Abstract method which does not have body, b.)
Implementation only Body Signature of these.
- In a class if one method also abstract method then that we should write class also abstract.
Ex: abstract class plane {
 abstract public void take off();
}
- abstract keyword can be used to method abstract method & such method implementation is not there only method signature.
- Abstract keyword can be used to class and method.
Abstract keyword can't be used to Variables.

Abstract class

```
abstract public void display();
```

↳

```
class phone extends abstract class {
```

↳

→ We can create obj of abstract class

↳ obj1 = new phone(); ✓

→ We can't create object for abstract class

↳ obj1 = new abstract(); X

→ Abstract class have both abstract & normal method

Abstract class have > all normal method

Abstract class have all abstract method

The subclass/ child class is extending abstract class
then either have to implement abstract method

(i) declare that class abstract

→ Constructor can't be made abstract

because already super(); method is there
inside so abstract rule implementation holding
but super method is there.

→ abstract as incomplete class

↳ we can't create object without constructor

↳ child class will inherit and create object

Can we make abstract class as final?
Ans. final class will not participate inheritance
test.

Abstract method, it will inherit but overridden

Variable \rightarrow abstract \rightarrow ⑥

\rightarrow abstract class constructor will call child super() parent class

Can we have constructor in abstract class - yes

How

abstract class Animal

if it not include constructor

class Demo2 Extends Animal

Defult (o) ^{parameter} constructor

super()

Although IS-A relationship skip
scanning and building vertex in class
through IS-A relationship
building communication and data navigation
class.

Has-A Relationship

→ IS-A Relationship is inheritance

parent-child relationship

we can achieve this by using extends
key word

→ ① Class Engine → Car HAS-A Engine

↳

Class Car

↳

② Class Address & Dependent Object
↳ Student HAS-A Address
↳ Class Student & Target Object
↳ Address

③ Class Account Employee HAS-A Address

↳

Class Employee

↳

Account account

Variable declaration

that Variable Should be
initialized.

→ Class Engine

Class Car

Has-A relationship

Engine Engine, New Engine, Minster Engine

Has-A Relationship - to establish another ^{class} object should be a part of this another class ref

→ The engine class object created and injected to each class

↳ class Engine { // Dependent Object

1

class Car { // Target Object

 Has-A Relationship

 Engine engine; // instance variable

Dependency Injection : The process of injecting dependent object into target object is called as

"Dependency Injection"

→ In how many instance Variables will initialize
two ways

a) Constructor dependency injection

b) Injecting dependent object into target object
through a constructor.

b. Setter dependency injection

→ Injecting dependent object into target object
through its setters.

Relationships in Java

As part of Java application development, we have two entities (class) as per the application requirements.

In Java application development, if we want to provide optimizations over memory utilization, code reusability, execution time, Shareability then we have to define relationships b/w entities.

- There are three types of relationship b/w entities
 - 1. HAS-A relationship (extensively used in projects)
 - 2. IS-A relationship
 - 3. USE-A relationship (not popular) ☺
- Q) what is the difference b/w HAS-A relationship and IS-A relationship

Ans: HAS-A relationship will define associations between entities (class) in Java applications, here associations b/w entities will improve communication b/w entities and data navigation between entities.

IS-A Relationship is able to define inheritance b/w entity classes, it will improve code reusability in Java applications.

Association - Relating two class in this style

We can achieve 4 types of Relationship

Associations In JAVA

here are four types of associations between entities

1. One - To - One Association (1:1) (target object
use private variables)
2. One - To - many Association (1:m)
3. Many - To - one Association (m:1)
4. Many - To - many Association (m:m)

To achieve associations between entities, we have to declare either single reference or array of reference Variable of one entity class in another entity class.

Ex- Class Address {

 - - -

 } Class Account {

 - - -

 } Class Employee {

 - - -

Address[] add; // It establish One - To - many
Account account; // One - to - one Association

}

between two class

communication between
class will build

Employee has account

Employee has ^{two} account
Current, Home

→ Reference Variable of
one class Provide one side
class - Association 1 : 1 situation

Many address associated with
Employee

→ that be in array
One to many

Code Reusability :-

class object {

 public boolean equals (Object obj) {

 }

 public String toString() {

 }

Some one wrote code
for anything already
we need to
type

class Student {

 }

System.out.println (new Student()); — Internally toString
method called

Relating two class

Ans. A relationship in creating reference Variable
in Target class to Two types :- 
1. m (1 to 1) 1 to 1
array variable

→ Java project = package.

This is primitive

Student class :-

Attributes of Student 
supper class variable

private String name;

Instance Variable

private Integer age;

two ways we can Set a value
constructor,
setter

private Double salary;

Integer id;

Generate constructor

Generate get and set method

Another class TestApp

possessing student object. ToString method called here.
So we should override ToString()

→ Anywhere two class used established association
But we used constructor only - even though
Ans - to initialise instances variable but those
variable even though they are Object type
not of class defined they are predefined
String, Integer, - predefined submit user code

output - Student [Name=Vijay, SId:10, Age=23],
type of self

Associations - ① one to one Association Mapping

e.g. Every employee have only one Account



First we have create dependent object and then
then we have inject to target object -
if Account is ready so we can give to
Employee

→ In injection - Account, Employee
In injection - on

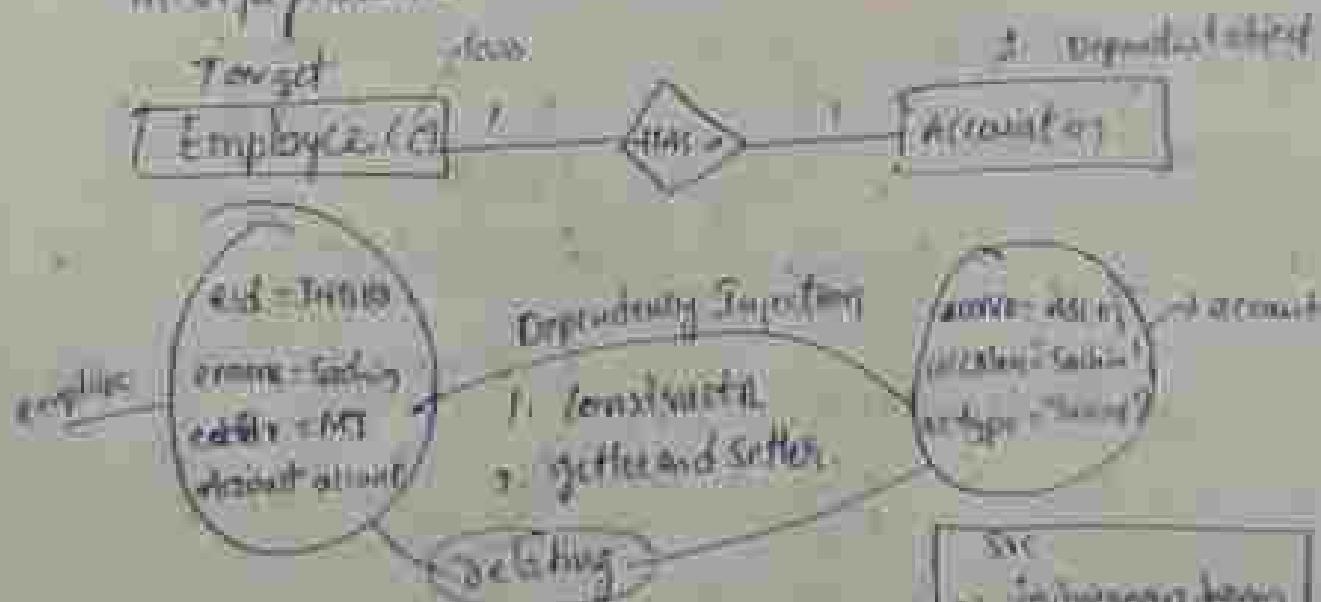
- Dependent Object code is ready to getting value by through constructor
- printing employee and account details
- If they are private then we can access and print remove private call (account, employee)

then in main method

First Create Account Object & inject value
 then Employee Object we are injecting account through constructor in employee.
 then call method to print details.

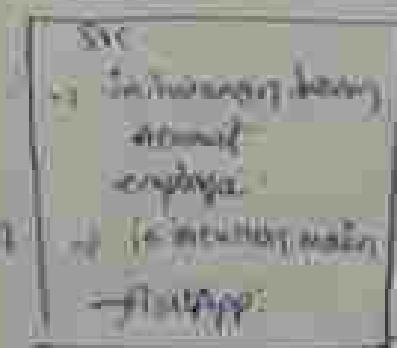
We have import class to main method

import bean.Account;



We are getting details of account for employee selected due to navigation

Communication and data navigation



1. One to One Association

It is a relation like entities whose one instance of an entity should be mapped with exactly one instance of another entity.

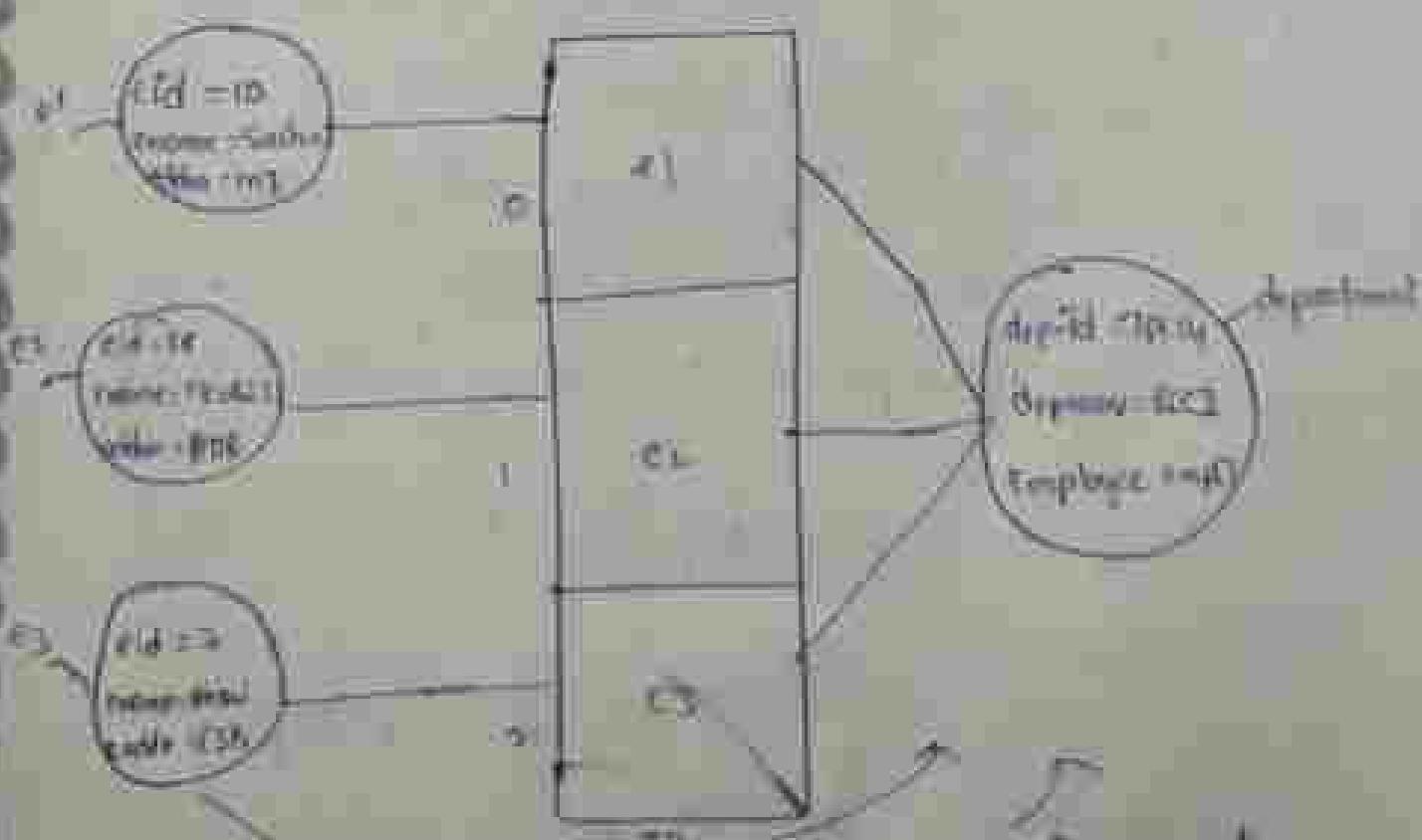
e.g. Every employee should have exactly one account.

② One to many Association

the single department has multiple employees.

One to many

many means (data to many)
Employee Employee



many employees creating and injecting into Department

First we have to create Employee class and Object

then, department class

We have to create employee [Emp] reference of employee through constructor.

→ Then department class we have to inject

reference of employee class in class

→ main method we have first employee object.

the array with objects.

→ to print employee data we have to use

for each loop because Array

Same process as one-to-one process

One-to-many association

→ One-to-many association between entity classes

It is a relationship between entity classes where one instance of an entity should be mapped with multiple instances of another entity

Example: Single department has multiple employees

Ex- One - many - association mapping

department has many employees

3 Many-to-one Association (One to many)

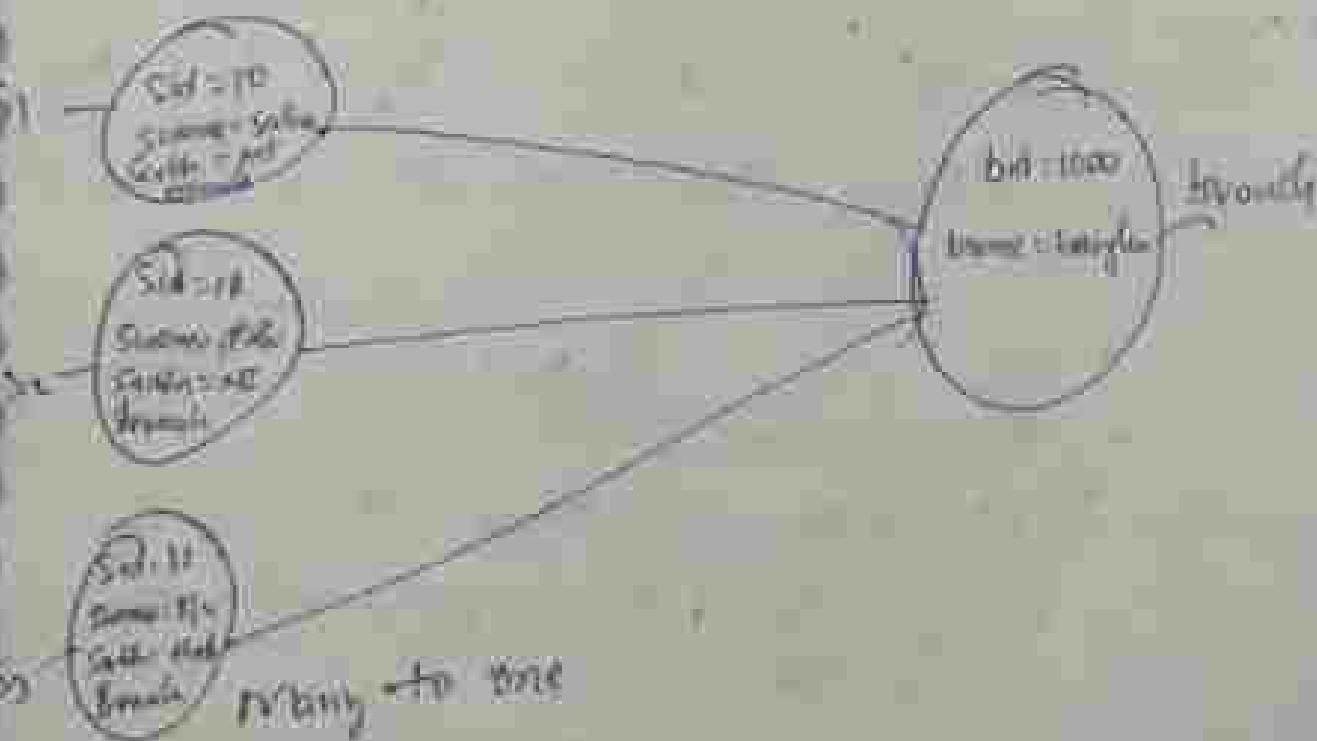
e.g. Multiple Student have joined with a single branch

dependent object is branch

Target object is Student

first we have create branch class
and then we reference in student class

So this inject is bit costly. - means more lines of code



many Students getting into branch.

process: Create branch class first

create student then reference we in Target class

In this method give value by letter and get by method chaining. \rightarrow for for for

Many - To - one Association

It is a relationship between entities, where multiple instances of an entity should be mapped with exactly one instance of another entity.

Ex: Multiple student have joined with a single branch.

eg: Refer :- Of - Many - One - Association mapping

4

Many - To - many Associations

Many Students have joined many courses.

First courses object and then inject to Student

Create course class. courseName, courseId, courseid

create many of courses to Course (Course (course))



Many - To many Associations

It is a relationship between entities, where multiple instances of an entity should be mapped with multiple instances of another entity.

Ex: multiple students have joined with multiple courses.

eg: Of - many - many - Association - mapping

Interface = its implementation to server side

Any SRS (Service Requirement Specification) is called as Interface

data base means where we can store data storage medium that keep data in table form record files

Ex- Sql, MySQL, Oracle, PostgreSQL

→ There are 3 data base to connect with them.



This is Service requirement Specification

to promote this Sun people gave us API

to connect JDBC (Java database connectivity)

API → Collection of class files



to run web applications we need special software (server will not JVM)

→ class JVM (part of server software) → Tomcat.

→ class web application → JVM (part of server software)

→ Sun people responsible to define JDBC API and database vendor will provide implementation for it.

Building Banking Application

Bank is giving some services to customer
Customer will not contact java code.

So, GUI will be -> picturel graphical user Interface
to connect java code and customer
GUI- Interface using this interaction is
happening b/w application and customer.

Def 2: From Client point of view an interface define
the set of services what is expecting from Service-
provider point of view an interface define the set of
services what is offering. So interface acts a
contract b/w client and service provider.

e.g.: GUI screen of ATM defines the set of services
what the customer is expecting.

Bank people defined the same set of services
what the customer is expecting.

Customer \Rightarrow GUI \Rightarrow Bank - Contract b/w customer
on \Rightarrow GUI \leftarrow offering on b/w bank,
expecting client

Def 3: Inside interface every method is always abstract
whether we are declaring or not hence interface
is considered as 100% pure abstract class.

Interface - Account

- // It is 100% abstract class
 - // the methods are abstract and public by default
- Void withdraw()
- Void deposit()
- Void checkBalance()

Summary : Interface corresponds to Service Requirement Specifications (SRS) or Contract between client and service provider. 100% pure abstract class.

- use ISample as Interface with public methods
- ISample - to give class name take Sample and add Impl \Rightarrow SampleImpl + class name

Declaration and Implementation of Interface

i) whenever we are implementing an interface compulsorily for every method of that interface we should provide implementation otherwise we have to declare class as abstract in that case child class is responsible to provide implementation of remaining methods.

- ii) whenever we are implementing an interface method compulsorily it should be declared as public either it would result in compile-time error (decreasing visibility).

Introduce Tsample

Void mid(); // 100% abstract class.

Void m2(); // methods by default abstract and public

Class SampleImpl implements Tsample

public Void m1() {

System.out.println("m1()");

}

public Void m2() {

System.out.println("m2()");

}

}

main() {

Tsample sample = new SampleImpl();

sample.m1();

sample.m2();

Realtime Coding hierarchy (Developers approach):

Interface (100% abstraction)

↳ implements

Abstract Class (not 100% abstraction)

↳ extends

Concrete class (no abstraction).

Intermediate Example

```
Void m1();  
Void m2();
```

Abstract class `ISampleImpl` implements `ISample`

```
public Void m1(){  
    S.o.p("m1()");  
}
```

Void m2(); → By default abstract

Class `SubISampleImpl` extends `ISampleImpl` &

```
public Void m2(){  
    S.o.p("m2()");  
}
```

main() {

```
    ISample Sample = new SubISampleImpl();  
    Sample.m1();  
    Sample.m2();
```

Inner classes

- The classes that are declared inside another class are called as inner class or nested class
- ① Inner classes have a special type of relationship that it can access all the members of outer class including private
- ② More readable and maintainable
- ③ Inner class requires less code
- ④ If a class is useful to only one other class then it is logical to embed it in that class and keep the two together.
- Four types
- Regular Inner class - class inside class
- Method local Inner class - method inside inner class
- Anonymous Inner class - object having creating class
- Static Inner class - class inside static inner class

① Regular Inner Class (for inner class class file is auto generated)

→ Class outer {

 class Inner {

 void InnerMethod() {

 System.out.println("Inner method");

 }

 void outerMethod() {

 Inner inner = new Inner();

Inner & Outer Class → To call inner class. @type
↳ innermethod();

main() {

Outer o = outer();

o.innermethod();

System.out.println("Outer Inner is new outer()");

↳ innermethod();

InnerClass inner = new InnerClass();
 inner.innermethod();

↳ innermethod();

method Local Inner Class:

class Outer {

void outermethod() {

class Inner {

void innermethod() {

System.out.println("inner");

}

System.out.println("outer");

Inner i = new Inner();

i.innermethod();

}

main() {

Outer o = new Outer();

o.outermethod();

2. Anonymous Inner Class.

class Parent {

 void msg() {

 System.out.println("hi");

}

class Present {

 Parent parent;

 Present() {

 parent = new Parent();

 parent.msg();

 }

 void msg() {

 System.out.println("Hello");

3. Static Inner Class.

class Outer {

 static class Inner {

 void inner() {

 System.out.println("inner");

}

 void outer() {

 System.out.println("outer");

}

 }

 Inner inner() {

 return new Inner();

Difference b/w extends vs implements.

① → extends: one class can extends only class at a time

ex: class One & class Two

class Two extends One &.

Interface → Implements: one class implements any no. of interface at a time

interface one {

void m1();

interface Two {

void m2();

class CommonImpl implements one, Two {

public void m1() {

public void m2() {

② Actions: Can a single class extends and implement simultaneously if it is first we have implementation extends and followed by implementations.

ex: interface One {

public void methodOne();

} class Two {

public void methodTwo();

Class Three extends Two, implements one {

@Override

public void methodOne()

@Override

public void methodTwo()

(from sql)

1. statement

2. prepared Statement

3. call able Statement

4. connection

5. resultset

Implementation

from (your sql)

for the body abstract

Case 3 An interface can extend any no. of interfaces at a time.

e.g. interface One {

 public void methodOne();

}

interface Two {

 public void methodTwo();

}

interface Three extends One, Two {

 public void methodOne();

 public void methodTwo();

 public void methodThree();

}

Abstract

public To make the method available for every implementation class

abstract Implementation class is responsible for providing the

Implementation

Q. Which API (Java 6.0) (*)

Implementation should be given by

- a. MySQL
- b. Oracle
- c. PostgreSQL

How many access modifiers in Java?

- a. public, b. private, c. protected, d. static, e. synchronized
- f. final, g. final, h. abstract, i. native, j. transient, k. volatile

(j and k only for variables)

default is by default

Since the methods present inside the interface is
→ public, abstract we can't use the modifiers like
static, private, protected, final, synchronized, native, final.

static → associated with implementation

abstract → can't have implementation

Interface Variables

→ Inside the interface we can define Variables

→ Inside the Interface Variables define requirement
level constants.

Every Variable present inside the interface is by
default public static final.

interface Sample f

int x=10;

notebook completed voice note
book (2) to be continued

public :: To make it available to implementation

class object

Static :: To access it without using implementation class name

final :: Implementation class can access the value without any modification

why methods are available without static but variables can't be static

a. complete information

method inside Interface

abstract (Incomplete)

Static :: complete information should be provided
class Test

static void main()

Variables inside Interface are public static final other access modifiers are illegal combination

a. private b. protected

c. transient d. volatile (Illegal)

Eg: interface Remote

int x; EF

Since the variable defined in interface is public static final, we cannot use modifiers like private

Since the variable is static and final, compulsory it should be initialized at the time of declaration otherwise it would result in compile time error.

Kerrybody

Java keywords are prefixed with `jav` so they
Can be used as Variable or object name or class name
Start with Small letter

Received with thanks by John Cottrell

- byte
 - short
 - int
 - long
 - float
 - double
 - char
 - boolean

- 1) if
 - 2) else
 - 3) Switch
 - 4) Case
 - 5) default
 - 6) for
 - 7) do
 - 8) while
 - 9) break
 - 10) continue
 - 11) return

- 1) frug
 2) calib
 3) hewly
 4) fliss
 5) flay
 6) curf (sp)
 Version 1)

卷之三

- public
private
protected
static
final
abstract
final

100

- 1) Class
 - 2) package
 - 3) import
 - 4) extends
 - 5) implements
 - 6) interface

60

- 1) news
 - 2) Pictures of
 - 3) Super
 - 4) (15)

Periodic Gr.

145

卷之三

307
W. 11th St. - U.S.A.

Land (187)

دیگر
کار
کار

ASCII Value

Special Characters

32 Space \rightarrow 64 @

65 A \rightarrow 90 Z Capital Alphabets

97 a \rightarrow 122 z Small letter Alphabets

Temp = ASCII Value

6 97 - 65 = 32 - difference we have

②

97 - 65

97 - 65

= 32 - ①

65 + 32

65 + 32

char(A + 32)

(A + 32)

65 + 32

(97) \Rightarrow a

→ Code [*both local and static final variable
in interface always local vars*]

Interface Example :-

Int a=10; // public static final by default

public class TestApp implements IExample {

public static void main (String [] args) {

Int a=20; // local Variable

System.out.println(a); // output is 20

}

(method area is shared)

Later on Jdk 8 version

heap area)

Servlet Code :-

→ Can a java class implements 2 interfaces simultaneously?

Yes - possible, but in both the interfaces method containing

Signature should be same, but different return types

(Compiler should only take method signature)

→ Interface Variables can be accessed from implementation class, but cannot modify it directly
modify

It would result in compile-time error

Eg:- Interface Remote {

Int VOLUME =100;

}

```
class Log implements Remote {
    public static void main (String args) {
        int volume = 10; // local variable
        System.out.println ("Value of volume is: " + volume);
    }
}
```

Interface Naming Conflicts

If 2 interfaces contain a method with same signature and same return type, then one method implementation is enough.

```
interface left {
    public void methodOne ();
}
```

```
interface right {
    public void methodOne ();
}
```

```
class Test implements left, right {
    @Override
```

```
    public void methodOne () {
    }
}
```

Case 2

If 2 interfaces contain a method with same name but different arguments in the implementation for both methods and these methods acts as overloaded methods.

Ex: Interface Left {

```
    public void methodOne();
```

```
}
```

Interface Right {

```
    public void methodOne();
```

```
}
```

Class Test implements Left, Right {

```
    @Override
```

```
    public void methodOne() {
```

```
    }
```

```
    @Override
```

```
    public void methodOne(int i) {
```

```
    }
```

```
}
```

Case 3:

If two interfaces contains a method with same signature but different return types then it is not possible to implement both interface simultaneously.

Ex: Interface Left {

```
    public void methodOne();
```

```
}
```

Interface Right {

```
    public int methodOne();
```

Class Test implements Left, Right {

```
    @Override
```

```
    public void methodOne() {
```

```
}
```

@Override

public int methodOne()

Note :-

- Q) Can a java class implements 2 interfaces simultaneously?
Ans : possible, except if two interfaces contains a method with same signature but different return types.

Variable naming conflicts :-

Two Variables can contain a Variable with same name and there may be a chance of Variable naming Variable naming.

Conflicts but we can resolve Variable naming conflicts by using Interface names.

Example :-

Interface Left {

int x=123;

}

Interface Right {

int y=999;

public class Test implements Left, Right {

public static void main(String[] args) {

System.out.println(x);

System.out.println(y);

Note = Inside interface the methods are by default "public" and "abstract".

Inside interface the variables are by default "public static" and "final".

We can also write an interface without any variables or abstract methods.

Interface Example :-

```
int a=10; public static final
```

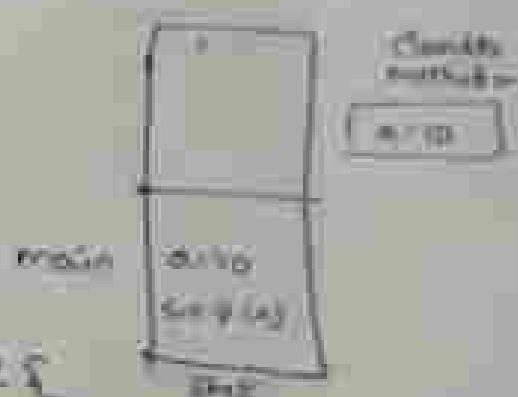
```
public class TestApp implements IExample {
```

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

```
        int a = 20; // local Variable
```

```
        System.out.println (a);
```

```
}
```



Marker Interfaces

```
interface Serializable {
```

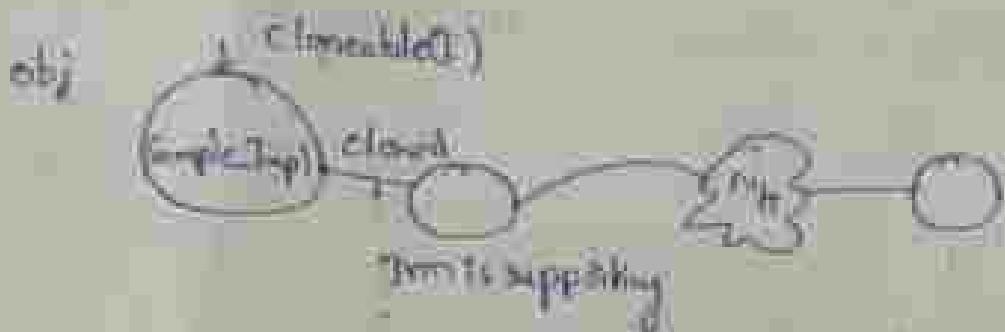
```
class SampleImpl implements Serializable {
```

```
    Serializable()
```



Interface Clonable

class SampleImpl implements Clonable {



Marker Interface

→ If an interface does not contain any methods and methods and by implementing that interface your object will get some ability such type of interface are called "Marker Interface" / "Flag Interface" / "Ability Interface".

→ Example

Serializable, Clonable, SingleThreadModel

Example 1

By implementing Serializable interface we can send that object across the network and we can save state of an object into the file

Example 2 By implementing SingleThreadModel interface server can process only one client request at a time so that we can get thread safety.

Example 3 By implementing `Cloneable` Interface our object is in a (position - present only one client) position to provide exactly duplicate cloned object.

Without having any methods in marker interface, our objects will get ability.

Ans - JVM is responsible to provide required ability why JVM is providing the required ability through `Interfaces`?

Ans to reduce the complexity of the programming.

Can we create our own marker interface?

Yes, it is possible but we need to customize JVM (hand ~~for begin~~).

Adaptor class (It is a design pattern allowed to solve the problem of direct implementation of interface methods).

It is a simple Java class that implements an interface only with empty implementation for every method.

If we implement an interface compulsorily we should give the body till all the methods, when this is not required is not this approach increases the length of the code and reduces readability.

Eg - interface X
 void m1();
 void m2();
 void m3();
 void m4();
 void m5();

```
class Test implements X{  
    public void m1(){  
        System.out.println("I am from m1()");  
    }  
    public void m2(){  
    }  
    public void m3(){  
    }  
    public void m4(){  
    }  
    public void m5(){  
    }  
}
```

In the above approach, even though we want only `m1()`, still we need to give body for all the abstract methods which increase the length of the code. To overcome this we need to use "Adapter class".

Instead of implementing the interface directly we opt for "Adapter class".

Adapter class are such classes which implements the interface and gives dummy implementation for all the abstract methods of interface.

So if we create adapter classes then we can easily give body only for those methods which are interested in giving the body.

Interface x

Void m1();

Void m2();

Void m3();

Void m4();

Void m5();

}

Abstract Class Adapter x implements x

public Void m1();

public Void m2();

public Void m3();

public Void m4();

public Void m5();

Class TestApp extends Adapter x

public void m3();

System.out.println("I am Adapter now");

Servlet (J)

implements

GeneralServlet (abstract class)

↳ extends

HttpServlet (abstract class)

↳ extends

newServlet Class

Topics

Interface & Wrapper class

interface completion

Wrapper class

imp topic and interview question

when to go interface, abstract class and concrete class?

Interface: It is preferred when we speak only about Specification (no implementation).

Abstract class: It is preferred when we speak about partial implementation.

Concrete class: It is preferred when we speak about complete implementation and ready to provide service then we go for Concrete class.

Difference b/w Interface and Abstract class

Interface: If we don't know anything about implementation just we have requirement specification then we should go for interface.

Abstract class: If we are talking about implementation but not completely then we should go for abstract class.

Interface: Every method present inside the interface public and abstract whether we are defining or not.

Abstract class : Every method present inside abstract class need not be public and abstract.

Interface : we can't declare interface methods with the modifiers like private, protected final, static, synchronized, native, strictfp

Abstract : There are no restrictions on abstract class method modifiers.

Interface : Every interface variable is always public static final whether we are declaring or not.

Abstract : No need to declare anything.

Interface : Every interface variable is always public static final we can't declare with modifiers like protected, private.

Abstract : No restriction in access modifiers.

Interface : for every interface variable compulsorily we should perform initialization at time of declaration otherwise we get compile time error.

Abstract : Not required to perform initialization of abstract class variables at time of declaration.

Interface : Inside interface we can't write static and instance blocks.

Abstract : Inside abstract class we can write static and instance blocks.

Interface :- we can't write Constructors.

Abstract :- Inside abstract we can write Constructors.

Note :-

Static block :- Class file loading happens and used to initialize static Variable.

Instance block :- During creation of an object just before the constructor call used for initialization instance Variable.

Constructor :- During creation of an object used for initialization instance Variable.

During child class Object creation, only child class object will be created but no parent class object will be created to (still constructs of child object will be created to bring the properties of parent parent is called to bring the properties of parent added to child).

To get address of object we have to use

`System.out.println(this.hashCode());`

Method to print

hashCode of object.

Q Can abstract class be object be created ?

A) No

B) Can abstract class contains Constructors ?

C) Yes.

What is need of abstract class?
Can we create an object of abstract class, Does it
contain constructor?

- Abstract class object cannot be created because it is abstract
- But constructor is used for construction of initial
the object

class Parent {

```
parent() {  
    System.out.println("parent constructor");  
    System.out.println(this.hashCode());  
}
```

class Child extends parent

```
child() {  
    System.out.println("child constructor");  
    System.out.println(this.hashCode());  
}
```

class main {

```
child c = child();  
c.copy(c.hashCode());
```

It will print same hashCode for all

why abstract class can contain constructor whereas interface does not contain constructor?

Abstract Class \Rightarrow Constructor It is used to perform Initialization of object.

* It is used to provide value of instance

Variable

* It is used to contain instance Variable which are required for child.

* Object to perform Initialization of those instance

Variables

Interface \Rightarrow every Variable is always static, public and final their is no change of creating instance Variable inside the class.

So, we should perform initialization at time of declaration.

So, constructor is not required for interface.

eg: Class Person {

String name;

int age;

new person (String name, int age){

this.name = name;

this.age = age;

}

Class Student extends person &

int roll no;

Student (String name, int age, int roll no) {

Super (name, age, roll no);

This roll no = roll no;

Can reference be created in abstract class

person p = new Student ("Vivek", 21, 61);

Can reference be created for interface?

Some Sample

Note : Every method present inside the interface is abstract, but in abstract class also we take only abstract methods.

What is the need of interface?

abstract class Sample {

 public abstract void m1();
 " " " m2();

interface Sample {

 void m1();

 void m2();

We can replace interface concept with abstract class, but it is not good practice

Ex:-

Interface X :-

(i)

(ii)

Class Test implements X :-

(i)

Test t = new Test();

(ii) performance is high

(iii) while implementing X we can extend one more class, through we can bring reusability

Ex:- abstract X :-

(i)

Class Test extends X :-

(ii)

(iii)

Test t = new Test();

performance is low

while extending X we can't extend any other class so we can't reuse

Note:- If extending is abstract we can't go for interface

Wrapper class :-

Purpose

1. To wrap primitives into object form so that we can handle primitives also just like objects.
2. To define several utility functions which are required for the primitives.

Constructors: Almost all the wrapper class have 2

Constructors

a. One taking primitive type

b. one taking String type

eg: Integer i = new Integer(10);
Integer i = new Integer("10");

Double d = new Double(10.5);

Double d = new Double("10.5");

Note: If String argument is not properly defined then it would result in runtime exception.

Called "NumberFormatException".

eg: Integer i = new Integer("ten"); here Number

format exception.

Wrapper class and its associated Constructors

Byte => byte and String

Short => short and String

Integer => int and String

Long => long and String

** float \Rightarrow float, string and double

double \Rightarrow double and string

** character \Rightarrow character

** boolean, boolean and string

eg:- float f = new float (10.54);

float f = new float ("10.54");

float f = new float (10.5);

float f = new float ("10.5");

float f = new float (10.5);

** character c = new character ('b');

character c = new character ("a"); // invalid

boolean b = new boolean (true);

boolean b = new boolean (false);

// boolean (true); //EE

// boolean (false); //EE

// boolean (true); //EE

// boolean (false); //EE

eg:-

class Test {

main() {

boolean b1 = new Boolean ("yes"); //EE

boolean b2 = new Boolean ("no"); //EE

s.o.p (b1);

s.o.p (b2);

~~S. o. p (b, equals(b)) ; false. - equals (false) - true~~

~~S. o p (b₁ == b₂) ; false~~

In case of Boolean Constructor boolean value be treated as true won't be case sensitive. part of "true" & all other it would be treated as false.

Note:- If we are passing string arguments then case is not important and content is not important.

If the content is case insensitive string of true then it is treated as true. In all other case it is treated as false.

Note:- In case of wrapper class, toString()

is overridden to print the data. In case of wrapper class, equals() is overridden.

To check the content.

Just like string class; wrapper class are also treated as "immutable".

Immutable class

If we create an object and if we try to modify or change with that change new object will be created and those changes will not reflected in old copy.

(Can we make our Classified class immutable?)
Ans) Yes possible as shown below

```
class Classified {
    int i;
    Test (int i) {
        this.i = i;
    }
    public Test modify (int i) {
        if (this.i == i)
            return this;
        else
            return new Test(i);
    }
}
public static void main (String [3] args) {
    Test t1 = new Test (10);
    Test t2 = t1.modify (10);
    Test t3 = t1.modify (100);
    Test t4 = t3.modify (100);
    System.out.println (t1 == t2); // true
    System.out.println (t1 == t3); // false
    System.out.println (t3 == t2); // false
    System.out.println (t1 == t4); // true
}
```



Wrapper class Conclusion

1.5. *u* Introduced

- To store primitive data also in object so that we can make use of utility methods, we need wrapper classes.
 - Number \rightarrow Byte, Short, Integer, Long, Float, Double
 (parent) 2 2 2 2 2
 - Object \rightarrow Boolean, Character
 (parent) 2 1 (Character)

String \Rightarrow Case and content is not imp
boolean

Office 14

`toString()` ⇒ return the hashcode of the object.

equals() \Rightarrow Compares the reference

(Wrapper classes) and (String class) are **Immutable**)

- testing (⇒ point the content of the object)

equation) \rightarrow Compares the data

Can we create our own timetable class?

1000

→ friend' method even he is showing methods are available in almost every wrapper class (methods overloading)
public static - helper methods (utility methods)

→ Wrapper class utility methods

1. `ValueOf()` method (cont goes java.lang.Integer)
2. `XXXValue()` method
3. `parseXXX()` method
4. `toString()` method

→ `public static wrapper ValueOf(String int)` throws `java.lang.NumberFormatException`;

→ `public static wrapper ValueOf(String data)` throws `java.lang.NumberFormatException`;

`public static wrapper ValueOf(Integer data);`

ValueOf() method

To create a wrapper object of from primitive type of `String` we use `ValueOf()` method

It is alternative to constructor of wrapper class, not suggested to use

Every wrapper class except character class contain static `ValueOf()` to create a wrapper object.

Ex:- `Integer i = Integer.ValueOf("10");`

`Double d = Double.ValueOf("10.5");`

`Boolean b = Boolean.ValueOf("true");`

`s = p(1); // 10`

`s = p(1); // 10.5`

`s = p(b); // Value`

Q42

public static ValueOf(String s, int radix)

\Rightarrow binary: 2(0,1)

\Rightarrow octal: 8(0-7)

\Rightarrow decimal: 10(0-9)

\Rightarrow hexadecimal: 16(0-9, a-f, A-F)

\Rightarrow base: 36(0-9, a-z)

Integer n1 = Integer.ValueOf("100");

System.out.println(n1); // 100

Integer n2 = Integer.ValueOf("100L");

System.out.println(n2); // 100

Integer n3 = Integer.ValueOf("ten");

\Rightarrow NumbervFormatException

Integer n4 = Integer.ValueOf("100,34");

System.out.println(n4); // NumbervFormatException

Q43. ~~Praktik~~

public static ValueOf(primitiveType)

Integer n1 = Integer.ValueOf(10);

primitive \Rightarrow string

Double d1 = Double.ValueOf(10.5);

\Rightarrow NumbervObject

Character c1 = Character.ValueOf('a');

Wrapper Object

Boolean b1 = Boolean.ValueOf(true);

ValueOf() - use this to convert

String
primitive

Wrapper object

public static Character ValueOf (char);

valueOf() - use this to convert

Wrapper
object

Primitive
type

Integer -> byte, short, int

long, float, double -> ((> value()))

Not applicable for

Character and boolean

$$\text{result} = \min \text{range} + (\text{total} - \text{minrange} - 1) \\ = 128 + (180 - 128 - 1) \\ = 128 + 2 = 130$$

Q2

doubleValue()

we can use doubleValue() to get primitive type
for the given wrapper object.

These methods are a part of every number type
Object.

(Byte, short, Integer, Long, float, Double) all these classes
have these 6 methods, which is written as shown
below.

Methods

```
public byte byteValue();
public short shortValue();
public int intValue();
public long longValue();
public float floatValue();
public double doubleValue();
```

eg:-

```
Integer i = new Integer(120);
```

```
// result = minrange + (total - minrange -1)
System.out.println(i.byteValue()); // -126
System.out.println(i.shortValue()); // 130
System.out.println(i.intValue()); // 130
System.out.println(i.longValue()); // 130
System.out.println(i.floatValue()); // 130.0
System.out.println(i.doubleValue()); // 130.0
```

3. charValue()

Character class Contains charValue() to get char primitive for given Character object

```
public char charValue();
```

```
Character C = new Character('c');
char ch = C.charValue();
System.out.println(ch);
```

4. boolean Value()

Boolean Class Contains booleanValue() to get boolean primitive for the given boolean object

```
public boolean Value()
```

eg:-

```
Boolean b = new Boolean("false");
boolean b1 = b.booleanValue();
System.out.println(b1); // false
```

String
and
Character
combination
available

In total xxxValue() case 36 in number

\Rightarrow xxxValue \Rightarrow Convert the wrapper Object \Rightarrow primitive

5. parseXXX()

parseXXX() \rightarrow use \rightarrow content



parseXXX()

use use parseXXX() to convert String object into primitive type

family

```
public static primitive parseX(String)
```

Every wrapper class, except Character class has parseXXX()
 \rightarrow Convert String into primitive type.

```
int i = Integer.parseInt("10")
```

```
double d = Double.parseDouble("10.5")
```

```
Boolean b = Boolean.parseBoolean("true")
```

~~usage~~ ~~usage~~ Command line arguments and wrapper class
In real time Coding

// Wrap - to take inputs from the command line and
perform arithmetic operations

class Test

```
public static void main (String [] args)
```

// valueOf() -> Converts String / primitive -> wrapper type

// .xxxValue() -> Converts wrapper type to primitive type

// parseXXX() -> Converts String to primitive type

// command line arguments -> String Inputs = args [0], args [1]

```
int i = Integer.parseInt(args[0])
```

```
int j = Integer.parseInt(args[1])
```

S. o p (i+j)

S. o p (i-j)

S. o p (i*i)

S. o p (i/i)

// args -> String, convert into primitive type and parse

Y Y

form 2

public static primitive parseXXX (String s, int radix)
{
 long l = Long.parseLong(s);
 return l;

Every Integral type wrapper class (Byte, short, Integer, long)
contains the following parseXXX() to convert specified
radix String to primitive type.

eg. - int i = Integer.parseInt ("111", 2);

System.out.println(i); // 5

Note :- ~~String~~ \Rightarrow parseIntXXX

String \Rightarrow parseXXX() \Rightarrow primitive type

④

toString()



toString()

To Convert the wrapped Object (in) primitive to
String

Every wrapper class contains toString()

```
final = public String toString()
```

- Every wrapper class (including Character class) contains the above `toString()` method to convert wrapper object to string.
 - `toString()` is the overriding version of `Object` class `toString()` method.
 - Whenever we are trying to print wrapper object reference internally this `toString()` method only executed.

Q1 Integer := Integer Value Of ("10")

System.out.println(); // internally it calls toString()
and prints the Data

Lec 10

public static String toString (primitive)

- Every wrapper class contains a static `valueOf()` method to convert primitive to string.

String t = Integer.toString(10);
| primitive type int

```
es: String s= Integer.toString(10)
```

String 5: Boolean - `hosting` (`True`)

String s = character twisting("a");

Commonly used
pink rose
Value of

Finals : Integer and long classes contains the following static toString() method to convert the primitive to specified radix string form.

```
public static String toString(primitive p, int radix)
```

↳ 2 to 24

```
ex: String s= Integer.toString(15, 2)
```

1110

```
System.out.println(s); // 1110
```

new class

2 1 3 2

1 1 0 1

Finals : Integer and long classes contains the following toBinaryString() methods.

```
public static String toBinaryString(primitive p)
```

```
public static long toBinaryString()
```

```
public static String toBinaryString()
```

Example

```
classDemo class Demo {
```

```
    public static void main (String args) { ..
```

```
        String s1= Integer.toBinaryString(9) // 1001
```

```
        String s2= Integer.toBinaryString(10) // 1101
```

```
        String s3= Integer.toBinaryString(20) // 10100
```

```
        System.out.println(s1, s2, s3)
```

Note

String class

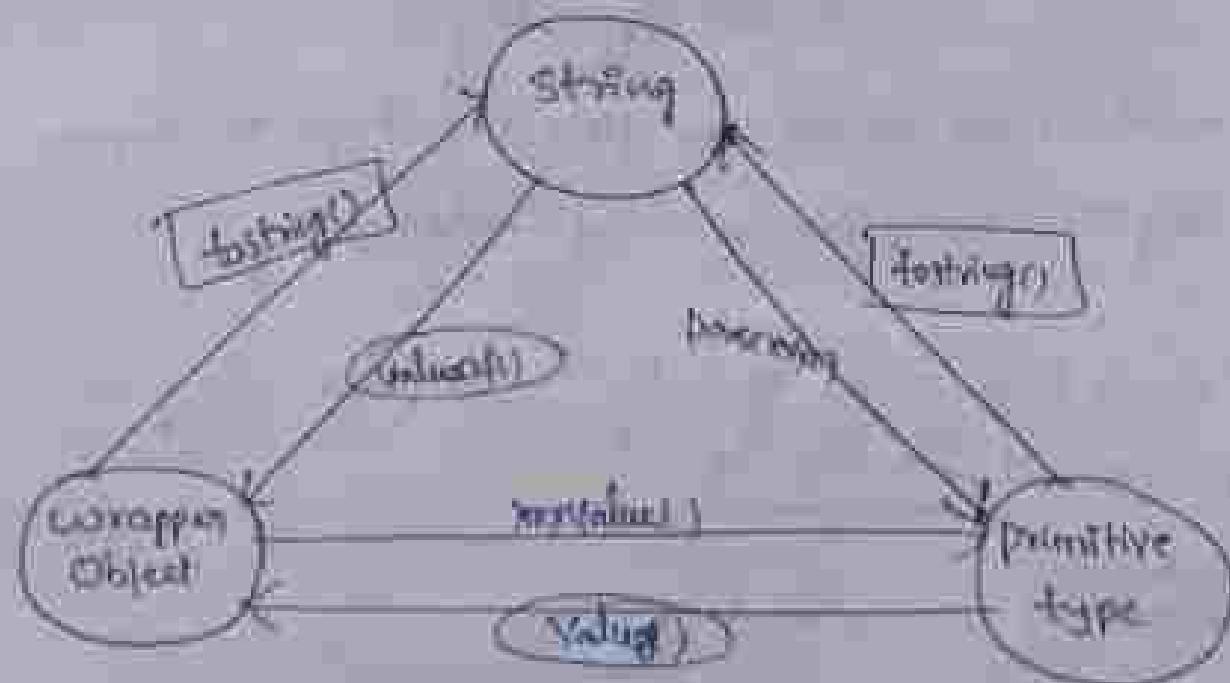
```
public static String valueOf (boolean)
public static String valueOf (char)
public static String valueOf (int)
public static String valueOf (long)
public static String valueOf (float)
public static String valueOf (double)
```

Class Name

```
String data = String valueOf (a); // static factory method
```

```
String data = "Hello".toUpperCase(); // instance factory method
```

Object calling



Conversion

Auto Boxing and Auto Unboxing

Q#1

```
Boolean b1 = Boolean.valueOf(true);  
if (b1)  
    System.out.println("Hello")
```

} in jdk 1.4

Q#2

```
ArrayList al = new ArrayList();  
al.add(10);
```

Compatibility error

Auto Boxing - close it, convert, put it up in box.

bottom same

autoboxing actual
Integer in = 10, // Integer it = Integer.valueOf(in)

Take primitive type wrap it up
in Object and keep it up in
Unrapper Object.

→ Automatic conversion of primitive type to unrapper
object by the compiler is called "Auto Boxing".

Note : Auto Boxing is done by the compiler using a
method called "valueOf()".

→ ~~AutoUnboxing~~
→ ~~Integer ii = new Integer(10);~~

int i = ii; // when ever data is unboxing
problem unbox convert one unboxing if

such giving it to primitive
(AutoUnbox Primitive can call)

Auto Unboxing
Phy. - (1) // Integer value (2)

(Auto Unboxing): Actually in bytecode

int i2 = 11 // int i2 = (1 - int value);

compiler converts Integer to int type using intValue()

Auto Unboxing

Automatic Conversion of wrapped object to primitive type by compiler is called "Auto Unboxing".

Integer i1 = new Integer(10)

int i2 = i1

compiler converts Integer to int type using intValue()

int i2 = i1.intValue();

Note:- Auto Unboxing is done by the compiler using a method called intValue()

Compiler will do the conversion automatically from
1. DLS Version



Autoboxing(values10)

Autounboxing(wrapper type))



Case 1

Class Test

```
Static Integer n=10; // Auto Boxing  
public static void main (String [] args)  
{  
    int i2 = n; // Auto UnBoxing  
      
    public static void m1 (Integer i2) // Auto Boxing  
    {  
        int k = i2; // Auto UnBoxing  
        System.out.println (k); // 10 - output  
    }  
}
```

Compiler is responsible for Conversion of primitive to wrapper and wrapper to primitive using the concept of "Auto Boxing" and "Auto UnBoxing".

Wrapper to wrapper
Reference will be given

Case 2

Class Test

```
Static Integer n; n=null  
public static void main (String [] args)  
{  
    int i1 = n; int i2 = i1.intValue(); // Null  
    System.out.println  
}
```

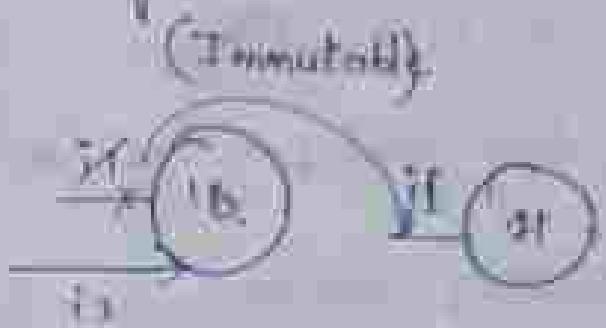
Case 3 :-

All wrappers are immutable

Integer $i_1 = 10$; // Auto Boxing

Integer $i_2 = 10$;

$i_1 + i_2 \Rightarrow i_1 = 10$ (+)



System.out.println(i_1);

System.out.println(i_2);

System.out.println($i_1 == i_2$);

Case 4 :-

Integer x = new Integer(10);

Integer y = new Integer(10);

System.out.println($x == y$); // false

Case 5 :-

Integer x = new Integer(10); // memory from heap area

Integer y = 10; // Auto Boxing \Rightarrow Integer y = Integer Value of 10;

System.out.println($x == y$); // false

Case 6 :-

Integer x = new Integer(10);

Integer y = x; \Rightarrow Reference 'y' is referred to printing to some object.

System.out.println($x == y$) is true.

Case 3 :

Integer x= 10;

Integer y= 10;

System.out.println(x==y);

Integer a= 100;

Integer b= 100;

System.out.println(a==b);

Integer i= 1000;



Integer j= 1000;



System.out.println(i==j);

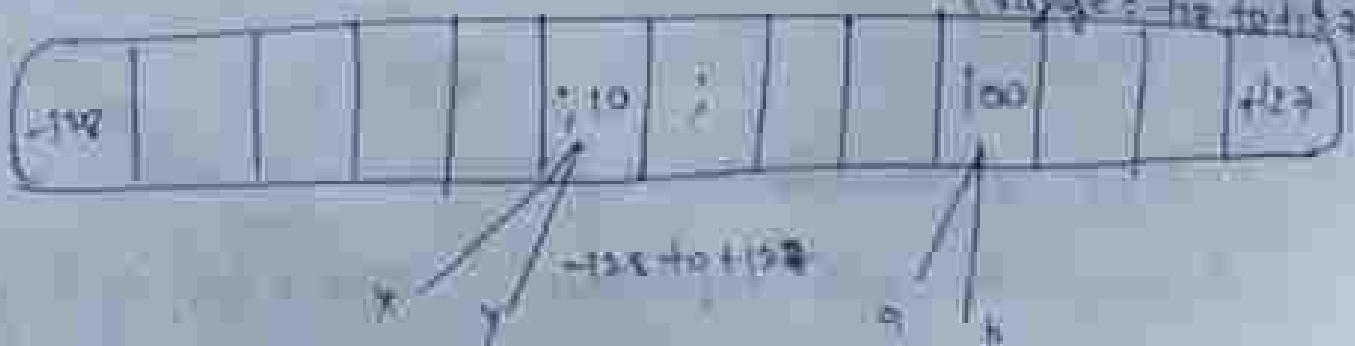
Compiler uses "ValueOf()" for AutoBoxing



Implemented in intelligent way in wrapper classes

at the time of looking class file
from with create buffer of object

buffer of objects to be used during AutoBoxing
(say x= 12 then ()



Note :- To implement auto-boxing concept Java wrapper class a buffer of object will be created at the time of class loading.

- During auto-boxing, if an object has to be created first JVM will check whether or not
- If it is available, the JVM will reuse the buffered object. Instead of creating a new object.
- If the object is not available inside buffer, then JVM will create a new object in the heap area, this approach improves the performance and memory utilization.

But this buffer concept is applicable only for ~~new few~~ few cases.

- Byte \Rightarrow (-128 to +127)
- Short \Rightarrow (-32768 to 32767)
- Integer \Rightarrow (-2147483648 to 2147483647)
- Long \Rightarrow (-2007223367504 to 2007223367503)
- Character \Rightarrow (0 to 65535)
- Boolean \Rightarrow true, false

In the remaining cases new object will be created

// String/primitive to wrapper \Rightarrow ValueOf()

// wrapper type to primitive \Rightarrow xxxValue()

In wrapped class \rightarrow immutable

Equality() \Rightarrow compare the content

toString() \Rightarrow print the content

* when compared with constructor it is recommended to use ValueOf() method to create wrapped object

IDE-IDE method overriding

IDE-IDE - no - max change in for the no. of arguments
new method should be written

Single method can handle any no. of
args. i.e. Single method can handle any no. of
arguments (all should be of same type)

(...)) All type before
variable

\rightarrow public void method(int... args)
s.o.p (new approach)

java called Var-args (ellipsis)

Var-args method

* Stands for Variable argument methods

In java language, if we have variable no. of
arguments, then compulsorily new method has to
written till IDEs.

But jdk 1.5 version, we can write single method
which can handle variable no. of arguments (but all
of them same type).

Syntax: method (data type ... VariableName) ...
... \Rightarrow ... stands for ellipse

e.g.

```
class Demo {  
    int add (int ...x) {  
        System.out.println ("Var-arg approach");  
    }  
}
```

Class Test

```
public static void main (String [] args) {  
    Demo d = new Demo ();  
    d.add ();  $\Rightarrow$    
    d.add ();  
    d.add (10, 20);  
    d.add (10, 20, 30);  
}
```

Output: Var-arg approach

Var-arg approach

Var-arg approach

Var-arg approach

Note: Internally the Var-arg method will convert to single dimension arrays, so we can access the Var-arg method arguments using index.

(from pg)
(from pg)

Q 2 : Class Demo

```
public void methodOne (int ... x) {
    int total = 0;
    for (int i = 0; i < x.length; i++) {
        total += x[i];
    }
    System.out.println ("The sum is " + total);
}

public static void main (String [] args) {
    Demo d = new Demo ();
    d.methodOne (); // the sum is 0
    d.methodOne (5); // The sum is 5
    d.methodOne (10, 10); // the sum is 20.
}
```

Q 3 : Class Demo

```
public void methodOne (int ... x) {
    int total = 0;
    for (int data : x) {
        total += data;
    }
    System.out.println ("The sum is " + total);
}
```

```
public static void main (String [] args) {  
    Demo d = new Demo();  
    d.methodOne (5); // 5  
    d.methodOne (10); // 10  
    d.methodOne (10, 20, 30); // 60  
}
```

Case 1

Valid Signatures

1. public void methodOne (int ... x)
2. public void methodOne (int ... x)
3. public void methodOne (int ... x)

Case 2 we can mix var arg with var argument

```
public void methodOne (int, int ... y) (Valid)  
public void methodOne (String s, int ... x)
```

Case 3 while mixing var arg with var argument var arg should be always last public void methodOne (int ... x, int ... y); (Invalid)

Case 4 In var argument list there should be only one var argument

```
public void methodOne (int ... x, int ... y); (Invalid)
```

Case 5 we can overload Var arg method, but Var arg method will get a call only if none of matches are found
(Just like default statement of switch case)

class Test {

```
    public void methodOne (int ...i) {  
        System.out.println ("Var arg method");  
    }  
    public void methodOne (int i) {  
        System.out.println ("Int arg method");  
    }  
    public static void main (String [] args) {  
        Test t = new Test();  
        t.methodOne (10); //Int arg method  
        t.methodOne (); //Var arg method  
        t.methodOne (10, 20, 30); //Int arg method  
    }  
}
```

Case 6 public void methodOne (int ...x) ⇒ it is
can be replace as int [] x

Case 7 public void methodOne (int ...x)
public void methodOne (int [] x)

Output CE because we cannot have two methods with same signature.

Single Dimension Array Vs Var arg method

1. but whenever Single Dimension Array is present we can replace it with Var arg.

Ex: `public static void main(String[] args) { }`

2. whenever Var arg is present we cannot replace it with Single Dimension Array.

Ex: `public void methodOne(String... args) { }` (invalid)

Note:

`int[] arr`

→ we can call to this method by passing group of int values and x will become 1D array (int[])

`int[] arr`

→ we can call to this method by passing 1D array only.

Note:

Ex:

`class Test`

`public void methodOne(int... x) { }`

`for (int data : x) { }`

`System.out.println(data);`

`class`

`public static void main(String[] args) { }`

`Test t = new Test();`

`t.methodOne(10, 20, 30);`

3

(In the above when x is handled as one-dimensional)

```

class Test {
    public void methodOne( int[]... x ) {
        for( int i = 0; i < x.length; i++ ) {
            for( int element : x[i] ) {
                System.out.println( element );
            }
        }
    }
}

public static void main( String[] args ) {
    Test t = new Test();
    int[ ] a = { 10, 20, 30 };
    int[ ] b = { 50, 60 };
    t.methodOne( a, b );
}

```

In the above program is declared as 2-D array

Note

methodOne(int... x)

→ we can call this method by passing a group of int Values, so it becomes 1-D array.

methodOne(int[]... x)

→ we can call this method by passing a group of 2-D int[], so it becomes 3-D array.

wrapper class

1. AutoBoxing

2. Widening (Implicit Type casting done by the Compiler (Applicable for primitive and wrapper type))

3. Var-args

Case 1: Widening Vs AutoBoxing

Class AutoBoxingAndAutoUnboxing

```
public static void methodOne (long l) {  
    System.out.println ("widening");  
}  
  
public static void methodOne (Integer i) {  
    System.out.println ("autoBoxing");  
}  
  
public static void main (String [] args) {  
    int x = 10;  
    methodOne (x); // primitive → do type casting  
    // → found →  
    // long (binding happens by compiler)  
}  
  
Output: widening
```

Case 2: Widening Vs Var-arg method

Class AutoBoxingAndAutoUnboxingDemo

```
public static void methodOne (long l) {  
    System.out.println ("widening");  
}  
  
public static void methodOne (int ... i) {  
    System.out.println ("Var-arg method");  
}  
  
public static void main (String [] args) {  
    int x = 10;  
    methodOne (x); // primitive → do type casting  
    // → found → long (binding happens by compiler)  
}  
  
Output: widening
```

Case 3 : AutoBoxing vs Var-arg method

class AutoBoxingAndVarInBoxingDemo {

 public static void methodOne(Integer i) {

 System.out.println("MethodOne")

 }

 public static void methodOne(Integer... i) {

 System.out.println("Var-arg method")

 }

 public static void main(String[] args) {

 int x=10;

 methodOne(x); // int → implicit type casting

 long aFloat, double

 // int → AutoBoxing→Object

 }

Case 4 : class AutoBoxingAndWidthBoxingDemo {

 public static void methodOne(Long l) {

 System.out.println("a long")

 }

 public static void main(String[] args) {

 int x=10;

 methodOne(x); // CE error: find the method

Point 2 : widening followed by narrowing is not allowed in

Java, but narrowing followed by widening is allowed.

Case 5:

Class: Autoblocking and the Unbinding Principle

```
public static void methodOne (Object o) {
```

```
System.out.println("object");
```

public static void main (String[] args) {

第 10 页

```
methedOne(x); // Autoloading → int → Integer
```

// widening \rightarrow Integer \rightarrow

Number;Object

out put-object

which of the following declarations are valid.

new Vs newinstance()

1. new is an operation to create an object or object if we know class name at the beginning.
2. newinstance() is a method returning class which can be used to create object.
3. If we don't know the class name at the beginning and it's available dynamically at runtime then we should use newinstance() method.

Q1 :- public class Test {

```
public static void main (String [] args) throws  
Exception {
```

// Take the input of the class name of which object
has to be created at the runtime

```
String className = args[0];
```

// Load the class file explicitly / implicitly
// className = Class.forName(className);

// for the loaded class object is created using
// zero parameter constructor only (what will be
// Object obj = className.newInstance());

// perform type casting to get Student Object

```
Student std = (Student) obj;
```

```
System.out.println (std);
```

})

If dynamically provide class name is not available then we will get the **RuntimeException** saying **Class Not Found Exception**

To use `newinstance()` method compulsorily corresponding class should contains no argument constructor otherwise we will get the **RuntimeException** saying "InstantiationException"

If the argument constructor is private then it would result in "IllegalAccessException" (CE-shouting)

Note: During type casting, if there is no relationship b/w a classes as parent-to child then it would result in "NoClassCast Exception".

Difference b/w new and newinstance()

new

`new` is an operation, which can be used to create an object.

we can use `new opname()` if we know the class name at the beginning.

```
Test t = new Test();
```

If the corresponding class file not available at Runtime then we will get **Runtime Exception** saying **NoClassFoundError**, it is unchecked

To used `new opname()` the corresponding class not required to contains no argument constructor.

newinstance()

`newinstance()` is a method, present in class `class`, which can be used to create an object.

We can use the `newinstance()` method, if we don't know class name at the beginning and available dynamically.

Runtime

`Object o = class.forName("arg0").newInstance();`

If the corresponding class file not available at Runtime - then we will get `RuntimeException` saying `ClassNotfoundException`.

To avoid `newinstance()` method the corresponding class should compulsorily contain no argument constructor, otherwise we will get `RuntimeException` saying `InstantiationException`.

Difference b/w ClassNotFoundException & NoClassfoundException

1. For hard coded class names at Runtime in the corresponding class file not available we will get `NoClassDefFoundException`, which is unchecked.

`Test t = new Test();`

At Runtime `Test` class file is not available then we will get `NoClassfoundException`.

2. For Dynamically provided class names at Runtime, if the corresponding class files is not available then we will get the `RuntimeException` saying

"Class Not found Exception"

Ex:- Object o= Class.forName("Test").newInstance();

Runtime if Test class like not available then we will get the ClassNot found Exception, which is checked exception.

Note :- new will create a memory on the heap area.

Student \Rightarrow JVM will search for student.class file in current working directory if found load the class file data into Method Area.

During the loading of .class file

a. static Variable will get memory set with default value.

b. static block gets executed

In the heap area for the required object memory for instance Variables is given JVM will set the default values to it

a. Execute the instance block if available

b. Call the constructor to set the meaningful values to the instance Variables.

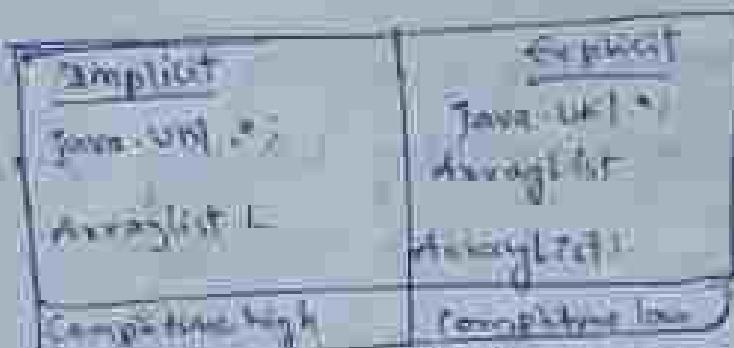
JVM will give the address of the object to hashing algorithm which generates the hashCode for the object and that hashCode will be returned as the reference to the programme.

`new` \Rightarrow required class details known to compiler
but not available at `JVM` then it would result in
"NoClassDefFoundError".

`NewInstance()` \Rightarrow required class details not
available at `JVM` then it would result in "ClassNot
FoundException".

as/also :

1. Import



2. `javac` \Rightarrow search for the required class information
in a) Current working directory
b) Did the programmer specified where
the class is available.

\Rightarrow we have ^{explicitly} import the package in where class
is present

`java.util.ArrayList al = new java.util.ArrayList();`
fully qualified path.

Import means - bring the class to our current
working directory

Informing the compiler plz search for `ArrayList`
class in
"java.util".

```
import java.util.ArrayList; // explicit import
import java.util.*; // implicit import
```

Lambda Expression & Interface of functional programming

Ques

Import Statement

```
class Test {
```

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

```
        ArrayList l = new ArrayList();
```

```
}
```

Output:

Compile time error

because ArrayList (class) is not there
in current directory.

→ we can resolve this problem by using fully
qualified name "java.util.ArrayList"

l = new java.util.ArrayList(); But problem with
using fully qualified name every time is it increases
length of the code and reduces readability.

→ we can resolve this problem by using import
statements

Example :

```
import java.util.ArrayList;  
class Test {  
    public static void main (String args[]) {  
        ArrayList L = new ArrayList();  
    }  
}
```

Output : b:\Java>javac Test.java

Hence whenever we are using import statement it is not required to use fully qualified names we can use short names directly this approach decreases length of the code and improves readability.

Case 1 : Types of Import Statements

There are 2 types of import statements.

- 1) Explicit Class Import
- 2) Implicit Class Import

Explicit Class Import

Example : `import java.util.ArrayList;`

⇒ This type of import is highly recommended to use because it improves readability of the code.

⇒ Best suitable for desktop users where readability is imp.

Implicit class import

Example: `import java.util.*;`

- ⇒ It is never recommended to use because it reduces readability of the code.
- ⇒ Best suitable for students where typing is important.

Case 2:

which of the following import statements are meaningful

- a. `import java.util;`
- b. `import java.util.ArrayList;` classid
- c. `import java.util.;`
- d. `import java.util.ArrayList;`

Case 3: Consider the following code.

```
class MyArrayList extends java.util.ArrayList
```

- ⇒ The code compiles fine even though we are not using import statements because we used fully qualified name.
- ⇒ whenever we are using fully qualified name it is not required to use import statement. Similarly whenever we are using import statements it is not require to use fully qualified name.

Case 4 :

```
import java.util.*;  
import java.sql.*;  
  
class Test {  
    public static void main (String args) {  
        Date d = new Date();  
    }  
}
```

the output -

Reference to Date is ambiguous both class java.sql.Date in java.sql and class java.util.Date in java.util match

```
Date d = new Date();
```

Note : Even in the list case also we may get the same ambiguity problem because it is available in both util and sql packages.

Case 5 : while resolving class names compiler will always gives the importance in the following order.

1. Explicit class Import
2. Classes present in current working directory
3. Implicit Class Import

Example :

```
import java.util.Date;  
import java.sql.*;
```

Class Test f

```
public static void main (String args[])
{
    Date d = new Date();
}
```

The code compiles fine and in this case util package Date will be considered.

Case 6 :-

Whenever we are importing a package all classes and interfaces present in that package are by default available but not sub package classes.

Java

| => util

| => Scanner class, ArrayList class, linkedList class

| => Regex

| => pattern class

To use pattern class in our program directly which import statement is required?

- a. import java.*;
- b. import java.util.*;
- c. import java.util.regex.*; //Valid (implicit)
- d. import java.util.regex.Pattern; //Valid (explicit)

Note :-

- if we refer to only .class files not sub packages - class files.

Case 7 :- In any java program the following 2 packages are not require to import because these are available by default to every java program

1. Java lang package
2. default package (current working directory)

Case 8 :- "Import statement is totally compile time concept" if we do not imports anywhere then more will be the compile time but there is no change in execution time.

Difference b/w C language's include and java language's import

1. It can be used in C/C++
2. At Compile time only compiler copy the code from Standard library and placed in current program.
3. It is Static Inclusion.
4. wastage of memory

Ex: <#include < > >

Import

1. It can be used in Java.
2. A runtime JVM will execute the corresponding Standard library and use its result in current program.
3. It is dynamic inclusion.
4. No wastage of memory.
Ex: <jsp:include>

Note:- In the case of C language #include all the header files will be loaded at the time of include statement hence it follows static loading. But in Java import statement no ".class" will be loaded at the time of import statements in the next lines of the code whenever we are using a particular class then only corresponding ".class" file will be loaded. Hence it follows "dynamic loading" or "load-on-demand"(sol) "load-on-fly".

JDK 1.5 Versions new features

1. for each
2. Var-arg
3. Outer
4. Generics
5. Auto boxing and Auto Unboxing
6. Co-variant return types.

Annotations

- 1. Enum
- 2. static import
- 3. String builder.

Static Import: This concept introduced in 1.5. According to sun static import improves readability of the code but according to worldwide programming experts (like us) static imports creates confusion and reduces readability of the code. Hence if there is no specific requirement never recommend to use a static import.

Usually we can access static members by using class name but whenever we are using static import it is not require to use class name we can access directly without static import.

class Test

```
public static void main (String args[])
{
    System.out.println (math.sqrt (4));
    System.out.println (math.max (15,20));
    System.out.println (Math.random ());
}
```

Output :- 2.0, 20, 0.5413561

* With static imports

Import static java.lang.Math.sqrt;

Import static java.lang.Math.*;

Class Test 4

```
public static void main (String args[]){  
    System.out.println (sqrt (4));  
    System.out.println (max (0,20));  
    System.out.println (random ());  
}
```

Output :- 2.0, 20, 0.5413561

Class Test 5

Static String name = "Sachin";

Output :- Test.name.length () \Rightarrow 6

Import java.io.PrintStream;

Class System {

static PrintStream out;

Class PrintStream {

public void println () {

System.out.println()

↳ It is a method of **PrintStream** class

↳ It is a reference of **PrintStream** class

↳ It is a class

Example 3:

```
import static java.lang.System.out;
class Test {
    public static void main (String args[])
    {
        out.println ("Hello");
        out.println ("Hi");
    }
}
```

Output : Hello Hi

Example 4:

```
import static java.lang.Integer.*;
import static java.lang.Byte.*;
import static java.lang.System.out.println;
class Test {
    public static void main (String args[])
    {
        System.out.println (maxValue);
    }
}
```

Output: Ambiguous ↗

Byte maxValue (maxValue);

Note :

Two packages contain a class or interface with the same name hence ambiguity problems are very rare in normal import.
But a class or interface can contain a method so very common here variable with the same name is very common. In static ambiguity problem is also very common. In static import while resolving static members compiler will give the precedence in the following order.

1. Current class static members

2. Explicit static import

3. Implicit static import

e.g. `import static java.lang.Integer.MAX_VALUE;`
`import static java.lang.Byte.SIZE;`

class Test {

 static int MAX_VALUE = 999;

 public static void main (String args) {

 System.out.println (MAX_VALUE);

3)

Which of the following import statement is valid?

```
import java.lang.math.*; //invalid
import static java.lang.math.*; //valid
import java.lang.Math; //valid
import static java.lang.math.*; //invalid
import static java.lang.Math.sqrt.*; //invalid
import static java.lang.Math.sqrt; //invalid
import java.lang.Math.Sqrt; //invalid
import static java.lang.Math.Sqrt(); //invalid
import static java.lang.Math.Sqrt(); //valid
```

Usage of static import reduces readability
and creates confusion hence it is not
a specific requirement never be recommended to
use static import

What is the difference b/w general import
and static import

Normal → we can use normal import to import
classes and interfaces of a package
→ whenever we are using normal import we
can access class and interfaces directly by
their short name it is not required to use
qualified names

Static import

→ we can use static import to import static
members of a particular class

→ whenever we are using static import it is not

not require ~~you~~ than name the constructor.

Static members directory

Lambda Expression

Lambda Expression → Functional Interface → Interface
Inheritance Abstraction

In interface if one abstract method is present we can call it has functional interface

Interface Demo

Void disp(); → functional interface

↑

functional interface and lambda expression work together.

② Functional Interface → It will inform to

Compiler and developer like it is an functional interface

② Functional Interface

Interface Demo

Void disp();

Void disp();

↑

Compiler error because we mentioned above my functional interface it means that interface has only method.

↑

use ~~can't~~ Lambda expression without function
placeholder.

Lambda expression = a short block of code
which takes in parameters and returns a value

In Java ↗

→ ↗ arrow operator
↳ Lambda Operator

before

void disp(λ)
s.o.p ("Hello")

To write Lambda expression

we need to use (\rightarrow) - min and $>$ greater

Anonymous method means unname method

parameters ↗ → ↗ ↗
↳ Body
↳ Lambda Operator

we need to write Lambda expression

i) use \rightarrow

ii) divide 3 parts.

a. left of lambda operator parameters
required

b. Right of lambda operator body requires
{} ↗ ↗

$(\rightarrow \{ s.o.p ("Hello") \})$

normal method

```
Void disp()
  S.o.p ("Hello")
```

In method we have only one statement
then `{}>` brackets we can ignore

\Rightarrow `S.o.p ("Hello")`

lambda Expression can't be write alone
lambda expression and functional interface
work together.

@Functional Interface

Interface Demo \Rightarrow

```
Void disp()
```

\uparrow

main Uf

Demo `cl = () -> S.o.p ("Hello");` (without `{}`)
`cl. disp();` because one statement
(\Rightarrow)

Demo `cl = () ->`

```
  S.o.p ("Hello")
```

(we can write with `{}`)

```
}
```

```
)
```

even one statement is

it's fine. (there is no `{}>`)

for functional interface we can write anonymous
own inner class instead of lambda expression

Class Demo{

 Anonymous inner class

 void disp();

}

 main (){}

 Demo d = new Demo (){

 public void disp(){
 System.out.println ("Hello");

}

 }

 d. disp();

for functional interface total 2 ways

one - Normal Tradition

2. Lambda expression

3. Anonymous inner class

But lambda expression is able to write only
one abstract method is present

Annotations :-

@FunctionalInterface

interface Demo{

 void add (int a, int b);

}

main(14)

→ we can make

Dimn d = (int a, int b)

Int res = a+b;

S.o.p (res);

3)

8. odd (10,20);

Parameter & Return type

Introduce Sub

4

Int Sub (int num);

5

main() {

Sub s = (num); → if

int res = num; → s;

return res;

6)

S. sub (10); → we can store also

S. sub (C. sub (10)); → we can directly print value.

7) To write lambda exp we need to use

lambda operator →

8) Lambda operator divided into points to
write lambda exp.

9) left side of lambda operator () = we
needed

if one parameter is
there then no need of
()

num → num 1 = s;

return statement

is not needed.

- 4) Right side of lambda operator Quady is needed
- 5) left side of single parameter is there no need of ()
- 6) left side of parameters data types is optional.
- 7) Right side of body contains one statement than {} is optional
- 8) Right side in body of single line implementation then return type is optional.
- 9) {} is mandatory in case if there is return statement explicitly present and in body code than the statement is present
 - i) used to compute length of string
- 10) Group 2.2
Interface Demo

```
Int odd (Int a, Int b);
```

Class Demo1 implements Demo1

```
public Int odd (Int a, Int b) {  
    Int res = a+b;  
    return res;
```

main()

Demo d = new Demo(10)

d.add(10, 20) // To store return value
s.o.p(d.add(10, 20)) // To display return value

Lambda

main() {

 Demo d = (a, b) \Rightarrow res = a+b;

 s.o.p(d.add(10, 6));

Inner class

main() {

 Demo d = new Demo();

 public int add (int a, int b) {

 int res = a+b;

 return res;

 s.o.p(d.add(10, 20));

Method hiding

- Static methods do participate in inheritance
Yes, but we can't override if we do that
It will consider as Specified method.
- It is called method hiding

→ class parent {

 public void static disp()

 public static void disp()

 System.out.println("Hello parent");

}

class Child extends parent

 public static void disp()

 System.out.println("Hello child");

public class MethodHiding {

 public static void main (String [] args) {

 parent p = new Child();

 p.disp();

}

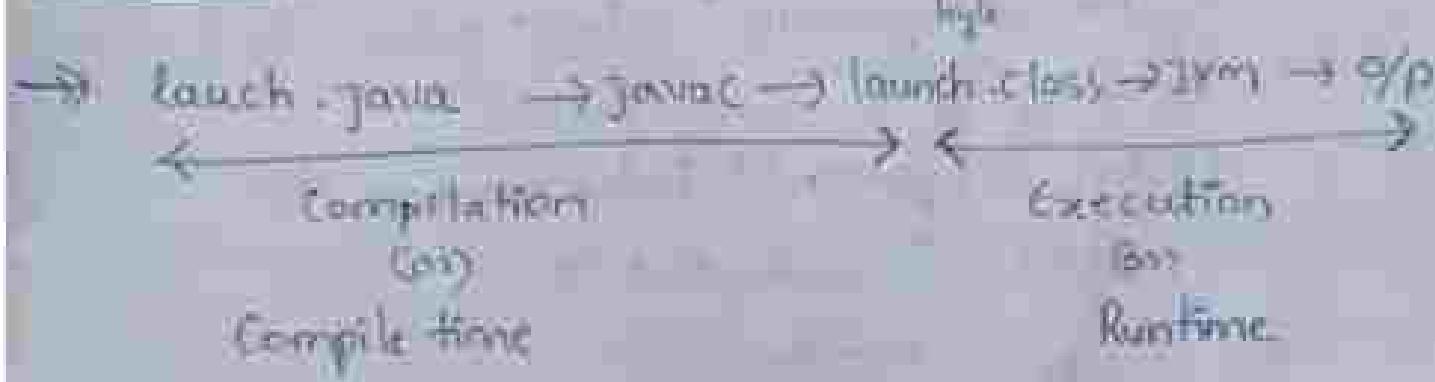
out put : Hello parent, (we can't override
Static methods)

Exception Handling

If Java program is in executing that means it is running.

→ try, catch, throws, finally

web app = Without internet we can't run the app



Error : Because of Some language Syntax

(i, f, void) it will check at compilation
Compile time error

Runtime error : i) Runtime error → because of logical mistake but $\approx 10\%$ arithmetic error
ii) $\approx 80\%$
Runtime
logical mistake (by zero division)

Exception : @-fault input by user (or) developer

In between only application will exit is called exception

② Exception refers to mistake that will occur during runtime of our application which will result in abrupt termination.

Exception Example ① (type and check this code)

```
Ex:- import java.util.*;
public class Demo {
    public static void main (String [] args) {
        Scanner scan = new Scanner (System.in);
        System.out.println (" Be divided operation app");
        System.out.println (" Enter number 1");
        number1 = scan.nextInt();
        System.out.println (" Enter number 2");
        number2 = scan.nextInt();
        int divide = 
$$\frac{\text{number1}}{\text{number2}}$$
;
        System.out.println (divide);
        System.out.println (" Operation over");
    }
}
```

Output :- Number 1 is entering number

① $\frac{100}{0}$

Number 2 is

Output = 10

② Case

Number = 100

Number = 0

Output = -

Exception will come means abruptly terminated by telling (Arithmetic exception by $0/0$, at main method) like this.

So we have handle exception without abrupt termination, then exception handling came to picture

Exception Handling: It is process to handle not to abrupt termination should not happen

try - catch - throws - finally \Rightarrow keywords

We are going to use in Exception handling.

Exceptions are two types

- 1) checked exception: Compiler only checks the risky code it will tell like (hey developer you are doing some risky please handle it) At
- 2) unchecked exception: Compiler will not check it will tell like (hey developer you can headache from compiling and giving you - you do with jvm at runtime) During runtime it will check and throw an exception. like example ①

→ How to handle exception :- ~~try - catch - finally~~

try {

... Both case we should write we can't write only one alone

↳

(without we should give meaningful)

Catch (which classes work)

{

}

}

It will handle try block

try { } - trying block without even risk code we have to write there

catch (class) { } - exception handling

Exception handling in this example

Example 3

```
public class S
```

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

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

```
        System.out.println ("division operation app");
```

keeping
input
into
block

```
        try {
```

```
            System.out.println ("Enter num1");
```

```
            num1 = Scan.nextInt();
```

```
            System.out.println ("Enter num2");
```

```
            num2 = Scan.nextInt();
```

```
            int res = num1 / num2;
```

```
            System.out.println (res);
```

```
        } catch (Exception e) {
```

```
            System.out.println ("please don't enter non-integer");
```

```
        } System.out.println ("app operation terminated over");
```

Output:
num1 = 100 num2 = 10
res = 10 operation over

(Case ②)

```
num1 = 100
```

```
num2 = 15
```

Output: Please don't enter non-integer
app operation over terminated.

Assume: If any risky code is there then we will put in try block
Catch block: it will handle the try block
if there is any exception then only catch block will execute otherwise try block will execute

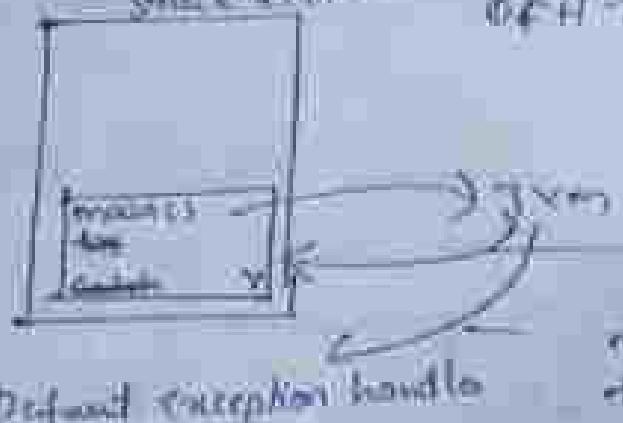
if exception statement is there in try block then from that exception line till catch block block line no statement will not execute directly catch block execute
Ex: No lines code

```
try{  
    5 line -> exception -> then 6, 7, 8 will not execute  
    directly catch block ⑨  
    case ⑩ line -> catch block  
    {  
        9 line -> and after catch  
        block lines  
        ⑪  
    }  
    ⑫ -> After catch block
```

If there is no exception in try the catch block will not execute
but if exception or no exception but after catch block lines will execute.

Stack frame

OFH - Default exception handler

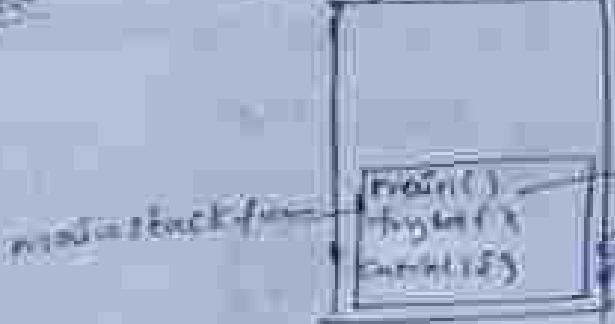


JVM will check it not

matter how to DEH
then JVM throw exception

when ever some exception raises in main method
where ever that exception line it will throw create
object and throw to JVM, then after JVM will
check again in that Stack frame like whether
programmer handled exception or not, if not then
go to default exception handler.

Stack frame



JVM will check it if not

matter will not get DEH

DEH

when ever some exception raises in uncaught stack-frame
where ever that exception line it will create object and
throws to JVM, then after JVM will check again
in that stack-frame like whether programmer handled
it not, if he handled it will not go default exception
handler.

For one try block we can write no of catch block

For one try block we can write only one catch block also

day 2 "Exception handling"

Example with cases

Case 1

Statement -1; "if there is no exception raised"

try {

→ Statement -1, 2, 3, 4, 5, 6 will be executed
resulting in normal termination

Statement -2;

"if an exception is raised at statement
and the corresponding catch block is
matched"

Statement -3;

→ Statement -1, 2, 4 & 5 will be executed
resulting in normal termination

}

"If an exception is raised at statement
and the corresponding catch block is not
matched"

Statement -4;

→ Statement -1, 2 will be executed
resulting in abnormal termination

}

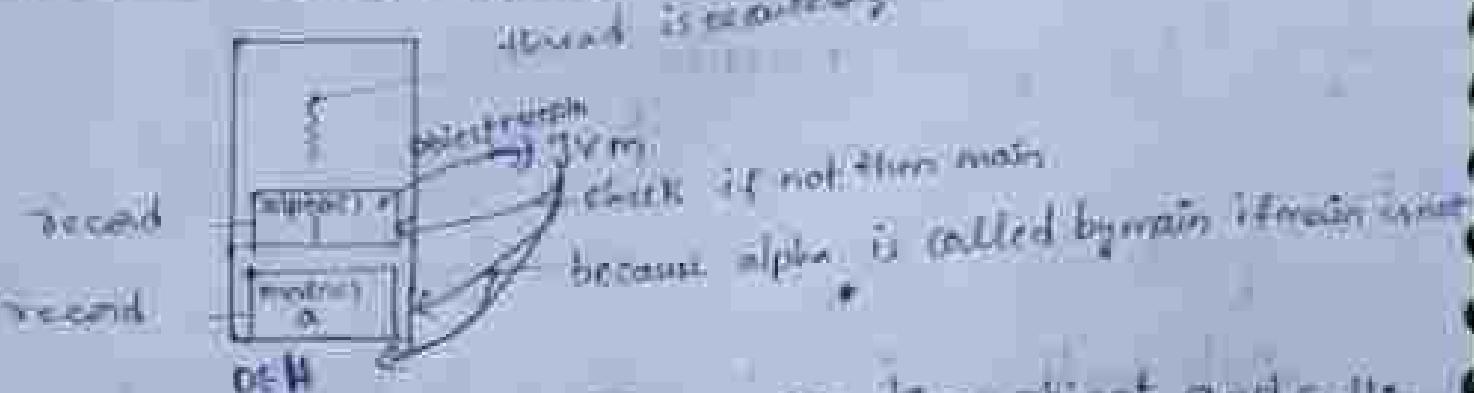
"If an exception is raised at statement
at Statement -3 or Statement -6"

Statement -5;

→ Statement -1 & 5 is not a part
of try block Resulting in abnormal
termination

→ we create class in that the Alpha contains some code, and from main method use some calling the alpha method.

Runtime Stack B there



In main method I am creating object and calling the alpha object

if anything wants execute the entire ~~code~~ brought to runtime stack the execute will execute

if alpha method contains exception it will catch alpha object give it JVM

and JVM will again in alpha method whether is exception handled or not. if not handled the JVM will check in main method because main method is calling alpha.

Note: In Alpha it will check if exception not handled in Alpha, JVM will check in main method, main method is one who is calling Alpha. If it is exception not handled in main it goes to default exception handling.

Example exception

Exception in thread "main" java.lang.ArithmaticException
by 200 at Alpha.alpha (LaunchException.java:17)
at LaunchException.main (LaunchException.java:26).

(DE+)

hcy in main throw some exception is there
Matter went to Alpha and after main method they both
not handled so matter come to Default exception handle

→ whenever there is an exception

- ① Handle Exception (try { } - catch (→))
- ② Duck the Exception (throws)
- ③ Re-throwing an Exception (throw, throws, try, catch, finally)

→ {try, catch, throws, finally}

throw exception	Block
class Vijay { Class Alpha() { } } int a = 10; int b = 0; int c = a/b; - exception System.out.println(c); }	class Vijay { } } void beta () throws exception { } } Alpha a = new Alpha(); } } a.alpha(); public class Main { public static void main () { } } Beta b = new Beta(); } } b.beta();

In above example

we have to write throws exception if you not handling the exception for good practice below who is the caller of that method no need check entire code.

when it is calling only, it get to know exception by there it has to handle and again if it is not handled then who is calling it will go to them.

If no one is handling it will go to default exception handling (DEH)

It will throws exception automatically but to be more safe (throws exception),

who ever writing code he has to handle the exception compulsorily (no ducking)

Ducking means throwing exception without handing to who calling that if it is unchecked exception

unchecked : Compiler will not force us to handle

checked : Compiler will force to handle that exception

throws Thread.sleep(6000); compiler error.

It will sleep 60 sec

nothing will happen but still compiler will force to handle just indicating to caller

Now check it (throws exception)

throws → Be throwing to throwing already

handled exception. Object to caller

in the try catch block write (throws e)

After throws keyword whatever it is it will not execute.

So execute or that we have to write in finally block (finally f y) to close finally block we can't write alone with try block we can.

~~try & catch~~ → to handle exception

throws → check if method signature

throws → Catch → throws

finally → Close resources

→ What is exception object (contd)

Name of the exception: ArithmeticException

Description of the exception: / by zero

Stack Trace of Demo.alpha (Launch Java : 52)

The exception: launch.main (Launch.java:57)

method to print Exception Information

Throwable

getMessage()

toString()

printStackTrace()

1. e.getMessage() prints the description of the exception
Example: / by zero

2. e.toString() prints the name and the description of the exception

Example: ArithmeticException: / by zero

3. e.printStackTrace() prints the name and the description of exception along with the stack trace
Example: ArithmeticException: / by zero
at Demo.alpha()

finally →① if no exception - execute

→② if exception & catch match

→③ if exception & catch not match

to close connection we use finally { }

after return statement below statement will not execute

so we write in finally block if
will dominate the return and execute

System.exit(0); — total JVM will close

total Java application will close

Exception Caused	Exception Handled	finally block Executed
no ✗	— ✗	Yes ✓
Yes ✓	Yes ✗	Yes ✗
Yes ✗	No ✗	Yes ✓

↑ ↓

finally block Executing types

Statement-1)	<u>Case 1</u> → If there is no Exception raised → Statement-1,2,3,4,5 & 7 will be executed Resulting in Normal Termination
Statement-2)	<u>Case 2</u> If an exception is raised at Statement-3 and the corresponding catch block is matched → Statement-1,2,5 & 7 will be executed Resulting in Normal Termination
Statement-3)	<u>Case 3</u> If an exception is raised at and catch block matched → Statement-1,2,6 & 6 will be executed Resulting in Abnormal Termination
Statement-4)	<u>Case 4</u> If an exception is raised at Statement-5 → Resulting in Abnormal Termination But before that finally block will be executed.
Catch (xxxx)	
Statement-5)	
finally	
Statement-6)	
Statement-7)	
DE H	

Differences b/w throw & throw

throws

→ throw keyword is used to
explicitly throw an exception to
the JVM.

→ throw keyword is followed by
an instance of throwable or a
subclass of throwable.

→ throws keyword is used
to declare an exception to
delegated exception handling
responsibility to the caller.

throws keyword placed within the method body

multiple exceptions can be thrown with a single thrown clause

used for throwing an exception

```
try {  
    catch (Exception e) {  
        throw e;  
    }  
}
```

throws keyword is followed by exception class name

throws keyword issued with the method signature

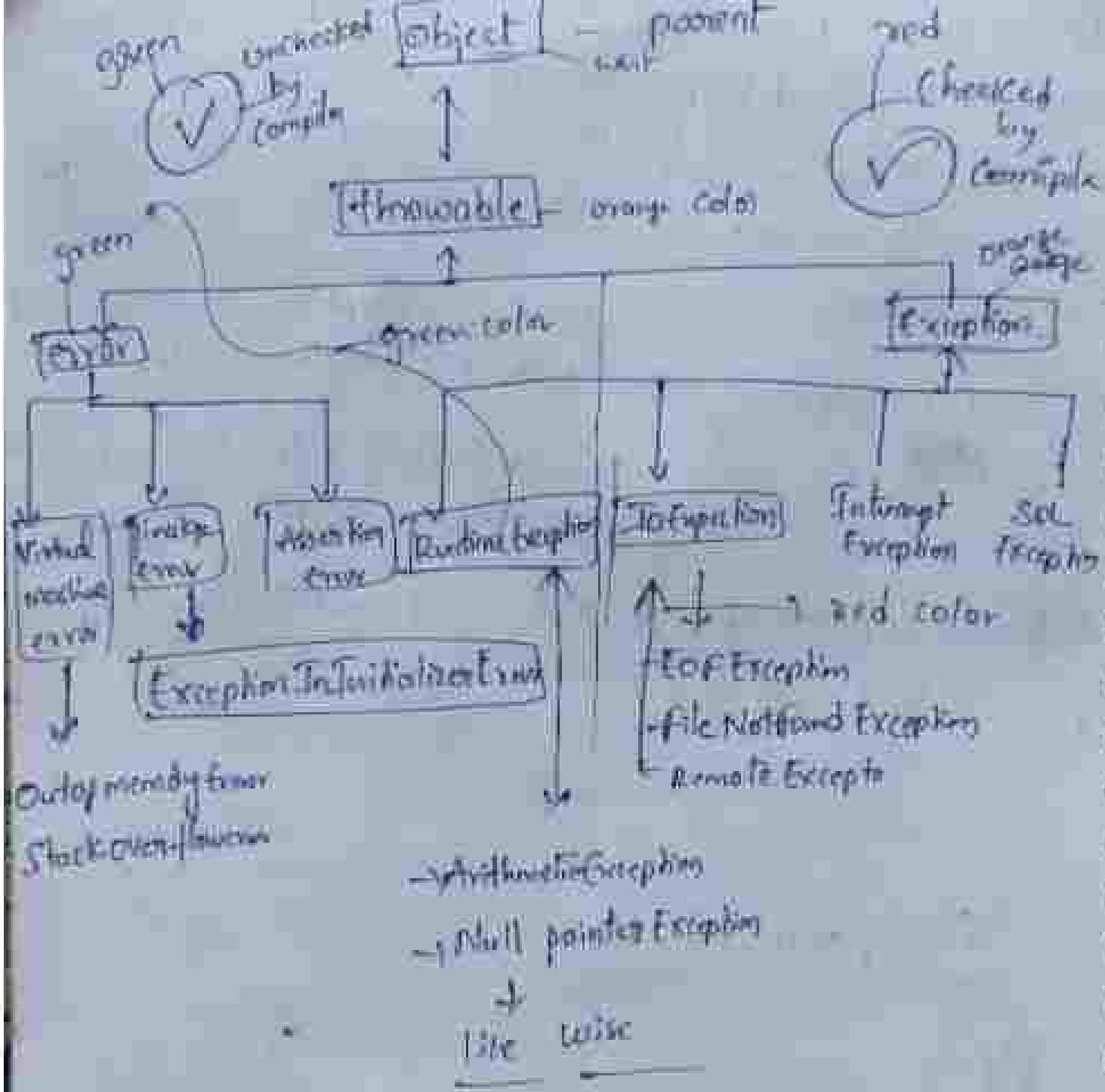
multiple exceptions can be declared with a single throws clause

used in both rethrowing and catching an exception

```
try {  
    void test() throws Exception {  
        //  
    }  
    catch (Exception e) {  
        //  
    }  
}
```

Exception Handling

```
try {  
    risky / suspicious code  
    catch (Exception e) {  
        handling / alternate / pre cautionary code  
    }  
    finally {  
        Resource deallocation / clean up code.  
    }  
}
```



↳ Unchecked exception

→ fully checked exception

↳ possibly checked means by this:-

So, Emperion class is partially checked.

Day 3 Exception Handling

Nested try - catch and finally

try {

Statement-1;

Statement-2;

Statement-3;

try {

Statement-4;

Statement-5;

Statement-6;

}

Catch (xxx e) {

Statement-7;

}

finally {

Statement-8;

}

Statement-9;

}

Catch (yyy e) {

Statement-10;

}

finally {

Statement-11;

}

Statement-12;

Case 1 : If no exception occurs

→ Statements-1,2,3,4,5,6,7,8,9,11,12 will be executed resulting in Normal Termination.

Case 2 : If an exception occurs at

Statement-2 and the corresponding catch block is matched

→ Statements-1,10,11 & 12 will be executed resulting in Normal Termination.

Case 3 : If an exception occurs at

Statement-2 and the corresponding catch block is not matched

→ Statements-1 & 11 will be executed resulting in Abnormal Termination

Case 4 : If an exception occurs at

Statement-5 and the corresponding inner catch block is matched

→ Statements-1,2,3,4,7,8,9,11,6,12 will be executed resulting in Normal Termination

Case 5 : If an exception occurs at statements and the corresponding inner catch block is not matched, but other catch block in

→ Statements-1,2,3,4,5,10,11,12,13,14 will be executed resulting in Normal Termination.

Case 6 → If an exception occurs at statements-5 and both inner and outer catch blocks are not matched
→ Statements -1,2,3,4,6 & 11 will be executed
Resulting in Abnormal Termination

Case 7 : If an exception occurs at statement-7 and the corresponding catch block is matched
→ Statements - 1,2,3,6,8,10,11,12 will be executed with normal termination.

Case 8 : If an exception occurs at statement-7 and the corresponding catch block is not matched.
→ Statements - 1,2,3,6,8,10,11 will be executed with Abnormal Termination.

Various possible Combinations of try-catch-finally

Case 1
only try
try
{
}
Not Valid and
Valid Syntax
of try-catch-
finally.

Case 2
only catch
catch(XXX e)
{
}
Not Valid

Case 3

only finally

finally

{} (Not Valid)

}

Case 4 try-catch

try

{}

} (Valid)

catch(XXX e)

{}

}

Case 5

Reverse order

catch(XXX e)

{}

} (Not Valid)

}

try

{}

}

Case 6 multiple try

try {}

try {}

(Not Valid)

catch(XXX e) {}

Case 7

multiple try

try {}

catch(XXX e)

{}

} (Not Valid)

try

{}

Case 8

multiple try-catch

try {}

catch(XXX e) {}

(Valid)

try {}

catch(XXX e) {}

Case 9

multiple catch

try {}
catch (xxx e) {}
catch (yyy e) {}

Conc: to multiple catch

try {}

catch (xxx e)

{

}

(Not Valid)

catch (yyy e)

{

Y

Case 10

multiple Exception in one catch

try {}
catch (xxx1 yyy1 e) {}
f
y

Conc: try - catch - finally

try

{

(Valid)

}

catch (xxx e) {}

finally {}

Case 11

try - finally

catch (xxx e) {}
finally {}

Conc: try - finally

try {}

finally {}

(Valid)

Case 12

try

finally {} (Not Valid)

catch (xxx e) {}

Y

Case 16:

try multiple
Catch-Finally

```
try { }  
    catch (Exception e) { }  
    finally { }  
    catch (IOException e) { }  
    finally { }
```

Case 17: Multiple Finally

try { }

Catch (Exception)

```
{ }  
finally { }  
finally { }
```

Case 18: Statements in-between try-catch-finally

```
try { }  
    System.out.println("Init");  
    catch (IOException e) { }  
    System.out.println("Hello")
```

(Not Valid)

```
finally { }  
    System.out.println("Hello")  
    finally { }
```

Case 19: only try with resource

try (R) {} Valid

Case 20: Nested try-catch-finally

```
try { }  
    try { }  
        catch (Exception e) { }  
    finally { }  
    catch (IOException e) { }  
    finally { }  
}  
}
```

Synchronous Exceptions

Occurs at a specific program statement.

Class launch {

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

```
    String str = null;
```

```
    str.toUpperCase();
```

Output

java.lang.NullPointerException
at launch.main (Launch.java:6)

Class Launch {

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

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

```
    a[5] = 10;
```

Output

java.lang.ArrayIndexOutOfBoundsException
(exception at launch.main (Launch.java:6))

Asynchronous Exceptions

Occurs anywhere in the program.

Class launch {

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

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

```
    System.out.println ("Enter name: ");
```

```
    String name = scan.nextLine();
```

Output

Enter your name:
Sachin

Example

Keyboard

Interrupt

Enter your grades:

50 60 70 80 90
Keyboard interrupt

Exception in thread "main" terminate batch job (y/n):

QUESTION

Custom Exception

User defined exception

parent class of all classes

class throwable extends Object

{

 String msg; (contd) Specialized section

 public throwable(String msg){

 this.msg=msg;

 one method

 }

(msg)

class Exception extends throwable.

Exception(String msg)

 Super(msg);

class Ict extends Exception

Ict(String msg){

 Super(msg);

while creating printing String ()

7

~~Ex:~~ Application Pto

```
import java.util.Scanner;  
class applicant
```

example on custom exception

```
→ import java.util.Scanner;  
class UnderAgeException extends Exception  
{  
    public UnderAgeException (String message)  
    {  
        super (message);  
    }  
}
```

```
class OverAgeException extends Exception
```

```
{  
    public OverAgeException (String message)  
    {  
        super (message);  
    }  
}
```

class applicant

```
    int age;  
    public void input()  
    {  
        Scanner scan = new Scanner (System.in);  
        System.out.println ("please enter your age");  
        age = scan.nextInt();  
    }
```

```
Void Verify() { throws UnderAgeException, OverAgeException }
```

```
if (age < 18) {
```

```
UnderAgeExceptioniae = new UnderAgeException("oh, the  
System.out.println(iae.getMessage());  
throws iae;
```

```
else if (age > 60) {
```

```
OverAgeExceptioniae = new OverAgeException  
System.out.println(iae.getMessage());  
throws iae;
```

```
else {
```

```
System.out.println("you are eligible");
```

```
}
```

```
public void init() {
```

```
Applicant a = new Applicant();
```

```
try {
```

```
a.input();
```

```
a.verify();
```

```
catch (UnderAgeException|OverAgeException e)
```

```
try {
```

```
a.input();
```

```
a.verify();
```

```
catch (UnderAgeException|OverAgeException e)
```

```
System.out.println("Don't even try again!");
System.exit(0);
```

```
public class main{}
```

```
public static void main(String[] args) {
    File file = new File("file.txt");
    file.createNewFile();
```

Topic 4: Exception Handling

- 1. try with resource
- 2. try with multiple-catch block
- 3. Rules of Overriding associated with Exception
- Remaining topics to be discussed.
- 4. instanceof Vs instanceof (object obj)
- 5. Aim is to Create a user-defined package and in real time project how it is used!

Resource - a inbuilt class for which we create object using a Object you made a call to those method to perform some operation. It called resource buffer reader in a resource.

1.4 Version Enhancements

1.7.4

1. try with `resources`
2. try with `multicatch` block

Until `jdk-1.6`, it is compulsorily required to write `finally` block to close all the resources which were open as a part of `try` block.

```
eg BufferReader br=null  
try {  
    br = new BufferedReader(new FileReader("abc.txt"));  
    catch (IOException ie) {  
        ie.printStackTrace();  
    }  
    finally {  
        try {  
            if (br!=null) {  
                br.close();  
            }  
        }  
        catch (IOException ie) {  
            ie.printStackTrace();  
        }  
    }  
}
```

problems in the above approach

1. Compulsory the programmer is required to close all opened resources which increases the complexity of the program.
 2. Compulsory we should write finally block explicitly, which increases the length of the code and reduces readability. To overcome this problem sun has introduced try with resources in "1.7" version of jdk.
- ## try with resources

In this approach, the resources which are opened as a part of try block will be closed automatically once the control reaches to the end of try block normally or abnormally, so it is not required to close explicitly so the complexity of the program would be reduced.

If it is not required to write finally block explicitly, so, length of the code would be reduced and readability be improved.

```
try (BufferedReader br = new BufferedReader(new FileReader("abc.txt"))){  
    //use br and perform the necessary operation.  
}
```

Once the control reaches the end of try automatically br will be closed.

```
} catch (IOException e){}
```

II. Handling Code

Rules of using try with resource

- ① we can declare any no of resources, but all these resources should be separated with `try (R1, R2, R3;){}` to lose the resources.
- ② All resources are said to be AutoCloseable if the class implements an interface called "java.lang.AutoCloseable" either directly or indirectly or in package classes, java.sql package or `java.io` package.
- public interface java.lang.AutoCloseable {
 public abstract void close() throws java.lang.Exception;
- note : which ever class has implemented this interface those classes objects are referred as "resources".
- ④ All resources reference by default are treated as implicitly `final` and hence we can't perform `Decomposition` with `try` block.

```
try (BufferedReader br = new BufferedReader(new File("abc.txt"))){  
    br = new BufferedReader(new File("abc.txt"))  
    writer("abc.txt"))  
    but put : if : can't reassign a value
```

4) Until 1.6 Version try Should Compulsorily be followed by either catch or finally, but from 1.7 Version we can make only take try with resources without catch or finally

```
try (R) {  
    } //Valid
```

5) Advantage of try with resources Concept is finally block will become dummy because we cannot required to close resources Explicitly

6) try with resource nesting is also possible.

```
try (R1) {  
    try (R2) {  
        try (R3) {  
        }  
    }  
}
```

Multi catch block

With `try...catch` 1.6 even though we have multiple exception handling same handling code we have to write a separate `catch` block for every exception, it increases the length of the code and reduces readability.

logic

33

Fetch (Arithmetical Exceptions only)

one print Stark (1966:1)

9 Catch (null pointer exception, the) C

the print stack. Then click 'OK'.

To overcome this problem Sun has introduced "multi catch block" concept in 1.7 Version.

110

10

```
catch (AuthentificationException | NullpointerException e) {  
    e.printStackTrace();  
}
```

In multiple catch, there should not be any relation
b/w exception types (either child to parent or parent
to child or same type) it would result in compile time
error.

e.g.: try {

```
    } catch (ArithmaticException | Exception e) {  
    e.printStackTrace();
```

Output: Compile Time Error

throw \Rightarrow handle the exception using catch block and
throw \Rightarrow handle the exception object to the caller
throw \Rightarrow it batch the exception object to the caller
throw \Rightarrow method signature and commonly used if
the exception is "Checked Exception".

Checked Exception \Rightarrow compiler will check for the handling
code only then compilation is successful.

e.g.: IOException, SQLException, ... are all
checked exceptions.

Unchecked Exception \Rightarrow compiler will not check for
the handling code, but JVM come into picture
and possibility of "Successful" or "abnormal" termination.

e.g.: Runtime exception and its child classes: error
and its child classes are all "Unchecked Exception".

Rules of overriding when exception is involved

While overriding in the child class method throw any checked exception

Compulsory the parent class method should throw the same checked exception as its parent otherwise we will get compile time error. There will be no restrictions on unchecked exception.

Ex:

Class parent

```
public void methodOne()
```

class child method extends parent {
 public void methodOne() throws exception {

error in methodOne() in child. Cannot override method
One() in parent public void methodOne() throws
exception. & overridden method does not throw
exception.

Rules w.r.t Overriding

parent : public void methodOne() throws exception { }
child : public void methodOne()

Output : Valid

parent : public void methodOne() { }

child : public void methodOne() throws exception { }

Output : Invalid

parent : public void methodOne () throws Exception()
child : public void methodOne () throws Exception()
Output : Valid.

parent : public void methodOne () throws IOException
child : public void methodOne () throws IOException()
Output : Valid

parent : public void methodOne () throws IOException()
child : public void methodOne () throws FileNotFoundException
Exception, IOException()
Output : Valid

instanceof

1. we can use the instanceof operator to check whether the given an object is particular type or not

↳ instance of X

↳ reference

↳ Class/Interface Name.

eg : ArrayList al = new ArrayList(); //inbuilt object
where we can keep any type of other objects

al.add (new Student()); //0th position.

al.add (new Cricket()); //1st position

al.add (new Customer()); //2nd position

Object o = l.get (0); //l is an ArrayList object
if (o instanceof Student) {

Student s = (Student)o;

else if (Customer) {
Customer c = (Customer) o;

will Thread t = new Thread();

System.out.println (t instanceof Thread); //true

System.out.println (t instanceof Object); //true

System.out.println (t instanceof Runnable); //true

System.out.println (t instanceof Comparable); //true

Ex - public class Thread extends Object implements Runnable {

?

→ To use instanceof operator compulsorily there should be some relation between argument types either child to parent or parent to child or same type. Otherwise we will

eg - String s = new String ("sackin");

System.out.println (s instanceof Thread); //CE

Thread t = new Thread();

System.out.println (t instanceof String); //CE

→ whenever we are checking the parent object is child type or not by using instanceof operator that we get false.

Object o = new Object();

System.out.println (o instanceof String); //false.

Object o = new String ("arshak");

System.out.println (o instanceof String); //true
→ for any class or interface x null instanceof x
is always returns false

System.out.println (null instanceof X); //false

```
public class Test {  
    public static void main (String [] args) {  
        Object t = new Thread ();  
        System.out.println (t instanceof Object); //true  
        System.out.println (t instanceof Thread); //true  
        System.out.println (t instanceof Runnable); //true  
        System.out.println (t instanceof String); //false  
    }  
}
```

Is instanceof ()

Difference b/w instanceof and instanceof ()

instanceof

instanceof: an operator which can be used to check whether the given object is particular type or not.
If we know at the type at beginning if it is available.

e.g: String s = new String ("Gagan");

System.out.println (s instanceof Object); //true
If we know the type at the beginning only.

instanceOf()

isinstanceof() is a method present in class class. We can use isInstanceof() method to check whether the given object is particular type or not we don't know at the type at beginning & it is available Dynamically at runtime.

Class Test

```
public static void main(String[] args) {  
    Test t = new Test();
```

System.out.println(class.getName(args[0])). Instance.
(b) it's args[0] -- we don't know the type at
beginning.

java Test Test // true

java Test String // false

java Test Object // true

Computer Math Threading

Task → Activity / Piece of work

→ Syllabus

1. Introduction

1. The ways to define, instantiate and start a new Thread

Ex. By extending Thread class

Ex. By implementing Runnable interface

2. Thread class Constructors

4. Thread priority.

5. Getting and setting name of a thread.

6. The methods to prevent (stop) Thread execution

1. yield()

2. join()

3. sleep()

7. Synchronization

8. Inter Thread Communication

9. Dead lock

10. Daemon Threads

11. Various Conclusion

1. To Stop a Thread

2. Suspend & resume of a thread.

3. Thread group

4. Current thread

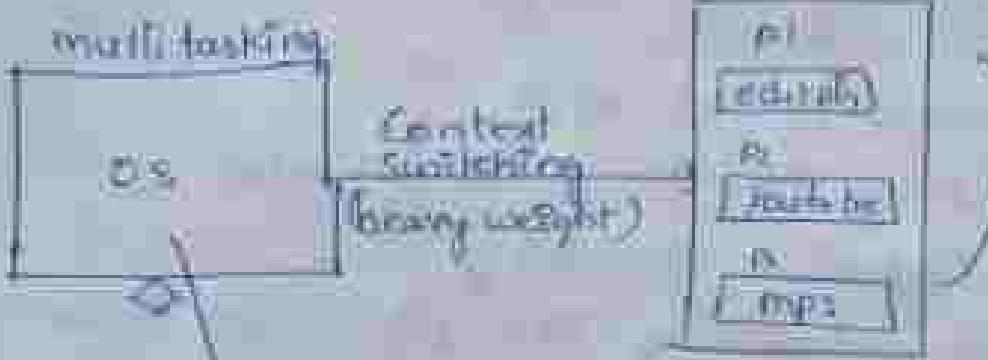
5. Thread local

13. Life Cycle of a Thread

Task \rightarrow Activity

process based

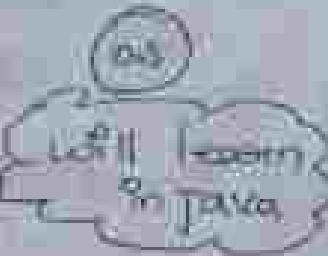
OS API (main, run, stop)



Operating System has
made multiple tasks
to run on the computer

multitasking = two types

- i) process based multitasking
- ii) Thread Based multitasking



Task \rightarrow Multiplexing is very easy.

Java API
for only by doing

Java

API (Thread, Runnable, ThreadLocal
packages (synchronized)
class fiber

multitasking

Executing several task simultaneously is the concept of multitasking.

There are 2 types of Multiple tasking.

- i) process based multitasking
- ii) thread based multitasking

process based multitasking

Executing several tasks simultaneously where each task is a separate independent process. Such type of multitasking is called

"process based multitasking"

e.g: typing a java program in editor

Listening to a song

downloading the file from Internet.

downloaded at OS level

process based multitasking is best suited at

Thread based multitasking

→ Executing several tasks simultaneously where each task is a separate independent part of the same program. It is called "Thread based multitasking".

Each independent part is called "Thread".

Each independent part is best suited at

① This type of multitasking is best suited at "programmatic level". The main advantages of multitasking is to reduce the response time of the system and to improve the performance.

② The main & important application areas of multitasking are:

a. To implement multimedia graphics

b. to develop web application Server (will focus in JEE)

c. to develop video games.

d. to develop animations.

Java provides inbuilt support to work with threads through API called Thread, Runnable Thread Group, Thread Local, ...

To work with multithreading, Java developer will code only for 10% remaining 90% Java API will take care.

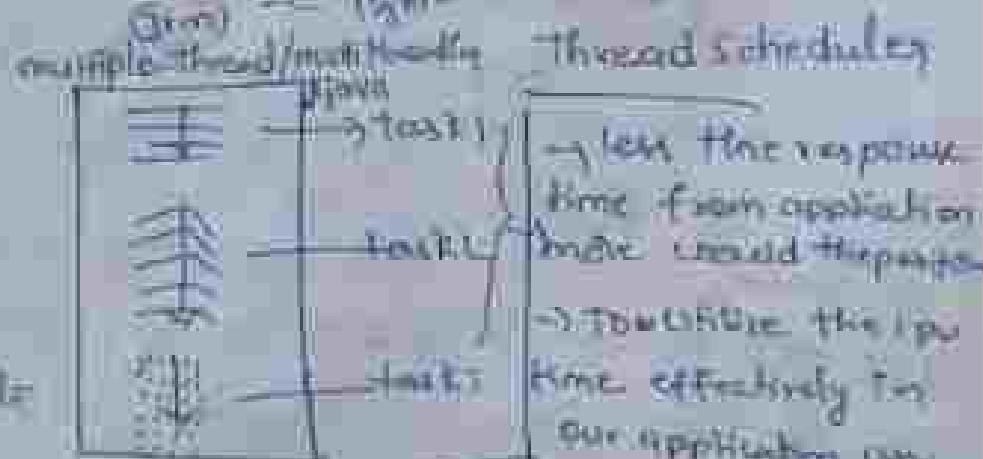
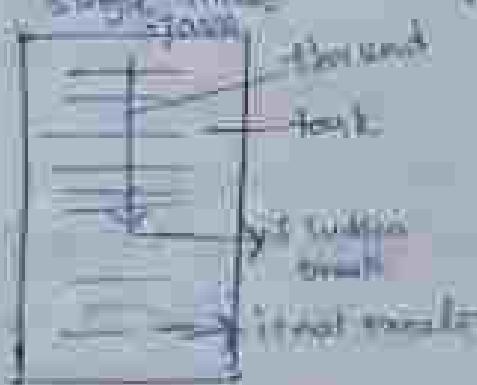
What is thread?

Separate flow of execution is called "thread". If there is only one flow then it is called "single thread". Programming for every thread there would be separate job.

In Java we can define a thread in two ways:

a. Implementing Runnable interface

b. Extending Thread class

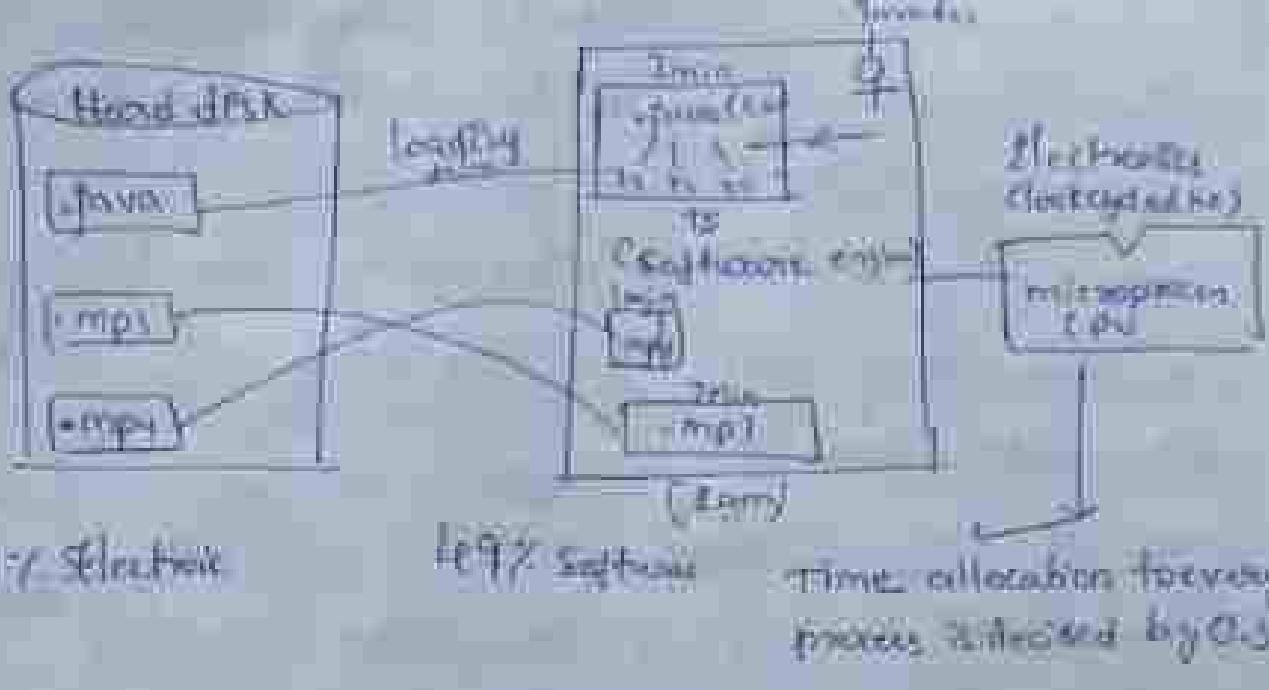


max. throughput time, performance of the application will increase.



Agenda of multithreading / multithreading

To use CPU time effectively, so that performance can be improved.



Extending Thread Class

→ We can create a Thread by extending a Thread

Class myThread extends Thread {

© 2010 Kuta Software LLC

public void sum()

```
for (int i=0; i<10; i++) {  
    S.o.p ("Hello Thread");  
}
```

defining a thread (writing a class and extending it)
 Thread Job a thread (create, run, join, etc.)

Class Thread Demo 4

```
public static void main(String ... args) {
    new Thread(() -> {
        System.out.println("Hello from thread 1");
    }).start();
    System.out.println("Hello from main thread");
}
```

Behind the scenes

- ① main thread is created automatically by JVM
② main thread creates child thread and starts the child thread

t.start()

Yield to JVM

System to System

Thread Scheduler

if multiple threads are waiting to execute then which thread will execute is decided by Thread Scheduler which is part of JVM. In case of multithreading we can't predict the exact output as possible output we can expect since jobs of threads are important because not interested in the timing performance should be improved.

Case 2 diff b/w (t.start()) and (t.run())

if we call t.start() and separate thread will be created which is responsible to execute t.run() method

if we call t.run(), no separate thread will be created rather the method will be called just like normal method by main thread

If we replace t.start() with t.run() then the output of the program would be

42 1



Case 3 :- (Importance of Thread class start() method)

For every thread, required mandatory activities like registering the thread with Thread Scheduler will be taken care by Thread class start() method and programmer is responsible of just doing the job of the thread inside run() method.

Start() acts like an interface to programmer.

public void start()

{ registration thread with ThreadScheduler }

 All other mandatory low level activities.

 Invoke or calling run() method.

we can conclude that without executing thread class start() method there is no chance of starting a new thread in Java.
 Note : This start() is considered as heart of multitasking.

Case 4) (If we are not overriding run() method)

If we are not overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won't get any output.

Q4: Class mythread extends Thread {

class ThreadDemo {

```
public static void main(String args) {  
    mythread t = new mythread();  
    t.start();  
}
```

It is highly recommended to override run() method, otherwise don't go for multithreading concept.

Caution

Q5: ~~Day 2~~

Case 5: Overloading of run() method
use can overload run() method, but Thread class start() will always call run() with zero argument. If we overload run method with arguments, then we need to explicitly call argument based run method and it will be executed just like normal method.

Q6:

```
class mythread Extends Thread {
```

```
    public void run() {
```

```
        System.out.println("no arg method");
```

```
    public void run(int i) {
```

```
        System.out.println("zero arg method");
```

class ThreadableDemo {

```
public static void main (String ... args) {  
    myThread t = new MyThread();  
    t.start();
```

Output: No arg method

Case 6: Overriding of start() method

if we override start() then our start() method will be executed just like normal method, but no new thread will be created and no new thread will be started.

Eg: class MyThread extends Thread {

```
public void run() {  
    System.out.println ("no arg method");
```

```
public void start() {
```

```
    System.out.println ("start arg method");
```

class ThreadDemo {

```
public static void main (String ... args) {  
    myThread t = new MyThread();  
    t.start();
```

Output: start arg method) Multithreaded only method
It is never recommended to override start() method.

Code 1

```
class MyThread extends Thread {  
    public void run() {  
        System.out.println("run method");  
    }  
    public void start() {  
        System.out.println("start method");  
    }  
}
```

Staple Thread Demo

```
public static void main (String [] args)  
{  
    myThread t = new myThread ();  
    t.start ();  
    System.out.println ("main method");  
}
```

Output (main thread) produced by only Main thread
main method } output -> Main thread
Start method }

2021

```
class myThread extends Thread {  
    public void start() {  
        super.start();  
        System.out.println("start method");  
    }  
    public void run() {  
        System.out.println("run method");  
    }  
}
```

Class Thread Demo

```
public static void main (String ... args) {
    myThread t = new myThread();
    t.start();
    System.out.println ("main method");
}
```

~~output~~ ~~thread~~
~~main method~~

→ a. Main method ~~output~~

→ b. start method ~~output~~

User defined thread

child thread will

→ a. run method ~~output~~ child run() method

Case 1: Life Cycle of a Thread

```
myThread t = new MyThread(); //Thread is in New State
t.start(); //Thread is in ready/Runnable State
```

→ If thread scheduler allocates CPU time then we say
thread entered into Running State

→ If run() is completed by thread then we say
thread entered into Dead State

→ Once we created a thread object then the
thread is said to be in new state or born
state

→ Once we call start() method then the
Thread is said to be in new state or born state

- Once we call start() method then the thread will be entered into ready to runnable state.
- If Thread Scheduler allocated Cpu then the thread will be entered into running state.
- Once run() method completes then the thread will entered into dead state.

Case 9: After starting the thread, we are not supposed to start the same thread again, then we say Thread is in "IllegalThreadStateException". We say Thread 1 is in myThread1. Thread 2 is in myThread2. Thread 1 is in begin thread state.



t.start();

t.start(); // IllegalThreadStateException
creation of thread using Runnable Interface

- Creating a thread using `java.lang.Thread` class
 - use `start()` from `Thread` class
 - override `run()` and define the job of the thread
- Creating a thread requirement to `implements Runnable` Interface

`void run();` it contains only `run` method.

class Thread implements Runnable { //Adapter class

 public void start()

1. Register the thread with Thread Scheduler
2. All other mandatory low level activities (memory level)
3. Invoke on call run() method

 public void run()

 //Job for a thread

}

}

Shortcuts of eclipse

Ctrl + Shift + T \Rightarrow To open a definition of any class

Ctrl + o \Rightarrow To list all the methods of the class

Note :- public void run() Thread ()

\Rightarrow Thread class start(), followed by Thread class

run()

public java.lang.Thread (java.lang.Runnable);

\Rightarrow Thread class start(), followed by

Class of Runnable run()

Defining a thread by implementing Runnable Interface

```
public interface Runnable {  
    public abstract void run();  
}  
public class Thread implements Runnable {  
    public void start() {  
        ① register Thread with thread scheduler  
        ② All other mandatory low level activities  
        ③ invoke run()  
    }  
    public void run() {  
        // empty implementation  
    }  
}
```

```
class MyRunnable implements Runnable {  
    @Override  
    public void run() {  
        for (int i=1; i<=10; i++)  
            System.out.println("child thread");  
    }  
}
```

public class ThreadDemo

```
public static void main (String [] args) {
    myRunnable r = new MyRunnable ();
    Thread t = new Thread (r); // Call MyRunnable run()
    t.start ();
```

for (int i=1; i<=10; i++)

```
System.out.println ("Main Thread");
```

}

Output:

Main Thread

a. main thread

....

child thread

a. child thread

....

....

Case Study

```
myRunnable r = new MyRunnable ();
```

```
Thread t1 = new Thread (r);
```

```
Thread t2 = new Thread (r);
```

Case 1: t1.start () . A new thread will be created which is responsible for executing Thread class run () - empty

~~multiple threads~~

Output main thread
main thread }

Case 2 : `ta.start()` A new thread will be created which is responsible for executing my Runnable `run()`

Output (main thread) , main thread
main thread
main thread
main thread
main thread

User defined thread :- child thread
child thread
child thread
child thread
child thread

Case 3 : `ti.run()` No new thread will be created but Thread Class `run()` will be executed just like normal method call

Output : main thread
main thread
main thread
main thread
main thread

Case 4: `t2.run()` -> No new thread will be created, but myRunnable class `run()` will be executed just like usual method call.

Output main-thread \rightarrow Child thread main method
Child thread main-thread
Child thread main-thread
Child thread main-thread
Child thread main-thread

Case 5: `t1.start()` It results in compile error
 \rightarrow Cannot symbol method starts myRunnable

Case 6: `t1.run()` No new thread will be created, but MyRunnable class `run()` will be executed just like usual method call.

Output Main-thread Child thread main-thread
Child thread main-thread
Child thread main-thread
Child thread main-thread
Child thread main-thread

(my Runnable \rightarrow new MyRunnable())

thread t1 = new Thread()

thread t1 = new Thread()

Case 1: `t1.start()`

Case 2: `t2.start()`

Case 3: `t3.start()`

Case 4: `t4.start()`

Case 5: `t5.start()`

Case 6: `t6.start()`

In which of the above cases a new thread will be created which is responsible for the execution of my Runnable `run()` method?

`t2.start()`

In which of the above cases a new thread will be created?

`t1.start()`

`t3.start()`

In which of the above cases MyRunnable class `func()` will be executed?

`t2.start()`

`t2.run()`

`t2.run()`

Different approach for creating a thread will be created?

~~`T1.start()`~~

~~`t1.start()`~~

- a. Extending Thread class.
- b. Implementing Runnable Interfaces

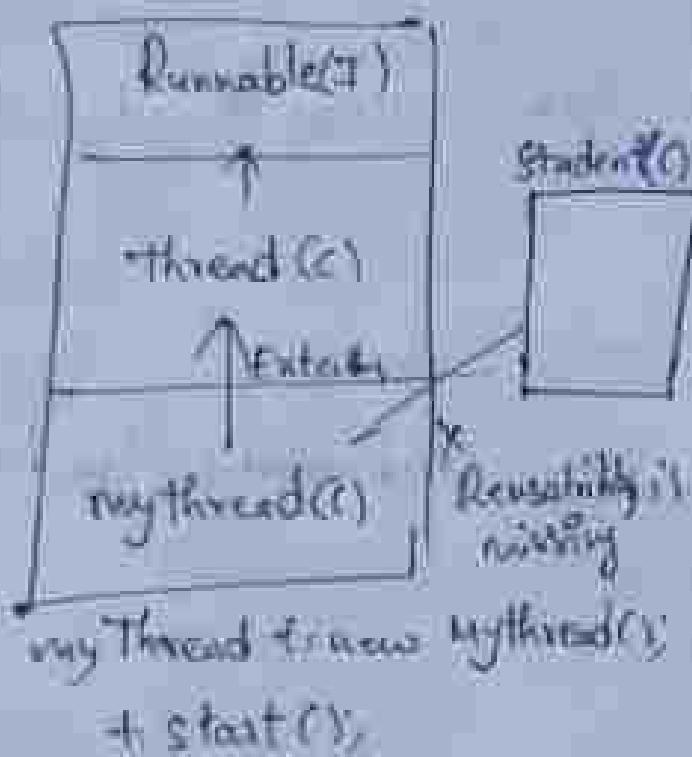
Life cycle of Thread

myThread t = new MyThread()



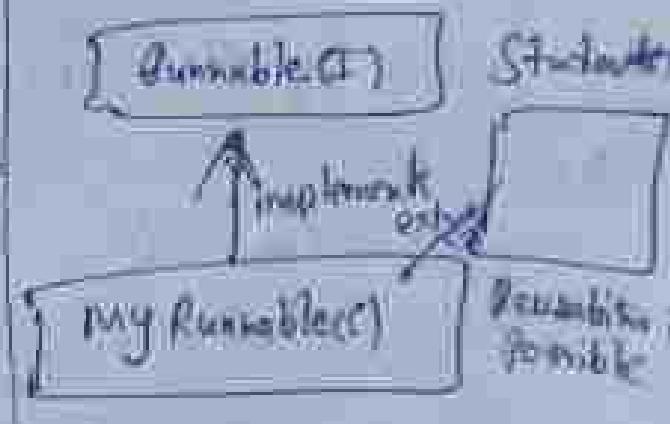
not SuggesTable

1st Approach: Extending Thread class



good approach

2nd Approach: Implementing Runnable



myRunnable r = new MyRunnable()
Thread t = new Thread(r)
t.start()

Which approach is good in above?

- a. Implementing Runnable Interface is recommended because one class can extend other class through which interface benefit can be gained into own class. Internally performance and memory level is also good when we work with interface.

- If we work with extends technique then we will miss out inheritance benefit bcoz already our class has inherited the feature of our "Thread class".
- So we usually don't prefer extends approach rather implements approach is used in real-time for working with "multi threading".

Various Constructors available in Thread class

- a. Thread t = new Thread(); daily usage ✓
- b. Thread t = new Thread(Runnable r);
- c. Thread t = new Thread(B string name);
- d. Thread t = new Thread(Runnable r, String name);
- e. Thread t = new Thread(ThreadGroup g, String name);
- f. Thread t = new Thread(ThreadGroup g, Runnable r, String name);
- g. Thread t = new Thread(ThreadGroup g, Runnable r, String name, long stackSize);
- h. Thread t = new Thread(ThreadGroup g, Runnable r);

All feasible approach to define a thread (not recommended)

class myThread extends Thread {
 public void run() {
 System.out.println("child thread");
 }
} class ThreadDemo {
 public static void main(String args) {

```
myThread t = new myThread();
Thread t1 = new myThread();
t1.start();
```

```
System.out.println("main thread");
```

(not ~~but~~ recommended)

Output: 2 threads are created

main thread

main thread

child thread

child thread

Internally related

Runnable



Thread



myThread

Name of the thread

Internally for every thread, there would be a name for the thread

a. name given by JVM

b. name given by the user

eg: class myThread extends Thread {

}

public class TestApp {

public static void main (String ... args) {

```
System.out.println(Thread.currentThread().getName()); //main
myThread = new MyThread();
t.start();
System.out.println(t.getName()); //Thread-0
Thread.currentThread().setName("Yash"); //yash
System.out.println(Thread.currentThread().getName()); //yash
System.out.println();
// Exception in thread "Yash" java.lang.Arith-
// exception by myZero
TestApp.main()
```

- It is also possible to change the name of the thread using setName(). Thread.currentThread().setName()
 - It is possible to get the name of the thread using getName(). Thread.currentThread().getName()

Methods :-

```
public final String getName();
public final void setName (String name);
```

```
public void run() {
    System.out.println("main() executed by
                        thread " + Thread
                        .currentThread().getName());
}
```

public class TestApp

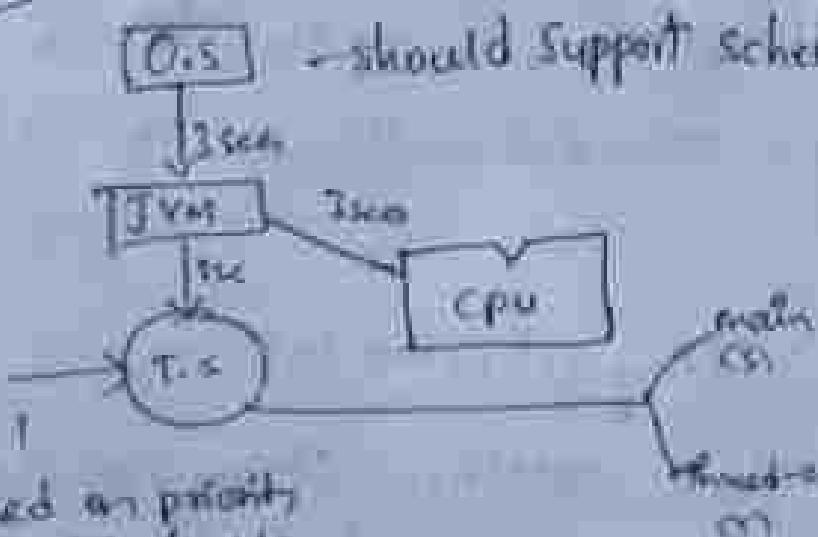
```
public static void main(String args) {
    MyThread t = new MyThread();
    t.start();
}
```

System.out.println("main() executed by thread")

MyThread.currentThread().getName()

Output: run() executed by thread > thread-0
main() executed by thread > main

Ranking priority



JVM will create
main thread & it
only start with a
default priority

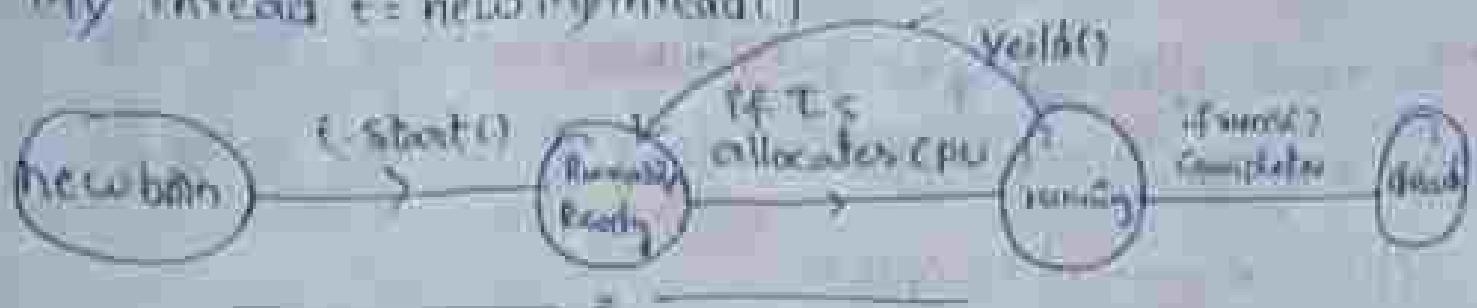
Based on priority
it will decide

If both threads have same priority then it's an algorithm which
is vendor dependent.

Life Cycle of Thread

①

My Thread t = new MyThread()



Thread Priorities (priorities)

- For every Thread in Java has some priority
- Valid range of priority is 1 to 10, it is not 0 to 10.
- if we try to give a different value then it would result in "IllegalArgumentException".

Thread.MIN_PRIORITY = 1

Thread.NORM_PRIORITY = 5

Thread.MAX_PRIORITY = 10

Thread class does not have priorities is Thread.
low_priority, Thread.high_priority

Thread scheduler allocates CPU time based on "priority".
if both threads have the same priority then
which thread will get a chance as a pjm
we can't predict becoz it is vendor dependent.
we can set and get priority values of the thread
using the following methods.

- a. public final void setPriority(int priorityNumber)
 - b. public final int getPriority()
- The following priorityNumber is from 1 to 10, if we try to give other values it would result in "IllegalArgumentexception"

```
System.out.println(Thread.currentThread().getPriority());  
& IllegalArgumentexception
```

Default Priority

The default priority for only main thread is 5, whereas as for other threads priority will be inherited from parent to child

parent Thread priority will be given as child thread priority

myThread +
newMyThread

Example class MyThread extends Thread {

```
    public class TestApp {
```

```
        public static void main (String ... args) {
```

```
            System.out.println(Thread.currentThread().getPriority());
```

```
            Thread.currentThread().setPriority(10);
```

```
            myThread += new MyThread();
```

```
            System.out.println(Thread.currentThread().getPriority()); //7
```

Reference

Thread



myThread

myThread is created by "mainThread", So
Priority of "mainThread" will be shared
as a priority for "myThread".

Code

class MyThread extends Thread {

@Override

public void run() {

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

System.out.println("Child thread");

}

}

public class TestApp {

public static void main (String ... args) {

myThread t = new MyThread();

t.setPriority (1); //line 1

t.start();

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

System.out.println("main thread");

}

Since priority of child thread is more than main thread, JVM will execute child thread first whereas for the parent thread priority is 5 so it will get last chance.

If we comment line-1, then we can't predict the order of execution b/w both the threads have same priority.

Some platforms don't provide proper support for thread priorities.

e.g. Windows 7, Windows 10, ...

we can prevent threads from execution.

- a. `Yield()` → refer  example life cycle
- b. `join()`
- c. `sleep()`

`yield()` → It creates a pause current executing thread for giving chance for waiting threads of same priority.

* If there is no waiting threads have low priority then same thread can continue its execution.

* If all the threads have same priority and if they are waiting then which thread will get chance we can't except it depends on thread scheduler.

The thread which is yield, when it will get the chance once again depends on the memory on thread scheduler and we can't expect exactly public static native void yield()

myThread t = new MyThread() //new state or born state
t.start() //enter into ready state/runnable state
if thread scheduler allocates process then enters into running state

- a. if running thread calls yield then it enters into runnable state
- if run() is finished with execution then it comes into ready state

Q11. Class MyThread extends Thread {

@ override

public void run() {

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

System.out.println("child thread");

Thread.yield(); //line 1

public class TestApp {

public static void main(String ... args) {

MyThread t = new MyThread();

t.start();

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

System.out.println("parent thread");

Note: if we comment line-1, then we can't expect the output before both the threads have same priority, then which thread the Thread Scheduler will schedule is not in the hands of programmer but if we don't comment line-1, then there is a possibility of main thread getting no more no of times, so main thread execution is faster than child thread will get chance.

Note: Some platforms won't provide proper support for yield(), because it is getting the execution code from other language preferably from C.

(b) join()

If the thread has to wait until the other thread finishes its execution then we need to go for join(), if t_1 executes to join() then t_1 should wait till t_2 finishes its execution, t_1 will be entered into waiting state until t_2 completes, once t_2 completes then t_1 can continue with its execution.

eg:-

Venue fixing \rightarrow $t_1.start()$
Wedding card printing \rightarrow $t_2.start()$ \rightarrow $t_1.join()$
Wedding card distribution \rightarrow $t_2.start()$ \rightarrow $t_2.join()$

Prototype of join()

public final void join() throws InterruptedException
public final void join (long ms) throws InterruptedException
public final void join (long ms, int ns) throws InterruptedException

Note: While one thread is in waiting state and if one more thread interrupt, then it would result in "InterruptedException". InterruptedException is checked exception which should always be handled.

Thread t = new Thread(); // new / blank state
t.start(); // ready / runnable state.

- If T.s allocates CPU time then thread enters into running state.
- If currently executing thread invokes t.join() / t.join(1000), t.join(1000, 100), then it would enter into waiting state.
- If the thread finishes the execution / time expires / interrupted then it would come back to ready state / runnable state.
- If run() is completed then it would enter into dead state.

MyThread class MyThread extends Thread {

 @Override

 public void run() {

 for (int i = 1; i < 10; i++) {

```

System.out.println("Sita Thread");
try {
    Thread.sleep(2000);
}
catch (InterruptedException e) {
}
}

public class Test {
    public static void main(String... args) throws
        InterruptedException {
        MyThread t = new MyThread();
        t.start();
        t.join(10000); // line -n1
        for (int i = 1; i <= 10; i++) {
            System.out.println("Rama Thread");
        }
    }
}

```

- If line -n1 is commented then we can't predict the output bcoz it is the duty of the T.S to assign C.P.U time
- ⇒ If line -n1 is not commented, then Rama Thread (main thread) will enter into waiting state till Sita Thread (child thread) finishes its execution.

Output

3. Threads

a. Child thread

 Sita thread

 sita thread

b. Main Thread

 Rama thread

 rama thread

Waiting of Child Thread Until Completing main Thread

we can make main thread to wait for child thread as well as we can make child thread also to wait for main thread

Ex:-

class mythread extends Thread

 Static Thread mt;

 @ override

 public void run()

 {

 mt.join();

 }

 Catch (InterruptedException e) {

 For (int i=0; i<10; i++) {

 System.out.println ("Child Thread");

```
public class Test3 {
    public static void main (String ... args) {
        myThread mt = Thread.currentThread();
        myThread t = new myThread();
        t.start();
        for (int i=1; i<=10; i++) {
            System.out.println ("main thread");
            Thread.sleep (2000);
        }
    }
}
```

Output

2 threads (main thread, child thread)

main thread

a.main thread

child thread

a.child thread

Ex2

```
class myThread extends Thread {
    static Thread mt;
```

@Override

```
public void run ()
```

try {

mt.join();

```
        catch (InterruptedException e) {  
    }  
    for (int i = 1; i <= 10; i++) {  
        System.out.println("child thread");  
    }  
}
```

```
public class Test3 {
```

```
    public static void main (String args) throws InterruptedException {  
        myThread t = new myThread();  
    }
```

```
    t.start();
```

```
    t.join();
```

```
    t.join();
```

```
    for (int i = 1; i <= 10; i++) {
```

```
        System.out.println("main thread");
```

```
        Thread.sleep (2000); // to see sleep
```

```
}
```

Output:

2 threads (main, child thread)

main thread

child thread

Note: if both the threads invoke `t.join()`, `mt.join()`
then the program would result in "deadlock".

200

public class Test3

```
public class Test3 {  
    public static void main (String... args) throws  
        TestException {
```

Third current thread (1) join (2)

10

→ Output: Deadlock, before main thread is waiting for the main thread itself

sleep()
if a thread don't want to perform any operations for a particular amount of time then we should go for sleep()

Signature

gnatible state native void sleep (long ms) through TE
public

public stake void sleep (long ms int ms) throws InterruptedException

every sleep method throws InterruptedException, which is a CheckedException so we should compulsorily handle the exception using try catch or by throws keyword otherwise it would result in compile time error

→ instead of new Thread (V) → new 6) ban state

4. Start in // ready/runnable state

- If TS allocates CPU time when it would enter into running state
- If `join()` completes, then it would enter into dead state
- If running thread invokes `sleep(1000)`/`sleep(1000,100)` then it would enter into sleeping state
- If time expires / if sleeping thread got interrupted then thread would come back to ready/runnable state

```

Code public class SlideRotator {
    public static void main (String args) {
        for (int i=1; i<=10; i++) {
            System.out.println ("Slide: " + i);
            Thread.sleep (1000);
        }
    }
}

```

Output:

Slide: 1

Slide: 2

Slide: 3

Slide: 4

Slide: 5

Slide: 6

Slide: 7

Slide: 8

Slide: 9

Slide: 10

11

Threads

Interrupting a Thread

depends on situation

public void interrupt()

→ if thread is in sleeping state ^(or) in waiting state we can interrupt a thread

class myThread extends Thread {

@Override

public void run()

try {

for (int i=1; i<=10; i++) {

System.out.println("I am busy thread");

Thread.sleep(2000);

catch (InterruptedException e) {

System.out.println("I got interrupted");

}

public class Test3 {

public static void main(String args) throws

InterruptedException {

myThread t=new myThread();

t.start();

t.interrupt(); //Fire - w

System.out.println("End of main()");

Scenario : If a comment line - hi

2 Thread

a. main thread

End of main ...

b. Child thread

Tom Lazy thread

...
...
if t.interrupt() then

Scenario :

Child thread

a. main thread

main thread

b. Child thread

Tom Lazy thread

Target Interrupted

ex 2 class MyThread extends Thread

Override

public void run()

for (int i=1; i<=10000; i++) {

System.out.println("Lazy thread : " + i);

System.out.println("Tom entering into sleeping state")

Thread.sleep(2000);

catch (InterruptedException ie) {
ie.printStackTrace();}

3

public class TestApp {

public static void main(String args) throws

myThread.sleep(1000); myThread();

fast();

if interrupt(); then

System.out.println("main thread");

7

line-11 is commented then no problem

line-11 is not commented then interrupt() will wait till
then this thread enters into waiting state/sleeping state

Note - If thread is interrupting another thread, but
target thread is not in waiting state/sleeping state
then there would be no exception.

then there would be no exception, till the target thread enters
interrupt() call to waiting state/sleeping state so that
then waiting on the target thread state so the

call won't be waited
once the target thread enters into waiting state/sleeping
state then interrupt() will interrupt and it leaves the
exception
interrupt() call will be waited only if the thread

interrupt() call will be waited only if the thread

does not enter into waiting state/sleeping state

yield(); join(); sleep();

1) purpose

a) yield()

TO pause current executing thread for giving the chance
of remaining waiting threads of same priority

b) join()

if a thread wants to wait until completing
some other thread then we should go for join()

c) sleep()

if a thread don't want to perform any operation
for a particular amount of time then we should go for sleep method.

② Is it static?
yield() yes
join() no
sleep() yes

④ Is it overloaded?
yield() no
join() yes
sleep() yes

③ Is it final?
yield() no
join() yes
sleep() no

⑤ Is it throws IEX?
yield() no
join() yes
sleep() yes

⑥ Is it native method?
yield() yes

join() no

sleep()
sleep(long no) \rightarrow native

sleep(long ms, int no) \rightarrow non-native

Note:- using Lambda Expression

Runnable x = () \rightarrow {

 for (int i = 1; i <= 5; i++)

 System.out.println("Child Thread");
 }

}

Thread t = new Thread(x);
t.start();

using anonymous inner class

new Thread() {
 Runnable t;

 t = outside

 public void run(){
 for (int i = 1; i <= 5; i++)

 System.out.println("Child Thread");

 };
 start();

Synchronization

- ① Synchronized is a keyword applicable only for methods and blocks.
- ② If we declare a method / block as synchronized then at a time only one thread can execute that method / block on that object.
- ③ The main advantage of synchronized keyword is we can resolve data inconsistency problems.
- ④ But the main disadvantage of synchronized keyword is it increases waiting time of the thread and affects performance of the system.
- ⑤ Hence if there is no specific requirement then never recommended to use synchronized keyword.
- ⑥ Internally synchronization concept is implemented by using lock concept.

Class X {

 Synchronized void m1() { }

 Synchronized void m2() { }

 void m3() { }

Key points

- ① If a thread invokes m1() then on the object lock will be applied.
- ② If a thread invokes m2() then m1() can't be called because lock of X object is with m1.
- ③ If a thread invokes m3() then execution will happen because m3() is non synchronized.
- lock concept is applied at the object level not at

the method level.

7) Every object in Java has a unique lock whenever we are using synchronized keyword then only lock concept will come into the picture.

8) if a thread wants to execute any synchronized method on the given object then it has to get the lock of that object.

once a thread got the lock of that object then it is allowed to execute any synchronized method on the object. if the synchronized method execution completes then automatically thread releases lock.

9) while a thread executing any synchronized method the remaining threads are not allowed to execute any synchronized method on that object simultaneously but remaining threads are allowed to execute any non synchronized method simultaneously. [lock concept is implemented based on the object but not based on method].

Note: Every object will have a synchronized area and non-synchronized area).

Synchronized Area \Rightarrow write the code only to perform update, insert, delete

non-synchronized Area \Rightarrow write the code only to perform select operation

at class level

class level lock

• perform read operation

Synchronized bookTicket() {
 // perform update operations

class Display {

 public void wish (String name) {
 for (int i=1; i<10; i++)

 {

 System.out.print ("Good morning: ");

 try {

 Thread.sleep (2000);

 } catch (InterruptedException e) {

 System.out.println (name);

~~catch~~

class myThread extends Thread {

 Display d;

 String name;

 myThread (Display d, String name) {

 this.d=d;

 this.name=name;

 @Override

 public void run() {

 d.wish (name);

```
public class Test {
    public static void main (String ... args) {
        Display d = new Display ();
        MyThread t1 = new MyThread (d, "dhoni");
        MyThread t2 = new MyThread (d, "yuvraaj");
        t1.start ();
        t2.start ();
    }
}
```

Output :- As noticed below the output is irregular
becoz at a time one resource called with 1
2. threads are acting simultaneously

3. Threads =
a. main Thread
b. Child Thread - 1
c. Child Thread - 2

Good morning - Good morning :-

Q3 Class Display :-

```
public synchronized void wish (String name) {
    for (int i = 1; i < 10; i++)
```

```
    System.out.println ("Good morning " + name);
```

```
    Thread.sleep (2000);
```

```
    catch (InterruptedException e) {
```

```
        System.out.println ("some error");
```

```
class myThread extends Thread {  
    Display d;  
    String name;  
    myThread(Display d, String name){  
        this.d=d;  
        this.name=name;  
    }  
}
```

```
@Override  
public void run(){  
    d.write(name);  
}
```

```
public class Test3 {  
    public static void main(String args){  
        Display d=new Display();  
        myThread t1=new myThread(d, "dhan");  
        myThread t2=new myThread(d, "yuv");  
        t1.start();  
        t2.start();  
    }  
}
```

Output:

3 blocks

a. main Thread

b. child thread 1

God morning :dhan

God morning :yuv

C. Child Thread -2

God morning :yuv

God morning :yuv



Note: As noticed above there are 2 threads which are trying to operate on single object called "Display" we need Synchronization to resolve the problem of "Data consistency".

Conclusion:

Display d1 = new Display();

Display d2 = new Display();

MyThread t1 = new MyThread(d1, "yuvraj");

MyThread t2 = new MyThread(d2, "ishan");

t1.start();

t2.start();

In the above case we get irregular output, because there are two different object and same the method is applied with object and both synchronized lock is applied with object and both the threads will start simultaneously on different Java objects due to which the output is "irregular".

Conclusion:

If multiple threads are operating on multiple objects then there is no impact of synchronization.

If multiple threads are operating on same Java objects then synchronized keyword is required (applicable).

Class level lock

1. Every class in Java has a unique level lock.

- ④ If a thread wants to execute static synchronized method then the thread acquires class level lock.
- ⑤ While a thread executing any static synchronized method the remaining threads can not allow to execute any static synchronized method of that class simultaneously. ~~and normal instance method~~
- ⑥ But 5 remaining threads are allowed to execute normal synchronized methods or normal static methods, and normal instance methods simultaneously.
- ⑦ Class level lock and object lock both are different and there is no relationship b/w these two.

eg:- Class X {

 static synchronized void f() { // class level lock
 static synchronized void g() {
 static int i; // no lock required
 Synchronized void h() { // object level lock
 int c; // no lock required
 }

 ↳ m(1) → Class level lock applied and change in

 ↳ m(2) → no change in class level lock

 ↳ m(3) → gets chance for execution without any lock

 ↳ m(4) → Object level lock applied & chance is given

 ↳ m(5) → gets a chance for execution without any lock

Q31

Class Display &

```
public synchronized void displayNumbers() {  
    for (int i=1; i<=10; i++)
```

```
        System.out.print(i);
```

```
    try {  
        Thread.sleep(3000);  
    } catch (InterruptedException e) {  
    }
```

```
    }  
}
```

```
2) public synchronized void displayCharacters() {  
    for (int i=65; i<=90; i++)
```

```
        System.out.print((char)i);
```

```
    try {  
        Thread.sleep(3000);  
    } catch (InterruptedException e) {  
    }
```

```
    }  
}
```

```
3) class MyThread extends Thread {
```

```
    Display d;
```

```
    MyThread() {  
        d = new
```

```
        @Override
```

```
        public void run() {
```

3. display numbers()

```
class myThread2 extends Thread {  
    display d;  
    myThread (display d) {  
        this.d=d;  
    }  
}
```

```
@Override  
public void run() {  
    d.displayCharacters();  
}
```

```
public class Test {  
    public static void main (String args) {  
        display d1=new display();  
        myThread1 t1=new myThread1 (d1);  
        myThread2 t2=new myThread2 (d1);  
        t1.start();  
        t2.start();  
    }  
}
```

Output :

- Threads
 - a. main Thread
 - b. user defined Thread (displayCharacters())
 - c. user defined Thread
displayNumbers()

④ Synchronized block

```
Synchronized void m1() {  
    ...  
}
```

If few lines of code is required to get synchronized then it is not recommended to make method only as synchronized.

If we do this then for threads performance will be low. To solve this problem we use "Synchronized block", due to synchronized block performance will be improved.

Case Study

If a thread got a lock of current object, then it is allowed to execute that block.

a. Synchronized (Method) :-

```
class Test
{
    synchronized void m1()
    {
        // code
    }
}
```

To get a lock of particular object :-

b. Synchronized (Object) :-

```
class Test
{
    synchronized void m1(Test obj)
    {
        // code
    }
}
```

If a thread got a lock of particular object is, then it is allowed to execute that block.

c. To get class level lock we have to declare synchronized block as follows :-

```
Synchronized (Object.class) {
```

```
}
```

If a thread gets class level block, then it is allowed to execute that block.

Algo 12

Synchronized block

Ex: 1

Ques

class Display {

 public void wish (String name) {

 // 1-latch lines of code

 synchronized (this) {

 for (int i=1; i<=10; i++)

 System.out.println ("Good morning");

 try { Thread.sleep (5000); }

 catch (Exception e) {

 System.out.println (name);

 } // 1-latch lines of code

 }

class myThread extends Thread {

 Display d;

 String name;

 myThread (Display d, String name) {

 this.d = d;

 this.name = name; }

 public void run() {

 d.wish (name);

 }

class Test {

 public static void main (String [] args) {

 Display d = new Display ();

 myThread t1 = new myThread (d, "Rahul");

 myThread t2 = new myThread (d, "Yuvil");

1. start()
2. start()

TE = Interrupt Exception

class Display {
 public void run(String name) {
 // 1-line of code
 System.out.println("Good morning " + name);
 Thread.sleep(3000);
 System.out.println("Morning");
 // 1-line of code
 }
}

Class myThread extends Thread {
 Display d;
 String name;
 myThread(Display d, String name) {
 this.d = d;
 this.name = name;
 }
 public void run() {
 d.run(name);
 }
}

public class Test {
 public static void main(String[] args) {
 Display d = new Display();
 Display d1 = new Display();
 myThread t1 = new myThread(d, "Morning");
 myThread t2 = new myThread(d1, "Evening");
 t1.start();
 t2.start();
 }
}

Output = Morning
Morning
Evening
Evening
Two object and
two threads
act on two
different objects

Ques 2 Class Display &

```
public void with (String name) {
    System.out.println ("Good morning " + name);
    try {
        Thread.sleep (5000);
    } catch (InterruptedException e) {
        System.out.println (name);
    }
}
```

10 // 10th lines of code

Class myThread extends Thread &

```
Display d;
String name;
myThread (Display d, String name) {
    Thread = d;
    this.name = name;
}
public void run() {
    d.with (name);
}
public class Test {
    public static void main (String [] args) {
        Display d = new Display ();
        myThread t1 = new myThread (d, "Akash");
        myThread t2 = new myThread (d, "Yuvvi");
        t1.start ();
        t2.start ();
    }
}
```

Note : It object & thread, but the thread which gets a chance applied class level lock so output is regular.

class lock Context applicable only for objects on class level, but not for primitive types if we try to do it types, but not for primitive types if we try to do it would result in compile time error because "uncontrolled type" is not required.

Q: int x = 10;
Synchronized (x) {
 if (x == 10) {
 x = 20;
 }
}

Required answer is

Interthread Communication (Remember, postbox is example)

Two threads can communicate each other with the help of a notify()

- a. notify()
- b. notifyAll()
- c. wait()

notify() → Thread which is performing updation should call notify(), so the waiting thread will get notification so it will continue with its execution with the updated items.

wait() → Thread which is expecting notification/updation should call wait() immediately the thread will enter into waiting state.

If a thread wants to call wait(), notify() / notifyAll() then compulsory the thread should be the owner of the object otherwise it would result in illegal monitor self-destruction.

meaning thread should be owner of that object if thread has lock of that object.

It means these methods are part of Synchronized block or Synchronized method, if we try to use outside.

Synchronized area then it would result in `Runtime Exception` called "IllegalMonitorStateException".

If a thread calls `wait()` on any object, then first it immediately releases the lock on that object and returns into waiting state.

If other thread calls `notify()` on any object, then he may or may not release the lock on that object immediately. Except `wait()`, `notifyAll()` lock can't be released by other methods.

Note: `yield()`, `sleep()`, `join()` → will release the lock, otherwise `wait()`, `notify()`, `notifyAll()` → will release the lock, otherwise inter thread communication won't happen.

Once a thread calls `wait()`, `notify()`, `notifyAll()` methods on any object even it releases the lock of that particular object but not all locks it has.

Method prototype of `wait()`, `notify()`, `notifyAll()` method prototypre of `wait()`, `notify()`, `notifyAll()` methods on any object then it releases the lock of that particular object but

- ① public final void wait() throws InterruptedException
- ② public final void wait() throws InterruptedException
- ③ public final void wait(long timeout) throws InterruptedException
- ④ public final native void notify()
- ⑤ public final void notifyAll()

Interview Question

method like `wait()`, `notify()`, `notifyAll()` are present inside object class, yet in Thread class.

Interview question

method `Object.wait()`, `Object.notify()` & `Object.notifyAll()`

Thread will call `wait()`, `notify()`, `notifyAll()` on objects like postBox, stack, customer, student, ...

→ `obj.wait()`, `obj.notify()`, `obj.notifyAll()`

These methods should be available for every object. If the method has to be available for every object in Java then these methods should come in Object class.

Program

e.g. → class ThreadB extends Thread {
 int total = 0;

 @Override

```
    public void run() {
        for (int i = 0; i < 100; i++) {
            total = i;
        }
    }
```

public class Test {

```
    public static void main(String[] args) throws Exception {
        Thread b = new ThreadB();
        b.start();
        System.out.println(b.total);
    }
}
```

A. `start()`

If i replace with `Thread.sleep(1000)` then

Thread will enter into waiting statement but within 10 sec. if the condition is not ready then we should not use `Thread.sleep(1000)`

Q Ques 2 \rightarrow if we replace with `b.join()`, then main thread will enter into waiting state, then main thread will ~~join~~ and child thread will execute the loop, till then ~~join~~ is over and child thread will come to running state, main thread has to wait again thread to complete the operation result.

```
for (int i=0; i<100; i++)  
    total += i;
```

Now if we use `c.join()` in available main thread has to wait till last line of code (main thread should wait for the complete code to finish).

Soln: Class Thread extends Thread &
int total = 0;

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

```
    synchronized (this) {
```

```
        System.out.println("child thread started  
        calculation");
```

```
        for (int i=0; i<100; i++)  
            total += i;
```

```
    }  
    System.out.println("child thread completed  
    give notification");
```

```
}
```

```
public class Test {
```

```
    public static void main (String [] args) throws  
    "ThreadEx" {  
        Thread t = new Thread (
```

```
            b.start());
```

```
        Thread.sleep (10000); // 10 sec
```

```
        synchronized (b) {
```

```
            System.out.println ("main thread is calling wait  
            on object");
```

br.wait();

System.out.println("main thread got notification");

System.out.println("b.total:");

Output: Child thread started calculation

Child thread trying to give notification

main thread is called wait on B object hence child thread will never get notification

Code: class ThreadB extends Thread {
 int total = 0;

Override

public void run() {

Synchronized (this) {

System.out.println("Child thread started calculation");

for (int i=0; i<100; i++) {
 total+=i;

System.out.println("Child thread trying to give
notification"); // Step-1
 this.notify(); // Step-2

public class Test {

public static void main (String[] args) throws

InterruptedException {

b.start();

Synchronized(b) {

System.out.println("main thread is calling wait
on B object"); // Step-1

b.wait(10000); // 10sec

Synchronized(b) {
 System.out.println("main thread is getting notification"); // Step-2

outputs child thread started calculation

(child thread trying to give notification
main thread is still waiting on B object for 10sec
main thread got notification

2.2

producer consumer problem

producer → produce the item and update in the queue
(consumers → consume the item from the queue (wait))

class producer extends Thread {

 producer () {

 synchronized (q) {

 produce the item and update in the queue
 q.notify();

 }

 class consumer extends Thread {

 consumer () {

 synchronized (q) {

 if (q is empty) {

 q.wait();

 else {

 consume the item from the queue

 }

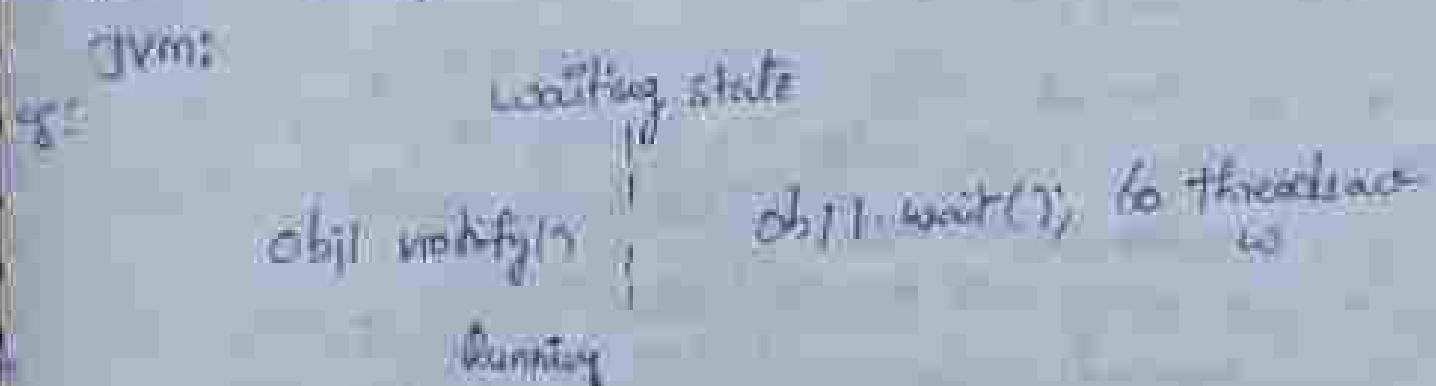
Difference b/w notify and notifyAll()

notify() → To give notification only for one waiting thread.

notifyAll() → To give notification for many waiting threads

Thread

→ we can use notify() method to give notification to only one thread. If multiple threads are waiting then only one thread will get the chance. If multiple threads are waiting then only one thread will get the chance and remaining threads have to wait for another notification. But which thread will be notified (random) we can't expect exactly it depends on JVM.

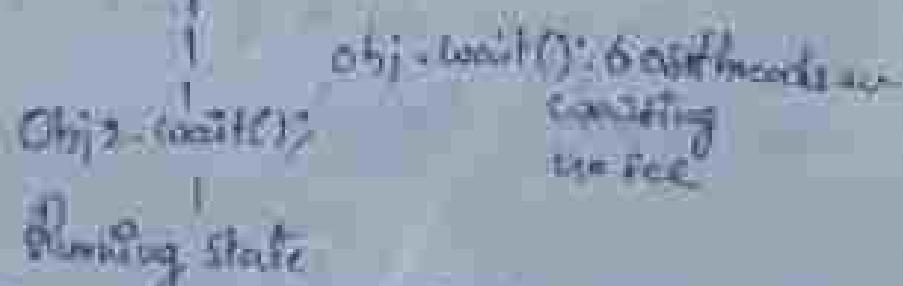


Among 6 threads which thread will get a chance we don't have control over that it is decided by JVM (threadscheduler).

⇒ we can use notifyAll() method to give the notification to all waiting threads of particular object.

All waiting threads will be notified and will be executed one by one, because they required lock.

obj.notifyAll() | waiting state



Note:- On which object we are calling wait(), notify() and notifyAll() methods that corresponding object lock we have to get but not other object locks.

Stack S1 = new Stack();

Stack S2 = new Stack();

Synchronized(S1) {
S2.wait(); // throws InterruptedException

Synchronized(S2) {
S2.wait(); // valid

Question based on lock

- Immediately it will enter
- ① If a thread calls wait() immediately it will enter waiting state without releasing any lock (false)
 - ② If a thread calls wait() it releases the lock that object but may not immediately (false).
 - ③ If a thread calls wait() on any object it releases all locks acquired by that thread and enters into waiting state (false).
 - ④ If a thread calls wait() on any object, it immediately releases the lock of that particular object and enters into waiting state (true).
 - ⑤ If a thread calls notify() on any object, it immediately releases the lock of that particular object (invalid).
 - ⑥ If a thread calls notify() on any object but may not releases the lock of that object but may not immediately (false).

Ques Dead lock
If 2 threads are waiting for each other forever (without end) such type of situation (irritive waiting) is called dead lock. There are resolution techniques for dead lock but several prioritization techniques are possible. Synchonized keyword is the cause for deadlock hence whenever Synchonized keyword we have to take special care we are using synchronized keyword.

Sol Class A

```
public void d1() {
    System.out.println("Thread-1 start execution of d1()");
    Thread.sleep(5000);
    catch (InterruptedException e) {
        System.out.println("Thread-1 trying to call to last()");
        e.printStackTrace();
    }
}
```

Class B

```
public void d2() {
    System.out.println("Thread-2 start execution of d2()");
    Thread.sleep(5000);
    catch (InterruptedException e) {
        System.out.println("Thread-2 trying to call to last()");
        e.printStackTrace();
    }
}
```

```
public void last() {
    System.out.println("Inside last() method");
}
```

public class Test extends Thread {

 synchronized {

 System.out.println("B()");

 public void run() {

 System.out.println("A()");

 } // run() is executed by main thread.

} // class Test

void main() {

 Test t = new Test();

 t.start();

 System.out.println("String() execute");

 t.join();

 t.interrupt();

 } // main() is not required to

 Since methods are not synchronized, lock is not required, so

 no dead lock.

 synchronized {

 System.out.println("A()");

 } // synchronized block

 synchronized {

 System.out.println("B()");

 } // synchronized block

 synchronized {

 System.out.println("C()");

 } // synchronized block

 } // class Test

 public synchronized void d() {

 System.out.println("D()");

 } // synchronized block

 } // class Test

 } // class Test

 } // class Test

 } // class Test

public synchronized void last() {

} System.out.println("Main thread last() method")

j }

Class B extends Thread {

public synchronized void do() {

} System.out.println("Thread-1 starts running")

try {

Thread.sleep(500); // 500

}

catch (InterruptedException e) {

} System.out.println("Thread-2 trying to call do()")

a last() {

public synchronized void last() {

} System.out.println("Thread-2 last() method")

}

public class Test extends Thread {

A = new A();

B = new B();

public void run() {

Thread.sleep(100); // line executed by main thread

a do();

public void run() {

b.do(); // line executed by child thread

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

Test t = new Test();

t.start(); // main thread is executing

j }

In the above program there is a possibility of Dead Lock.

Output

Thread-1 starts execution of do()

Thread-2 starts execution of do()

Thread-1 trying to call `b.fetch()`

Thread-2 trying to call `a.fetch()`

if there can be waiting

$t_1 \rightarrow$ Starts `dt()`, since `dt()` is Synchronized and a part of 'A' class so t_1 applies lock of (A) and starts the execution, while executing it encounters Thread `sleep()` so it gives chance for t_2 Thread-2 after getting a chance again by T_2 , it tries to execute b.fetch after getting a chance again by T_2 , it enters into waiting state but lock of b is with t_2 thread so it enters into waiting state but lock of a is with t_1 thread, so t_1 enters into waiting state but lock of a is with t_1 thread and it would be Since both the threads are in waiting state and it would be Note: Synchronized is the only reason why there is a deadlock so we should be careful when we use Synchronized keyword, if we remove atleast one Synchronized word then the program won't enter into deadlock

Deadlock vs Starvation

long waiting of a thread, where waiting never ends is termed as deadlock. long waiting of a thread, where waiting ends at certain point long waiting of a thread, where waiting never ends is called starvation.

eg: assume we have 100 threads, where all 100 threads have priority is 10, but one thread is there which has priority 1, now the thread with priority 1 has to wait for instance but still it is a

as above, but it has to wait for long time scenario called "starvation".

Note - Low priority thread has to wait until completing all priority threads, but ends at certain point which is nothing but starvation.

Daemon threads

The thread which is executing in the background is called "Daemon thread".

e.g. - Thread listeners, Signal Dispatcher, Garbage Collector...
mention the example of main.

1. producer
2. consumer
3. main thread
4. ...
5. ...
6. ...

Main objective of Daemon thread

The main objective of Daemon thread, to provide support for non-Daemon threads (main thread).

e.g. - if main thread runs with low priority then will call Garbage collector thread to destroy the useless objects, so, that no. of bytes of free memory will be improved with this free memory main thread can continue its execution. Usually Daemon threads having low priority, but based on our requirement daemon threads can run with high priority.

also

JVM \Rightarrow creates 2 threads

- a. Daemon Thread (priority = 1, priority = 10)
- b. main (priority = 5)

while executing the main code, if there is a leakage of memory then immediately JVM will change the priority of Daemon thread to 10, so Garbage collector activities Daemon thread will release the memory after defining it immediately it changes the priority to 1, so main thread it will continue.

How to check whether the thread is Daemon or not?

public boolean isDaemon() \Rightarrow To check whether the thread is "Daemon".

public void setDaemon(boolean b) throws IllegalThreadStateException
 \Rightarrow true, means the thread will become Daemon, before starting the thread we need to make the thread as "Daemon" otherwise it would result in "IllegalThreadStateException".

What is the default nature of the thread?

Ans: By default the main thread is "Non Daemon" & for all remaining thread Daemon nature is inherited from parent to child, that is if the parent thread is "Daemon" then child thread will become "Daemon" and if the parent thread is "Non Daemon" then automatically child thread is also "Non Daemon". Is it possible to change the Non-Daemon nature of main thread?

Ans: Not possible to change the Non-Daemon nature of main thread starting in not "main" hands, it will be started by "JVM".

Q: Class myThread extends Thread { }

public class Test { }

```
public static void main (String [] args) { }
```

```
System.out.println (Thread.currentThread()); // Daemon
```

```
Thread.currentThread().setDaemon (true); // the whole
```

myThread t = new myThread();

```
System.out.println (t); // Daemon (1); Not die
```

```
t.setDaemon (true);
```

```
t.start ();
```

```
System.out.println (t); // Daemon (1); Not die
```

Note: whenever last Non-Daemon threads terminates, automatically all Daemon threads will be terminated irrespective of their position.

e.g.: makeup man in shooting is a Daemon thread
here's main thread

if here code is over, then automatically the makeup code is also over automatically

e.g.: Class myThread extends Thread { }

```
public void run() { }
```

```
for (int i=1; i<=10; i++) { }
```

```
System.out.println ("child thread");
```

try {

```
Thread.sleep (2000); // 2 sec
```

catch (InterruptedException e) { }

```
System.out.println (e);
```

public class Test 3

```
public static void main (String[] args) {  
    MyThread t = new MyThread();  
    t.setDaemon (true); //stmt-1  
    t.start();  
    System.out.println ("end of main thread");
```

Output: if we comment stmt-1 then both the threads are non-Daemon threads it would continue with the execution end of main thread

Old thread

Old thread

...

Output: if we remove comment on stmt-1, then main thread is Non-Daemon thread whereas user-defined thread is Daemon thread, if the main thread finishes the execution then automatically the Daemon thread also will finish the execution.

Java

→ green Thread - only with jvm without taking os support

→ Native os model → taking support of os and jvm like

↳ Linux os threads very user friendly support

→ platform - start (if it can't work with user-defined code)

→ stop - if it can't work with user-defined code

↳ by using

→ Suspend and resume -

↳ os code
↳ 105

Collections in Java

Ques

Arrays/ArrayList / -

- To store data small data create Variable name and use one String variable
- if data is large value it is difficult to create so many Variables.
- So we have array (large data ~~is~~ ^{is} ~~inflexible~~ ^{flexible})
- Single Var Name
- Large Volume of data
- Indexed Data Structure
- ~~State~~ - only Simillar data / homogeneous data.
- Can array store mixed data / heterogeneous data (no) - Answer is no

disadvantage of array

- Size is fixed.
 - It demands Contiguous memory locations.
 - no Inbuilt class (no methods)
 - because of this array has limitations.
- vector
- Inbuilt class are legacy class
- not maintaining uniformed data structure.
 - Java happens to be Open Source Programming Language

→ Joshua - creating library of Get/Printers
class

↳

↓
Inbuilt methods

he gave name on collections → Sun's collections →
they collection in API

→ we have to understand and use it
don't need to code anything just use one
using

→ collection . all major 7 classes - 1
↳ (Collection) → (ArrayList) → (LinkedList) → (Queue) → (Stack) → (Set) → (Map)
① ArrayList
② linked list
③ Array Queue
④ Priority Queue
⑤ TreeSet
⑥ HashSet
⑦ linked-HashSet
map → important and legacy classes

① ArrayList → List(T) interface

Inbuilt in Collection API

ArrayList : Internally it follows dynamic array
data structure.

↑

Create

ArrayList ob = new ArrayList();

! it is class Inbuilt class & it is

- `odd(10)`
- `odd(20)`
- `odd(20)` ~~will~~ `(10, 10, 20)`
- `odd(10)`
 - what even ~~an odd~~ it will store in collection as an Object auto boxing
- if data heterogeneous it will store as Object and ~~homogeneous~~ also and
- ArrayList we can store heterogeneous data as Object
- Dynamic Array means dynamically size and will grow
 - 

(increasing growing)
- size will grow based on delta
- it is grows dynamically not fixed number
- array size fixed it will not grow
- it has guaranteed with boundaries
- here utility relax not restricted
- can use copy one collection into another or vice versa collection to another collection

we add front and last. Com able add to-front and middle and rear-end.

→ Size is increasing

→ we use only to add object in Array to
any given index.

Frontend



middle



we Com able

but



not able

rear-end



we Com able

but

not able

not able

not able

it is taking time because it is shifting data

to next index.

→ Linked List → (List of deque) - follows

doubly Linked List DS)

1. Homogeneous Data



2. Heterogeneous Data



3. Stored as object

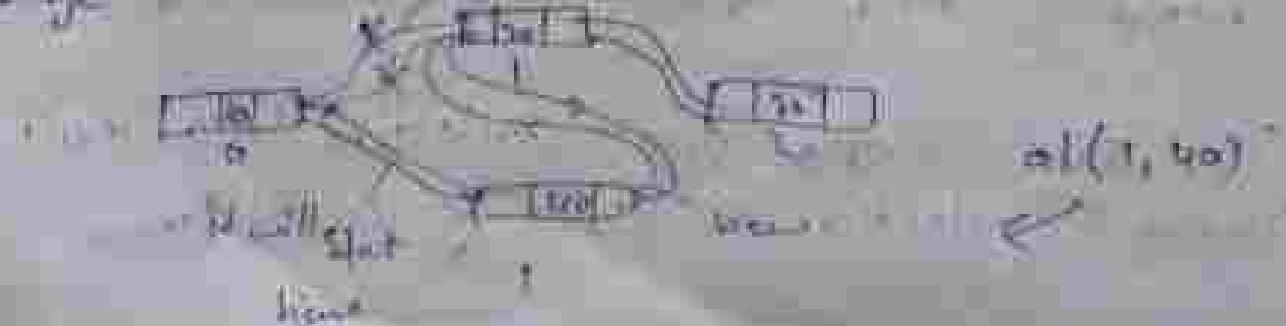
(nodes)

location :  before available

Right point to left link (vice versa)

it will link all the nodes and to each

→ here no shifting of index, only link will
change



→ Linked list (it is views & linked list (it))

Unlinked (it),

Unlinked ("Unlinked")

It.add (20);

System.out.println (it); → Output = [10, 10, 20]

It.addFirst ("Vijay")

System.out.println (it); → Output = [Vijay, 10, 10, 20]

It.add (3, "Bangalore")

It.addLast ("Richmond Town")

System.out.println (it); → [Vijay, 10, "Bangalore", 20, "Richmond Town"]

→ We have large volume data section

→ Use of ArrayList and Linked list

→ Homo and Heterogeneous data

→ every data as object. It will store

→ String based adding is allowed

→ In array String based adding possible but in linked list not possible

→ If we want to add first and last and

at given position Linked list is recommended

→ Linked list is more efficient than array list in data structures.

→ We can make use of dynamic memory allocation in linked list

→ After these does it's array concept gone?
outdated?

→ Answer is "No"

when we have storing large data.

→ Arrays → Collection
(ArrayList, LinkedList)

→ When to use array over ArrayList?

→ When ever size of the data is known

→ If you are sure that data is homogenous/similar

then you must go with array

→ array is faster than ArrayList

→ ArrayList → Collection → internally implemented
→ Object → It implements Comparable

→ primitive-object → new instance has weak

→ array → primitive data

→ Object also → It is fast

→ In the application when you know
homogenous and size is fixed.

→ In the application when you don't know
size and heterogeneous then go with
ArrayList.

→ At given position to perform insertion
→ use LinkedList.

ArrayList & Linked List

- duplicates are allowed in both
- like `l.add(10)` and `l.add(10)`
- when we are using `int` built method
- ④ That built method what type of data it is accepted
- ⑤ that built method give me what
- ⑥ when we any built method it will show what type of method
- ⑦ data `contains(11)` - It will give whether it is ArrayList contains
- boolean `res = arr.contains(11);`
`System.out.println(res);` //true
- ⑧ System.out.println - when we will use debugging purpose
- ⑨ capacity - how much you have
- ⑩ `get(2)` - index - size - how much you will
- ⑪ `isEmpty()` - check - is it empty
- ⑫ `ensureCapacity(10)` - size print same also
- ⑬ `tripleSize()` - increasing twice size
- ⑭ `getClass()` - entire quantity name we will get
- ⑮ `clear()` - it will clear data - size 0
- ⑯ Conclusion point -

`ArrayList<Integer> l1 = new ArrayList<Integer>();`

String

only `Integer` will store. not other type of data.

- ArrayList → lounge Data
- Both Heterogeneous
- Dynamically DS →
- Indexed Based Access
- Duplicates allowed
- Efficient Insertion at rear end
- List (T)
- Using Prebuilt methods → long run we perform

- LinkedList → doubly linked list
 - ↳ Nodes & links with previous node & next node
 - Can we perform Insertion at any given position
 - one Collection another Collection
 - store data as object
 - List → Deque → Duplicate allowed → Random access
 - Extra methods → LinkedList

Common methods in LinkedList $\{10, 20, 30, 40\}$

- ① Element(1)
- ② getFirst() → it will get first object
- ③ getLast() → it will get last object
- ④ getIndex(0) → it will get Index
- ⑤ getLastIndex() → it will get last index of consider

→ offerFirst() - it will add first to
offerLast() - it will add last to
offerFirst() & offerLast() similar working
offerFirst() > it should will get odd

→ peek() and poll() - both are returning data

peekFirst() it will get first collection of data.
pollFirst() it will get first collection of data.
element will remove given collection
from queue. it will take and give you

→ Array Deque - Dynamic only (double end queue)
internally mind Queue

Queue - DS => FIFO - First In, First Out

Index based accessing is not allowed \times

insertion will happen from both end also
both end will do insertion

offerFirst(), offerLast() \rightarrow \checkmark offer(index, val);

duplicates are allowed \checkmark

offer add we can't offer \times

→ Deque (Interface)

- collection, map, generic, enum, annotation
inner class package.

→ oldest of insertion is maintained probably
2 class

Priority Queue is → @ The following min-heap Data Structure
(Queue) is implemented

import java.util.*; all packages

- index based is not allowed
- add (Object), add (Collection) - methods present in Priority Queue
- add (int); addAll (all) -
- output will not get the index of insertion
- In entire collection which ever highest object that will be available at beginning

↳ add (30)

↳ add (60)

↳ add (50)

∴ o.p (q)

Output

[50, 30, 60]

→ min - heap

→ 100, 50, 150, 25, 45, 125, 175

input

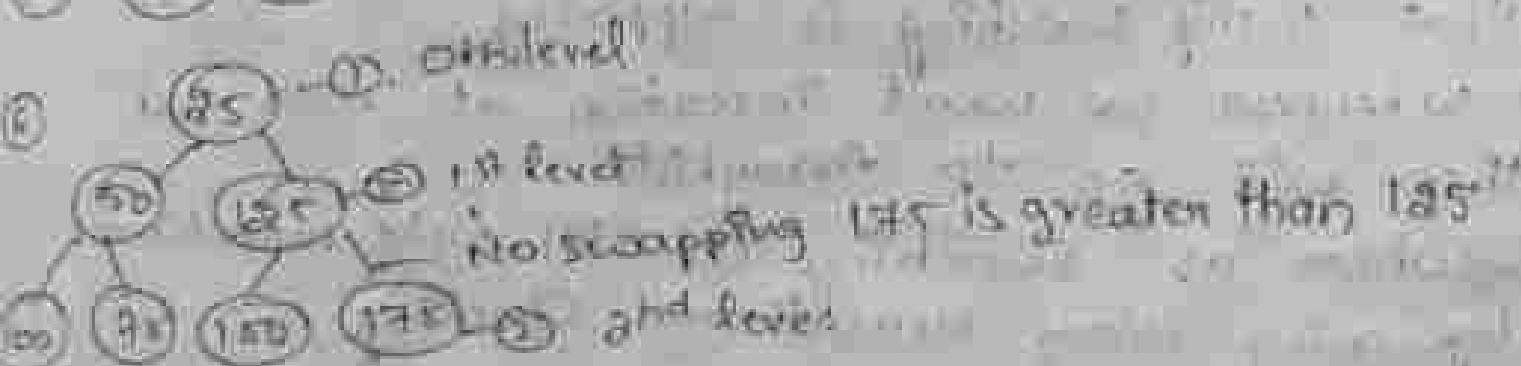
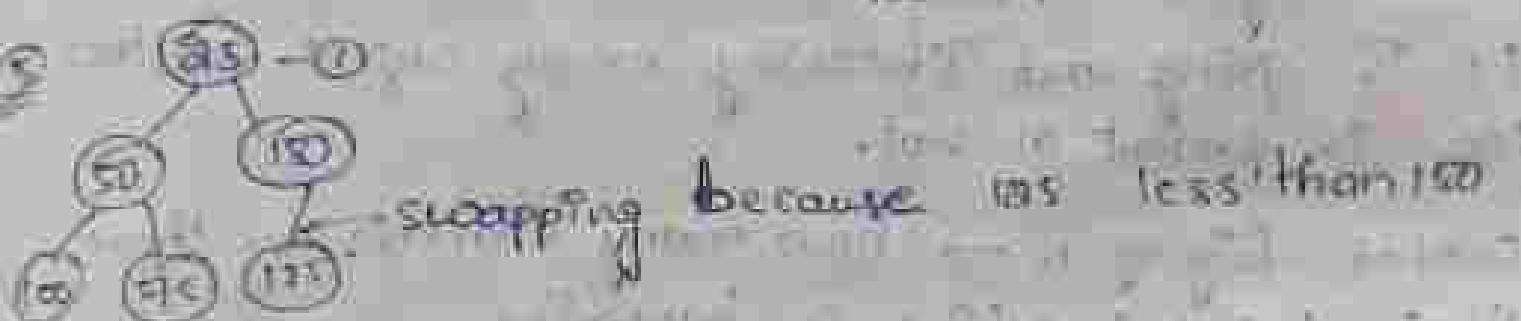
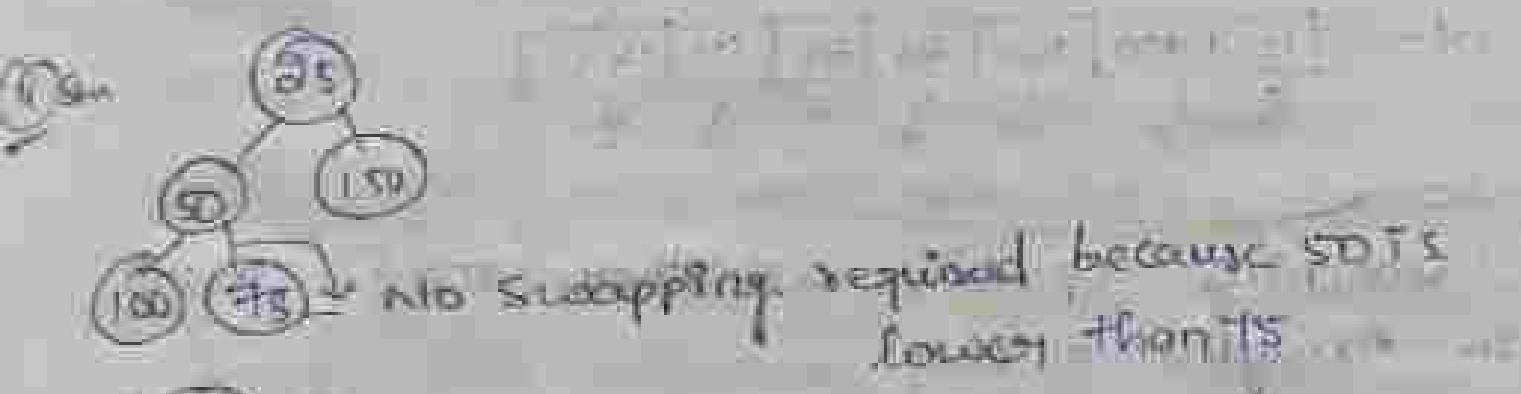
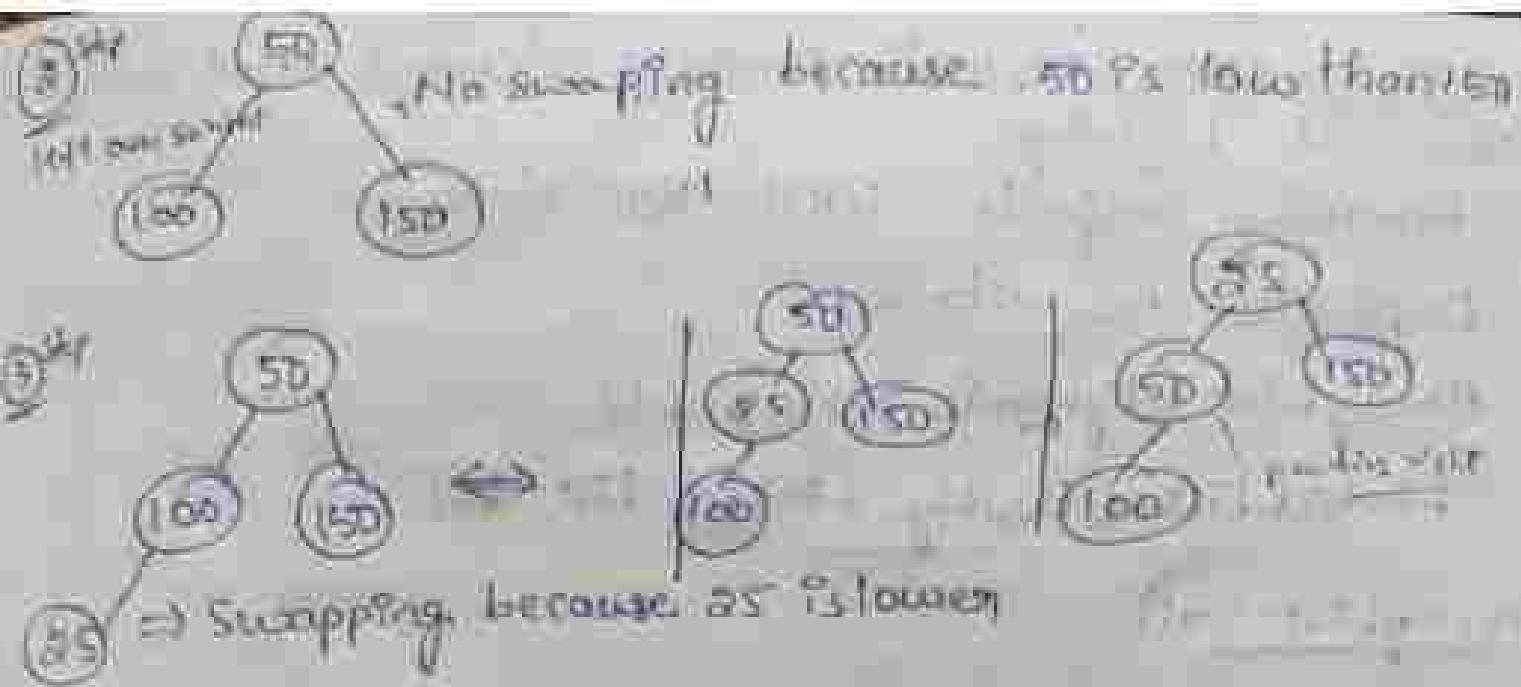
→ 25, 50, 125, 100, 45, 150, 175

output

∴ it is happening

adding of data will be left and right.





Output [85, 50, 125, 100, 145, 180, 175]

[85, 50, 125, 100, 145, 180, 175]

- which ever highest priority element that will come at beginning
- Remaining objects will not follow
- Duplicates are allowed.
- ArrayList (Dynamic Array DS)
- LinkedList (Doubly Linked List DS)

ArrayList (a1)



Key = 200

Searching Operation: whether 200 is present in ArrayList or not
It is going and checking every object whether 200 is present or not.

$O(n) \rightarrow$ Big of $n \rightarrow$ how many elements are there that many searching is happening

- whenever we want insertion at rear and other will go to ArrayList
- whereas as searching it will take more time by going every element and checking

LinkedList (ll) (it follows doubly linked list)

Data will be stored in the form of nodes



Searching

How many elements are there that many compositions
+ compositions $O(n) \Rightarrow \text{Big O of } n$

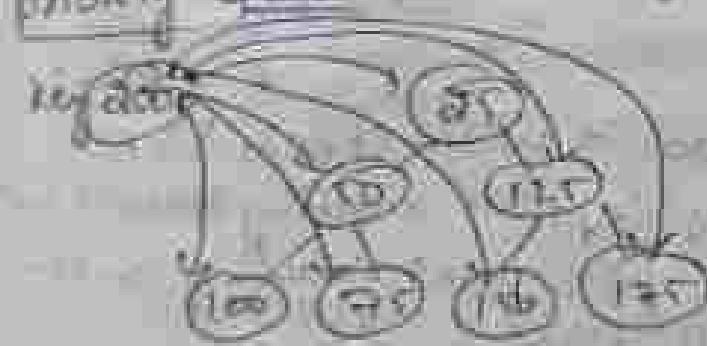
ArrayList (It follows Double ended Queue DS)



Searching \rightarrow Big O of n

How many elements are there that many compositions
+ compositions $O(n) \Rightarrow \text{Big O of } n$

Priority Queue : It follows min-heap



\rightarrow compositions

Big O of n $\rightarrow O(n)$

All above classes are good for insertion whereas
linked are taking so much time

Tree set ($O(n)$) It follows Binary Search Tree DS
+ If we want to get output as descending
Sorted order we can go with TreeSet

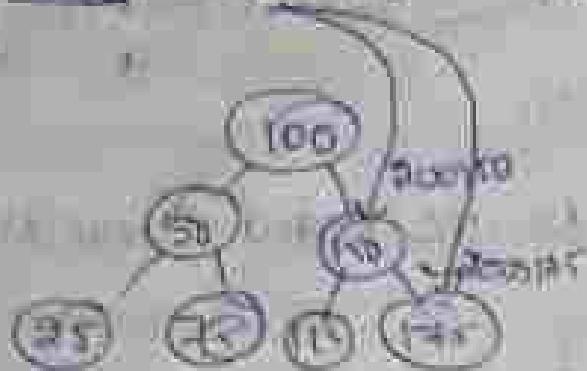
100 50 150 25 75 125 175 \rightarrow Input

50 100 150, 25, 125, 175 \rightarrow Output

Key = 200

200 > 100

Searching



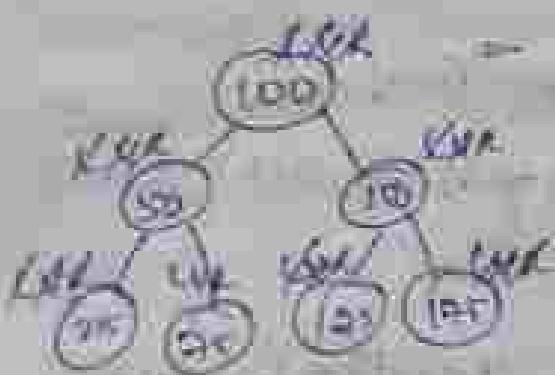
Q4 Q5 balanced tree

↳ Compositions, Searching

Q. Just compare in other classes.

Time Complexity $\rightarrow \Theta(\log n)$ happens based on Priority

It follows Pre-order traversal LVR means L = left
Q = value
R = right

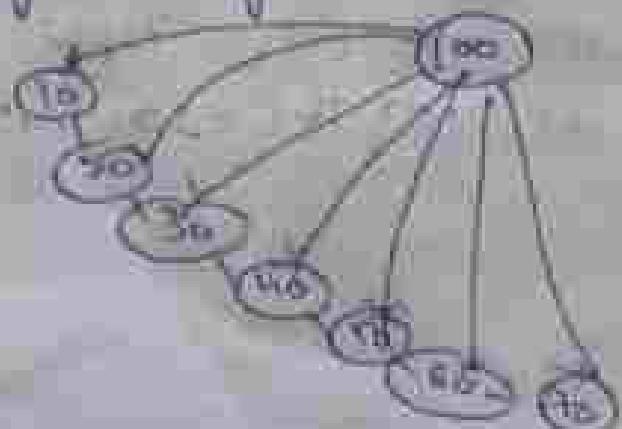


In 100 it will check LVR
means, left is anything then then
then it will print value of 100 then
it will Right

25 50 200 100 120 150 175 \rightarrow box will get
output

Input: 10 20 30 40 50 60 70

↳ apply Binary Search Tree



This type of Tree called
Skewed Tree

↑ objects

↑ compositions

Time Complexity O(n)

If we store the data by following Tree set
It will follows Priority Search Tree.
If we are using Tree set class to store data
as sorting order.

1) tree follows balanced binary tree and
store output then searching is faster
2) tree follows skewed tree and store output
then searching is slower so we should not go
searching here.

to build methods $\{p \Rightarrow 100, 50, 150, 20, 75, 125, 145\}$

ts.ceiling(50); //150 Ceiling method if we search in any obj
if that object is present that obj will give
ts.upper(50); //25 you same obj

ts.floor(75); //150 Floor method if we search in any obj
if that object is present that obj will
give you same obj.

ts.ceiling(40); //150 Ceiling method If we search in any obj
if that obj is not present then it will
ts.floor(40); //100 give you higher obj.

ts.floor(40); //100 Floor method will give you higher obj
ts.floor(40); //20 Floor method if we search in any obj
if that obj is not present then it will give you lower
value.

Hash set :- (It follows hashing algorithm)

⇒ Hash function & associated hash table both have factor (15%).



Because of formula which is present in hash function. Hash function and hash table works that if one insert any value that will go at that specific position in Hash table and sit at specific position in hash function because of formula present in hash function.

Suppose 100 is 15% means In hash table 15% filled buckets are there so that 15 buckets are filled then that hash table will become double.

Searching is very very fast in Hash Set.

Insertion based is not allowed.

Output will not get an insertion and it will not sort and highest priority not follow.

Linked list set :- (It follows hashing algorithm)

- It very fast & a Reconciling
 - Output will get as iteration
 - Index based is not allowed
- How to access the values present inside collection?

ArrayList al = new ArrayList();

al.add(10); // Auto boxing is happen to side

al.add(20); primitive to object

al.add(30); al.add(new int(40));

al.add(50);

al.add('c');

al.add("Vijay")

ArrayList al2 = new ArrayList();

al2.add(10);

al2.add("Vijay");

al2.add(20);

al2.add(30);

al2.add(40);

System.out.println(al2); [10, Vijay, 20, 30] -> just to point values

2. Ways are there to access & Index based is allowed only to List(i) & base

// loop normal \Rightarrow Here travelling
for (int i=0; i<al2.size(); i++)
Object o= al2.get(i);
S.o.p(o);
(or)
S.o.p (al2.get(i));
}

// for each -
for (Object obj : al2) {
S.o.p (obj);
}

// Iteration al2 \rightarrow [10|20|30|40|50|60]

One method is common for all the classes. Is
iterations. (if we want to access the data
in all the classes we can go with iterations).

Iteration it1 = al2.iterator(); \rightarrow whenever we
(call this method) active

if (it1.hasNext() == true) {
(, it will call iterate anywhere if the
is not

element in

it will go the location and give the

while (itr.hasNext()) {
 obj = itr.next();
 System.out.println(obj);
}

It has many values are there
that body should print.

Iteration i = (Integer) itr.next();
So p (itr.next()); → first to print.

It will fetch traverse direction also only to
ArrayList, linked list for only break
obj = listIterator.hasNext();

listIterator. $l_i = \text{listIterator}(\text{list}[\text{size}]);$
while (l_i.hasNext()) {
 System.out.println(l_i.next());
}

What if I want to reverse or access data
of other class in reverse direction?

Priority Queue Ad = new PriorityQueue

ad.add(10);

ad.add(20);

ad.add(30);

ad.add(40);

Index based insertion and deletion is not allowed
except ArrayList and LinkedList.

TreeSet add = new TreeSet();

ad. add (10);

ad. add (20);

ad. add (30);

ad. add (40);

TreeSet it = ad. iterator();

while (it. hasNext() == true) {

Integer i = (Integer) it. next();

System.out.println ("Array Element " + i);

}

LinkedList sl = new LinkedList();

sl.addAll (ad);

System.out.println (sl);

ListIterator li = (ListIterator) sl. listIterator (sl. size());

System.out.println ("Accessing TreeSet directly");

while (li. hasPrevious ()) {

li.

System.out.println (li. previous());

}

Iteration : Through a finite volume of data without data when we are travelling with the travelling we need to fetch the data. In that case iteration comes into picture

Iterator $\text{itr} = \text{ob.iterator}();$

$\text{L}, \text{collection name}$

hasNext() \rightarrow Iterator

will check whether next value

is there or not. next() \rightarrow that is length of collection

next() \rightarrow Iterator will go to next position and fetch the data also

Everything is stored as object

Linked list, Arraylist, TreeSet \rightarrow descending order

Iterator is there

Linked list $\text{ob} = \text{new Linkedlist}();$

1. add(10);

2. add(20);

3. add(30);

4. add(40);

1. descendingIterator();



2. ob.iterator() == the descendingIterator()

while (ob.iterator() != the descendingIterator())

2. System.out.println(ob.iterator().next()); \rightarrow it goes to (higher) & it prints

Before actual Java collections they were ~~packing~~ ~~unifying~~

Annotated \rightarrow before Iterator

Vector v = new Vector();

v.add(10);

v.add(20);

v.add(30);

v.add(40);

if we want access.

v.elements();

enumeration em = v.elements(); Iterator method Same as
v.iterator();

while (em.hasNextElements()) {

s.o.p(em.nextElement());

Same like Iterator, Iterator present in ~~java~~ class

Same like Iterator, Iterator is also there

Before collection topic Vector is also there
Vector enumeration is present it is working same
as Iterator.

why?

Before actual collections upto time Vector is
there means upto 1-2 versions now also Vector is there
if they remove Vector then previous projects build
by Vector those will gone so that's why still Vector is
there.

class	Index Access	Search	Iteration	BSL Access	Element Deletion	Element Insertion	Summarized
AL	✓	✓	✓	✓	✓	✗	✗
LL	✓	✓	✓	✓	✓	✗	✗
AD	✗	✓	✓	✗	✗	✗	✗
PQ	✗	✓	✓	✗	✓	✗	✗
TS	✗	✓	✓	✗	✗	✗	✗
HS	✗	✓	✓	✗	✗	✗	✗
LHS	✗	✓	✓	✗	✗	✗	✗

Hierarchy

Iterable(T)

↑

Collection(T)

↑
contains(T)

list(T)

→ AL
LL
vector
linked list
array

set(T)

set(T)

→ Hashset
Linked list

→ PQ

Deque(T)

↑

AD & class

SortedSet → iterator

↑
TreeSet & class

List L1 = new ArrayList(); *Two SE components*

Queue Q = new ArrayDeque();

List L2 = new LinkedList();

Deque D = new LinkedList();

d.peek();

If we → Create Deque with LinkedList we can't access all methods present in LinkedList

classes	DS	order of insertion	permit null value	duplicate
AL	Dynamic	✓	✓	✓
DL	double linked	✓	✓	✓
AD	double ended	✓	✗	✓
PQ	min-heap	✗	✗	✓
TS	Binary Search Tree	✗	✗	✗
HS	Hash Table	✗	✓	✗
LLS	Hash Table	✓	✓	✗

→ ArrayList al = new ArrayList();

al.add(100);

al.add(200);

al.add(200);

al.add(400);

for (int i = 0; i < al.size(); i++) {

S.o.p(al.get(i));

al.add(50); → we can add this line. program will not stop.

while accessing collection we attempt to modify collection then it is called Structured (or) Concurrent modification.

If we are Structured modification the output will not stop or execution.

If we are for each loop to access

collection and then we are using Structured modification then output will not stop (or) terminate.

Now by using Iterator (It is recommended to use iterator for accessing not "for" loop)

Iterator := al.iterator();

while (i.hasNext()) {

s.o.p (invent());

al.add (12); — Structural modification

} program is terminating abruptly

Output

Concurrent modification Exception → fail fast fail fast means it will fail very easily.

If we are doing Structured modification then we are using iterator it will fail fast.

If we are using for (or) for each loop if fail it not stop output (or) it will not fail.

Fail Safe: If we are doing structural modification then concurrent modification exception should not come. So then we can't use ArrayList (in normal classes). We have some special packages we should use those. CopyOnWriteArrayList (also known as CopyOnWriteArrayList),
Col-Add (1000),
Col-Add (500),
Col-Add (2000),
Col-Add (3000);
Iterator: $i = \text{col.iterator}();$
while ($i = \text{hasNext}();$) {
 S.o.p ($i \cdot \text{next}();$)
 Col-Add (10);
}

In this case we will not get any exception. In not getting output infinite. but concurrent modification will not allow very slowly.

Output: 100, 500, 2500, 3000

In all above three methods it's not possible to modify while access.

In normal for loop (in for each loop) we do structural modification output leads to infinite.

In Iterator while using normal ArrayList
we do structural code using normal
modification output leads to
concurrent modification exception - fail fast

In Iterator while using CopyOnWriteArrayList
we do structural modification output will
not be safe but it will not empty (a) not
leads to exception (b) not leads to infinite
output if it is called fail safe

TreeSet ts = new TreeSet(); (Binary Search Tree)

ts.add(100);

ts.add(650);

ts.add(150);

ts.add(75);

ts.add(125);

so p(ts) 50 75 100 125 150 = sorted order

ArrayList ts = new ArrayList();

ts.add(100);

ts.add(50);

ts.add(75);

ts.add(150);

Collections.sort(ts);

so p(ts);

Collection \rightarrow It is Interface / framework

Collection \rightarrow It is Class (Individual)

ArrayList ah = new ArrayList();

ah.add(10);

ah.add("Vijay");

ah.add(56);

ah.add("Kumar");

Collections.sort(ah);

System.out.println(ah);

Output

Exception

we can store heterogeneous data

allow we are trying to store
the will get exception

whenever data is homogeneous
we can go with Collections

whenever data is heterogeneous
we can go with Comparable
(Comparable Interface)

To avoid exception in above program we can use
generics means data should Same type

ArrayList<String> ah = new ArrayList<String>();

But we can't add multiple data with different
type.

ArrayList<Integer> ah = new ArrayList<Integer>();

ah.add(10); If accepts only Integer we are
able to add(10); testing after such ArrayList to
ah.add("10"); Since only Integer type.

class Student {

int age;

int marks;

public Student (int age, int marks) {

 this.age = age;

 this.marks = marks;

}
public int getAge () {

 return age;

}
public int getMarks () {

 return marks;

main () {

 Student s1 = new Student (12, 75);

 Student s2 = new Student (13, 86);

 ArrayList<Student> al = new ArrayList<Student>();

 al.add (s1);

 al.add (s2);

 Collections.sort (al);

 System.out.println (al);

Sorting can be

done for complex

objects

few more important inherit methods of collections

class

ArrayList<Student> al = new ArrayList<Student>();

al.add (s1);

al.add (s2);

al.add (s3);

al.add (s4);

int i = Collections.binarySearch (al, s2); it will give index.

S.O.p (i)

Collections .shuffle (all); \rightarrow we can shuffle our Collection

S.O.p (all);

Collections .min (only);

.min (all);

.frequency (all, no);

\leftarrow it will check how many times
no is there in collection.

~~value~~

~~key~~

all classes under collection hierarchy, API
framework has its own specially (features, inter-
BS) \rightarrow On obj / elements present using methods
we can perform.

Identifying card \rightarrow passport / Aadhar Card

home : Many } Many number of villages are
age : as } There in India

Address number \rightarrow address No } (Data name, duplicate
No.)

It is unique

No have same address no.

Data is present in form of pairs of key-value
key \rightarrow Associated data / value
 \rightarrow By using address number they
will be able to see own data (name, age, father name)

These type of data will store in map (key-value)

Map (key, value) (1.2 Y)

This is class under map interface

\rightarrow Hashmap \rightarrow widely used | Sometimes we go with
 \rightarrow Hashtable
 \rightarrow Properties
TreeMap (class) under
map interface

If we hear that data is present in form of pairs
then we need to store means we should go
with map interface

\rightarrow If we have simple data

key 410 Sachin, key-value we go with map
7. Mid Concept
18. Kohli

each pair of key-value is called entry

Key should be unique \rightarrow No duplicates

Value \rightarrow need not be unique \rightarrow Duplicates are
allowed Both together call entry

Interface Map

Interface entry { → Interface inside Interface is allowed

↳ $\{ \text{someMethods} \}$ entry is tightly coupled with map
means as long as map is present you
can't access that entry to access map

In map also data is stored as object Hashmap
follows hashing algorithm

HashMap Hp = new HashMap();

Hp.put (Key, Value);

Hp.put (10, sachin);

S.o.p (Hp); //output null

Hp.put (7, msd);

Hp.put (18, kohli);

S.o.p (Hp);

Output: but one getting in Key = value

↳ 18 = Kohli (7 = msd, 10 = sachin)

The output is not getting consider of insertion

LinkedHashMap Lhm = new LinkedHashMap();

Lhm.put (10, sachin);

Lhm.put (7, msd);

Output: { 10 = sachin, 7 = msd }

The output getting as order of insertion

Linked-Hash Map is sub class of hash map

HashMap → Jdk 1.2 v

LinkedHashMap → Jdk 1.4 v

Complex data

class Student {

String name;

int marks;

int age;

String city;

public Student (String name, int marks, int age) {

this.name = name;

this.marks = marks;

this.age = age;

public String getName() {

return name; }

}

public int getMarks() {

return marks; }

}

public int getAge() {

return age; }

}

public String toString() { return "

QUESTION

Student S1 = new Student ("Vijay", 99, 22);

1. S2 = " " ("Vaman", 98, 20);
2. S3 = " " ("Vijay", 91, 25);

flashmap hm = new FlashMap();

hm.put (1, S1);

hm.put (2, S2);

hm.put (3, S3);

S.o.p (hm); output : we got hashcode like 1 =

student1@973147

int a=10;

S.o.p (a); 1/10 \Rightarrow String

\hookrightarrow like method is going to call toString()
 \hookrightarrow Hashing();

\Rightarrow S.o.p (Object ref); Hashcode address of Object

S.o.p (object ref); Values will get

If we are not overriding to string () we will get
Hashcode. If it is overridden then we will get values

How to get values present inside Flashmap.

Now below are the Student Class presentation
previous page

Collection col = hm.values();

Iterator it = col.iterator();

colle City. In Next(3) &

So p.CurrNext(3)

3

Output Values will get

Vijay 98 22

Manish 97 20

Vijay 98 23

Set S = hm.entrySet()

SetKey Iterator <-- Iteratot iter = s.iterator()

while (iter.hasNext()) {

S.o.p(key);

3

Output Key will get

1,2,3

Set S = hm.entrySet()

Iterator iter = s.iterator()

while (iter.hasNext()) {

S.o.p(iter.next());

3

Output Values and Key we will get

1. Vijay 98 22

2. Manish 97 20

3. Vijay 98 23

Collection c = hm.entrySet();

c.iterator() --> Iterator itr = c.iterator();

hm.map[10] = new ArrayList();

hm.get(10).add("Vijay");

hm.get(22).add("Vijay");

while (itr.hasNext()) {

String s = (String)itr.next();
s.o.p(s);

s.o.p(itr.next());

}

Output : only we are getting values

Vijay

Vijay

Set s = hm.keySet();

s.iterator() --> Iterator itr = s.iterator();

while (itr.hasNext()) {

Integer i = (Integer)itr.next();

s.o.p(i);

s.o.p(itr.next());

}

Output : we will get only keys.

10, 22

Set s = hm.entrySet();

Iterator itr = s.iterator();

while (itr.hasNext()) {

s.o.p(itr.next()); s.o.p(i);

Output : Both key-value we get

[10 = Vinay, 12 = Vijay]

Above all three method output for

HashMap hm = new HashMap();

hm.put(10, "Sachin");

hm.put(11, "Vinay");

hm.put(12, "Vijay");

LinkedHashMap

HashMap

1.2V

DS: HashTable

order of insertion
will not get output same
parent of LinkedHashMap

1.4V

DS: HashTable doubly
linkedlist

Same output as order
of insertion
Sub class of Hash Map

Day 5 Simple program

```
class Person {
```

```
    String name;
```

```
    int age;
```

```
    public Person (String name, int age) {
```

```
        this.name = name;
```

```
        this.age = age;
```

```
}
```

```
public String getName() {
    return name;
}

public int getAge() {
    return age;
}

public String toString() {
    return name + " " + age;
}
```

```
main() {
    person p1 = new person("Vivay", 22);
    person p2 = new person("Kannan", 20);
    HashMap hm = new HashMap();
    hm.put(11, p1);
    hm.put(22, p2);
}
```

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

```
System.out.println("Please enter passport No:");
```

```
Integer key = scan.nextInt();
```

```
Boolean flag = false;
```

```
Set set = hm.entrySet();
```

```
Iterator ite = set.iterator();
```

```
while (ite.hasNext()) {
```

```
Map.Entry passport = (Entry) ite.next();
```

```
Integer key = passport.getKey();
```

```
(Integer) continue;
```

```
if (key == key1) {  
    sop ("Employee passport get Value()")  
}  
}  
if (flag == false) {  
    sop ("please passport not found")  
}
```

class Employee {

```
int empid;  
String name;  
String code;  
public String toString()  
    return "Hyder";
```

```
public void finalize()  
    sop ("Garbage collector collected");  
}
```

```
main()  
    employee = new Employee();  
    Hashmap hm = new Hashmap();  
    e = null; hm.put(e, "Hyder");
```

System.out sop ("Last line");

Internally, it internally calling finalize
method to clean object, which no one is
referencing.

Output

lost line

HashMap is dominating garbage collection
because hashmap is using employee reference

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

```
class Employee {
```

```
String name;
```

```
int age;
```

```
public void practice() {
```

```
System.out.println("Garbage Collected");
```

```
}
```

```
main()
```

```
Employee e = new Employee();
```

```
WeakHashMap hm = new WeakHashMap();
```

```
hm.put(e, "map");
```

```
e=null;
```

```
System.out.println();
```

```
System.out.println("lost line");
```

```
}
```

Output :-

Garbage Collected

lost line

je is dominating weak Hashmap

Hash Table

ArrayList > new HashTable
Hashtable
int put (11, "Alley")
int put (12, "Viffy")

System.out.println(ht);

Difference HashTable & HashMap

HashMap false (1-2)

Hashtable false (1-0)

Hashtable all methods are synchronized

HashMap methods are not synchronized (multiple threads)

is possible

Hashtable -> Thread Safe

Hashtable -> Thread not Safe

HashMap -> more performant

Hashtable -> high performance

HashMap -> high performance

HashMap always go

TreeMap()

TreeMap

int put (11, "Alley")

int put (12, "Viffy")

System.out.println(tm);

Interface (tell Java to implement Version)

Interface may tell we can't give implementation for methods which are present inside interface

void teaches();

void works();

}

Class Student implements Version

public void teaches();

System.out.println("Java");

public void works();

System.out.println("Java");

4

from Java is Inside Interface method with body is acceptable

Interface Version

void teaches();

we can give implementation

default

for method then we should

void disp();

compulsory use default

System.out.println("Hello");

very good

5

Class Version implements Version

public void disp();

we can override

System.out.println("Hello");

other methods

?

Interface Mixing

Static void disp()

so p ("Hello")

Execution have static
& default constructor
methods have body
inside interface

Ex 1. Implement interface if a method is
present in interface with static void disp()
not inherited. If we want to print static method
we can create reference of interface -> myobj disp()

- we will get output

Java or interface

private methods are not
accessible outside interface
of class. But we can
use private interface

Interface Mixing

private void disp()

so p ("private method")

3

Java or interface

private methods are not
accessible outside interface

of class. But we can
use private interface

private methods are not
accessible outside interface

of class. But we can
use private interface

Generics

- 1. Agenda
- 2. Type-Safety
- 3. Type-Casting
- 4. Generic Classes
- 5. Bounded Types
- 6. Generic methods and wild Card Character (?)
- 7. Communication with non-generic code
- 8. Conclusions

Defn: The main objective of generics is to provide Type-Safety and to resolve Type-Casting problems.

Case 1: Type-Safety

Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.

for example if our programming requirement is to hold String type of objects it is recommend to use String array.

In the case of String array we can add only String type of objects by mistake if we compile time error.

```
String name [ ] = new String [500];  
name [0] = "Kavin Rishi";  
name [1] = "Heiden";  
name [2] = new Integer (100); //LE
```

Incompatible types found : java.lang.Integer
required : java.lang.String

That is, we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type that is arrays are type safe.

But Collections are not type safe that is we can't provide any guarantee for the type of elements present inside collection.

For example, If our programming requirement is to hold only String type of objects then we can recommended to go for ArrayList.

By mistake If we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

Case 1: ArrayList al = new ArrayList();

al.add("Nitin Reddy");

al.add("Harley");

al.add(new Integer(10));

String name1 = (String) al.get(0);

String name2 = (String) al.get(1);

String name3 = (String) al.get(2); //Exception

java.lang.ClassCastException

java.lang.Integer cannot be cast to java.lang.String

String

Hence we can't guarantee safety of type of elements present inside Collections are not safe to use with respect to type.

Case 2: Type - Casting

In the case of array at the time of retrieval it is not required to perform any type casting

eg:

String name[] = new String[50];

name[0] = "Nitin Reddy";

name[1] = "Harley";

etc

String data = name[0]; // here type casting
is not required

But in the case of collection at the time of
retrieval compulsorily we should perform-type
casting otherwise we will get compile-time error

String name[] = new String[20];

name[0] = "Navin Reddy";

name[1] = "Hastech";

and type casting

String data = name[0]; // here type casting

is not required.

But in this case of collection at the time of
retrieval compulsorily we should perform-type casting
otherwise we will get compile-time error.

ArrayList al = new ArrayList();

al.add("Navin Reddy");

al.add("Hastech");

String name1 = al.get(0); // (E) Incompatible types

found : java.lang.Object required. Java.lang
String

String name1 = (String) al.get(0); // At the time of retrieval type casting is mandatory.
That is in Collections type casting is always a problem.

To overcome the above problems of Collections type casting Sun people introduced ^{Safety} Generics Concept.

1. To provide type safety to the Collections.
2. To resolve type casting problems.

To hold only String type of objects we can create a generic version of ArrayList as follows.

```
ArrayList<String> al = new ArrayList<String>();
al.add("NarayReddy");
al.add(10); // (Compile Time Error)
```

Symbol : method add (int) Location : class java.util.ArrayList<java.lang.String>
al.add(10);

For this ArrayList we can add only String type of objects by mistake if we were trying to add any other type we will get compile time error that is through generics we are getting type safety.

at the time of declaration it is not required
to perform any type casting but can assign
elements directly to String-type Variables.

ArrayList<String> al = new ArrayList<String>();
al.add("Naren Reddy");
System.out.println(al.get(0));

String name = al.get(0); // type casting
is not required as it is an type safe.

That is through generic System we can resolve
type casting problems.

Conclusion:-

1. polymorphism concept is applicable only for
the base type but not for parameter type
Usage of parent reference to hold child object
is called polymorphism.

2. ArrayList<String> al = new ArrayList<String>();
List<String> al = new ArrayList<String>();
Collections<String> al = new ArrayList<String>();
Collection<Object> al = new ArrayList<String>();
Incompatible types

3. Collections concept applicable only for objects, hence
for the parameter type we can use any classes or
interface name but not primitive.

Value (type) . otherwise cannot get completion
Error
e.g. `ArrayList<int> al = new ArrayList<int>;`

ller unexpected type found primitive required

19/12/22 Monday

enum : Java 1.5+ introduced

Constants Variable \rightarrow final

\hookrightarrow Note name is that name should be in Capital final int data <10; \rightarrow we can't change

DATA = 10; It is recommended to keep in Capital which has final Variable

enum : It is group of named Constants (6) It is group of predefined Constants

It refers to enumerated data | any own datatype

\Rightarrow NORTH, SOUTH, EAST, WEST is compass group of String type of data

To Create enum use keyword called "enum"

enum Result {

PASS, FAIL, NR; } // Static final Result pass

by default

enum Gender {

MALE, FEMALE, OTHERS; // Constant must be in
} // Capital

enum Company {

NORTH, SOUTH, EAST, WEST

class Person {

public static final int PASS = 25;

UPASS; // It leads to error. This file class we
should declare everything type of static final so on

Outside of class also we can write enum

Inside of class also we can write enum

enum Result {

PASS, FAIL, }

class Test {

enum Result {

PASS, FAIL, }

In program how many classes/intentiate/enum are there whenever we compile we will get separate class file for each class(es) interface(es) enum

4. Inside enum we can have fields / instance variables

-> Inside enum we can have methods and constructors

enum Result {

PASS, FAIL, NR; // static final

}

main() {

Result r = Result.PASS;

s.o.p(r);

Output:

PASS (itself it is a Variable and Value)

Behind the Scene

1) PASS → public static final Result PASS = new Result(); whenever we create enum internal class generated

2) FAIL → public static final Result FAIL = new Result();

Inside enum if we write constructor don't make it as public because it is invalid.

main() {

Course c = new Course("Java", "getCourseId()")

int id = c.getCourseId();
System.out.println(id);

}

Annotations :-

Comments :- These comments are only for developer after compiling. Comments will gone.

Annotations :- This annotations will give for developer and compiler and JVM also.

Annotation :- It is a extra information for developer, compiler and JVM.

Annotation :- Annotation \rightarrow parent of all Annotation.

Annotation \rightarrow Built in

\rightarrow Custom (our own)

@FunctionalInterface

Interface Test

Test getName();

{

Model

```
Test t = () -> { return " "; }
```

```
class JavaRecoring {  
    public void disp() {  
        System.out.println("parent");  
    }  
}  
class focus extends JavaRecoring {  
    @Override (inbuilt annotation)  
    public void disp() {  
        System.out.println("child");  
    }  
}
```

Annotation → we can use ~~the~~ class, interface, methods, fields (Variables), local Variables, constructors, parameters, enums.

Annotation - from Java 1.5

Can use ~~create~~ our own annotation

@ Target (ElementType)

@ Retention (RetentionType Runtime)

@ Interface (RocketPlayer) ~~the~~ tells compiler that it is not interface but it is annotation is being created

default interface

abstract class

@ RocketPlayer (RocketPlayer - India, USA, China)

Class Viratkohli {

int rank;

String name;

Setters and getters are generated

Main (S)

Virat Kohli VR = new Virat Kohli();

VR.setRank(200);

VR.setName("VR");

System.out.println(VR.getRank());

System.out.println(VR.getName());

What is the output?

Generics : (T, S, N)

Type parameter

ArrayList<T> ob = new ArrayList<T>();

Base parameter :

Generic classes

Until we use a non-generic version of ArrayList class is declared as follows.

Example

Class ArrayList {

add (Object o);

Object get (int index);

add() method. Can take object as the argument and hence we can add any type of object to the ArrayList. Due to this we can not get the type safety.

the return-type of method received the object from
and the object of return-type compilation & it should
perform type casting
and in Java a generic version of this function
is declared as follows

class ArrayList<T>

 odd (int index)

found in given requirement will be replaced
with user provided type
for example to hold only strong type of objects
we can create ArrayList objects as follows

Example:

ArrayList<String> l = new ArrayList<String>;
In this requirement compiler considered ArrayList as

Example:

class ArrayList<String>

 odd (String s);

 String get (int index);

odd() method can take only strong type as argument
hence we can add only strong type of objects to the list
if mistake if we write anything in odd() method
then we will get compile-time error

Ex-1 `ArrayList<String> al = new ArrayList<String>;`
al.add("Mavinreddy");
al.add("Sai");
al.add(10);
Symbol : method add (int)
location : class java.util.ArrayList
long string :
al.add(10);

Ex-2 `ArrayList<String> al = new ArrayList<String>;`
al.add("Mavinreddy");

String name = al.get(0); //Type Casting is required.
Hence through generics we are getting type safety.
At the time of retrieval it is not required to perform
any type casting we can assign the value directly
to string variables.

In Generics we are restricting a type or
type - parameter to the class such type of
parameterized class are nothing else nothing but generic
classes.

Generic class : class with type - parameters.

Based on our requirements we can create our own
generic classes also.

Example:

Class Account<T>

if

Account & Codes > gl = means Account of Goods

Account <silver> is now Account 433171

卷之三

fall 69 475

Trotz

Opportunities

Life obj -obj

build void Show();

```
System.out.println("the type of object is: "+obj.  
getclass()).get  
class();
```

Public Probate

Return address

1000 Genes for Demo 4

public static void main (String[] args) {

$\text{G}_{\text{int}} \times \text{Integ}_{\mathcal{O}} > \mathcal{O} = \text{Integ}_{\mathcal{O}} \quad \text{G}_{\text{int}} \times \text{Integ}_{\mathcal{O}} > (\mathcal{O})$

On the whole

```
System.out.println("getObjet(3)");
```

Can't string $\tilde{\varphi}_2$ unless $\text{Can't string } (\text{"magenta"},$
 $\text{blue})$

```
System.out.println(ja.getObjet());
```

Green <Osvaldo> 07:11 1998 Green <Osvaldo> (0.5)

THEORY

• ~~Topics out of print (e.g. geometry)~~

Output

The type of object is: java.lang.Integer 10

The type of object is: java.lang.String Interest

The type of object is: java.lang.Double 10.5

Note To get the underlying object of any reference type we use a method called `ref.getClassName()`

Ex:-

Introduce Calculation

Class Ratio implements Calculation

Class Quartz implements Calculation

Class Radio implements Calculation

Calculation obj = new Ratio();

```
System.out.println (obj.getClass().getName()); //Ratio
```

Calculation obj = new Quartz();

```
System.out.println (obj.getClass().getName()); //Quartz
```

Calculation obj = new Radio();

```
System.out.println (obj.getClass().getName()); //Radio
```

Lower bounded types

We can bound the type parameters to a particular range by using `extends` keyword. Such types are called bounded types.

Example

Class Test & T

```
class Test<T> {  
    T test (Integer> t) {  
        return test (String> t);  
    }  
}
```

Here we see that `test` has type `Integer`, but can pass any type and there are no restrictions, hence it is an **unbounded-type**.

Example 2

Class `Test<T extends X>`

- If `X` is a class, then `X` is the type parameter we can pass either `X` or its child classes.
- If `X` is an interface, then as the type parameter we can pass either `X` or its implementation classes.

class Test<T extends Number>

```
class Demo {  
    public static void main (String> args) {  
        Test<Integer> t1 = new Test<Integer>();  
        Test<String> t2 = new Test<String>();  
    }  
}
```

class Test<T extends Runnable>

```
class Demo {  
    public static void main (String> args) {  
        Test<Thread> t1 = new Test<Thread>();  
        Test<String> t2 = new Test<String>();  
    }  
}
```

Key points about bounded types

- we can't define bounded types by using implements and super keyword
 - But implements keyword purpose we can replace with extends keyword
- eg: `class Test <T implements Runnable> {}`
`class Test <T extends String> {}`

Another Topic

Key points about Bounded Types

- As the type parameter we can use any valid Java identifier but it's convention to use following
- eg. `class Test <T> {}`
`class Test <Dimension> {}`
- we can pass any no of type parameters need not to be one.

eg: `class HashMap<K,V> {}`

`HashMap<Integer, String> h;` also `HashMap<String, String> h;`

which of the following are valid?

class Test extends Number & Comparable // invalid
number → class

Number → interface

Comparable → interface

class Test extends Number & String // invalid
we can't extends more than one class at a time

class Test extends Number & Comparable // invalid
Number → interface
Comparable → interface

class Test extends Number & String // invalid
we can't extends more than one class at a time

class Test extends Number & Comparable // invalid
Number → interface
Comparable → interface

class Test extends Number & Comparable // invalid

Number → class

rule: first inherit and then implement so
invalid

Generic Class

Class Type parameter

Can we apply Type Parameter at method level?

Ans. Yes, if it is possible.

Generic methods and with - Card character (2)

• String type

① methodOne (ArrayList<String> al)

This method is applicable for ArrayList only

String type

methodOne (ArrayList<String> al) {

al.add ("Sachin");

al.add ("Rohit Yeddy");

al.add ("Pawar");

al.add (new Integer (10)); // invalid

al.add (new String ("10"));

Within the method we can add only String type
of objects and null to the list

② methodOne (new ArrayList<String>())

methodOne (new ArrayList<String>());

methodOne (new ArrayList<Integer>());

methodOne (new ArrayList<Runnable>());

ArrayList <?> l = new ArrayList

methodOne (new ArrayList<?>());

we can use this method to insert anything except
type but within the method we can add anything
to the list except null

Example

```
l.add(null); // invalid  
l.add("a"); // invalid  
l.add(10); // invalid
```

This method is useful whenever we are
performing only read operation.

methodOne (ArrayList<? extends Comparable> l)

→ In this we can make a call to method
by passing ArrayList of x type or it can be
type. Furthermore, we can make a call to method
by passing ArrayList of x type in the implementation
class.

methodOne (ArrayList<? extends Comparable> l){
l.add(null);}

Best suited only for read operation.

methodOne (ArrayList<? super Comparable> l)

→ In this case, we can make a call to
method by passing ArrayList of x type or it can be
super class of implementation class of x.

methodOne (ArrayList<? super Comparable> l){
l.add(x); l.add(null);}

- Which of the following declarations are allowed?
- ArrayList<String> arrList = new ArrayList<String>(); //Valid
 - ArrayList<?> arrList = new ArrayList<String>(); //Valid
 - ArrayList<?> arrList = new ArrayList<Integer>(); //Valid
 - ArrayList<?> arrList = new ArrayList<Number>(); //Valid

Type parameter at method level

↳ Type parameter at Class level

class Demo<T> {

↳ Type parameter defined
just before the return type
public <T> void m1(T t) {

}

- Which of the following declarations are allowed?
- public <T> void methodOne(<T>) { } //Valid
- public <T extends Number> void methodOne2(<T>); //Valid
- public <T extends Number & Comparable> void methodOne3(<T>); //Valid

public int extendable Number is Comparable< Number> and
overload method to be compared.

public int extends Number is Comparable< Number> and
overload method to be compared.

public int extends Number is Comparable< Number> and
overload method to be compared.

public int extends Number is Comparable< Number> and
overload method to be compared.

public int extends Number is Comparable< Number> and
overload method to be compared.

Communication with the generate code
To provide compatibility with old version Sun people
comprised the concept of generate proxy class
area in the following.

Example-

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.print.*;
import java.awt.print.PrinterJob;
import java.awt.print.PageFormat;
import java.awt.print.Paper;
import java.awt.print.**;
import java.awt.print.**;
import java.awt.print.**;
import java.awt.print.**;

```
public static void methodOne (ArrayList<String> l)
{
    l.add("a");
    l.add("dhoni");
    l.add("true");
}
```

Conclusion :-
Generics concept is applicable only at compilation time. Concept is applicable only at compilation time there is not such type of concept at runtime there is not such type of concept at the time of compilation, as the last step at the time of compilation, for implementation generics concept is removed, hence for implementation generics concept is not available.

Syntax :- ~~ArrayList<String> l = new ArrayList<String>();~~

~~ArrayList<Integer> l = new ArrayList<Integer>();~~

~~ArrayList<Double> l = new ArrayList<Double>();~~

All are equal at runtime becoz compiler will remove these generics syntax.

~~ArrayList<String> l = new ArrayList<String>();~~

Example :-

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

```
class Test {
```

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

```
        ArrayList<String> l = new ArrayList<String>();
```

↳ add (10)

↳ add (10, 2)

↳ add (true)

System.out.println("Hello, 10, 10.5, true")

Example 2:

import java.util.*;

class Test {

 public void methodOne (ArrayList<String> list)

 public void methodOne (ArrayList<Integer> list)

↳ duplicate methods found

Behind the scenes by the Compiler

1. Compiler will scan the code
2. check the assignment type
3. If generic found in the assignment type remove the generic syntax
4. Compiler will again check the syntax

Example 3:

The following 3 declarations are equal.

ArrayList<String> list = new ArrayList<String>();

ArrayList<String> list = new ArrayList<String>();

for these Arraylist Objects we can add only one type of objects.

l.add (2); // Valid

l.add (10); // Invalid

Comparable vs Comparable

public TreeSet ()

1 => when we use the above constructor
then internally we use Comparable interface method
to sort the objects based on default natural sorting
order.

What is Comparable Interface?

It is a functional Interface present in java.
This interface is internally used by TreeSet object
during sorting process of the Object.

@ functional Interface.

```
public interface java.lang.Comparable<T> {  
    public interface java.lang.Comparable<T> {  
        public abstract int compareTo(T);  
    }  
}
```

import java.util.*;

class Test

```
{  
    public static void main (String args)
```

// Sorting of objects will happen based on
default natural sorting order TreeSet is based
on TreeSet ()

```
ts.add("a")  
ts.add("b")  
ts.add("c")  
ts.add("d")
```

ts.add(null) // Null pointer exception

ts.add(10) // ClassCastException

System.out.println(ts); // Can't print

Note: Keeping the date inside TreeSet object

If we are keeping the date inside TreeSet object
then the date should be "Comparable".

a. homogeneous \Rightarrow because it uses Comparable
to sort the object.

b. The object should be "Comparing". Implement Comparable

Interface specified "Comparable".
If we don't do so, it would result in
"ClassCastException".

Ques?

Import java.util.

class Test {

public static void main (String[] args) {

} // Sorting of objects will happen based on default
natural sorting order
TreeSet ts = new TreeSet();

```
ts = new CharSequenceBuffer("A");
ts = ts.append("B");
ts = ts.append("C");
ts = ts.append("D");
ts = ts.append("E");
System.out.println(ts); // Class Cast Exception
```

Note :- All wrapper classes and String class has implemented Comparable interface. StringBuffer class has not implemented Comparable interface, so above program would result in "ClassCastException".

Note :- ~~Wednesday~~ - Sunday

Note :- If we are depending on Default Natural Ordering order, Comparable Objects should be Homogeneous and Comparable otherwise we will get RE : ClassCastException.

An object is said to be Comparable if and only if corresponding class implements Comparable interface. All wrapper classes, String class already implements Comparable interface but StringBuffer class doesn't implement Comparable interface.

Functional Interface

```
public interface Comparable<T> {  
    public abstract int compareTo(T);
```

obj1.compareTo(obj2)

- ↳ returns -ve value if obj1 has to come before obj2
- ↳ returns +ve value if obj1 has to come after obj2
- ↳ returns 0 value if both obj1 and obj2 are equal

import java.util.*;

Class Test {

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

```
TreeSet ts = new TreeSet();
```

```
ts.add ("A");
```

```
ts.add ("Z");
```

```
ts.add ("L");
```

```
ts.add ("K");
```

```
ts.add ("R");
```

```
System.out.println (ts);
```

Comparable<T>;

Comparable interface present in java.lang package and it contains only one method compareTo.

Obj 1. Composite (Obj2)

Return

Return - we'll find only if Obj1 has to come before Obj2
Return + we'll find only if Obj1 has to come after Obj2
Return & if only if Obj1 and Obj2 are equal

Code

```
System.out.println ("A".composite ("z")); //zero  
System.out.println ("z".composite ("k")); //one  
System.out.println ("z".composite ("z")); //zero  
System.out.println ("z".composite (null)); //NPF
```

whenever we are depending on Default Natural
Sorting Order and if we are trying to Insert
Elements then Functionally JVM will Call Composite()
to identify sorting order.

Reset to new TreeSet()

t.add ("k");

t.add ("z"); "z".composite ("k");

t.add ("m"); "m".composite ("k");

t.add ("r"); "r".composite ("k");

System.out.println (t); // [A, K, z] is sorting from

Note for String default natural sorting order
is "Ascending order".

for Number default natural sorting order is
"Ascending order".

Comparator (1)

Note: If we are not satisfied with Default natural sorting order OR if Default natural sorting order is Not Already Available Then we can Define Our own sorting by Using Comparator object.

public interface java.util.Comparator<T> {

 public abstract int compare (T v1, T v2);

 public abstract boolean equals (java.lang.Object);

Comparator(2)

This Interface present in java.util package.

Methods: It Contains 2 methods compare() and equals()

public int compare (Object obj1, Object obj2);

 Actions - we if and only if obj1 has to come Before obj2

 + we if and only if obj1 has to come After obj2

Return 0 if and only if obj1 and obj2 are Equal.

public boolean equals(Object obj)

Whenever we are implementing Compositing Compulsory we should provide implementation for compare()

Implementing equals() is optional because it is already available to our class from Object class through Inheritance.

Impact of equals()

Class TestDemo {

```
public static void main (String [] args) {  
    TestSet t = new TestSet();  
    t.add (10);  
    t.add (9);  
    t.add (15);  
    t.add (2);  
    t.add (10);  
    t.add (20);  
}
```

System.out.println(t) // [2, 9, 10, 15, 20]

Class MyCompositer implements Comparable {

```
public int compareTo (Object obj) {  
    Integer i1 = (Integer) obj;  
    Integer i2 = (Integer) obj;
```

```

if (i1 < i2)
    return i1;
else if (i1 > i2)
    return -i1;
else
    return 0;

```

Line 1: If we are not passing ComposedObject as an argument then JVM will call `comparator()`,

which is meant for Default Natural Sorting order (Ascending order).

In this case output is `[0, 1, 10, 15, 20]`.

At line 1 if we are passing ComposedObject then JVM will call `comparator()` instead of `comparator()`.

With it means the Customised sorting can be (Ascending / Descending order).

In this case the output is `[20, 15, 10, 5, 0]`.

Various possible Implementations of `comparator`

```

public int compare(Object obj1, Object obj2) {
    Integer i1 = (Integer) obj1;
    Integer i2 = (Integer) obj2;

```

```

return T1.compareTo());
return -T1.compareTo();
return T2.compareTo();
return -T2.compareTo(T1);
return +T;
return -T;
return 0;
}

```

Output

1. Ascending order
 2. Descending order
 3. Descending order
 4. Ascending order
 5. Insertion order
 6. Inverse of insertion order
 7. only first element will be inserted.
- Q) Write a program to insert String Objects into the Queue where the sorting order is of Reverse of Alphabetic order.
- Ans) Import java.util.*;
- Class TreeSetDemo
- ```

public static void main (String [] args) {
 TreeSet<String> myComp = new TreeSet (new myComparator());
}

```

```
t.add("sachin");
t.add("ponting");
t.add("sangakkara");
t.add("flawil");
t.add("kumar");
System.out.println(t);
```

3) **MyComposit** implements **Comparable** &  
class MyComposit implements Comparable<Object> {
 public int compareTo(Object obj) {
 String s1 = obj.toString();
 String s2 = (String) obj;
 return s1.compareTo(s2);
 }
}

Write a program to insert String buffer objects  
into the TreeSet where sorting order is alphabetical  
order:

```
import java.util.*;
class TreeSetDemo {
 public static void main(String[] args) {
 TreeSet t = new TreeSet(new MyComposit());
 t.add(new StringBuffer("a"));
 t.add(new StringBuffer("b"));
 t.add(new StringBuffer("c"));
 t.add(new StringBuffer("d"));
```

fixed-point string buffer (char\*)

## System-out printing (t)

Class MyComposite implements Composite  
public int composite (Object obj) {  
String s1 = obj[1].toString();  
String s2 = obj[2].toString();  
String s3 = obj[3].toString();  
return s1 + s2 + s3;  
}  
System.out.println (s1 + s2 + s3);

Write a program to insert String and String by Objects into the TreeSet where Sorting Order is Increasing length Order.

To 2 objects having some length, the consider  
their Alphabetic order.

Gr. A, ABC, AA, XX, ABCF, A

Output A, A, XX, ABC, ABC

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

Class: TreeSet Demo

public static void main (String[] args) {

Ticket for my Train (new my company)

4. and (new) strong buffer ("ABC"));

```
↳ add (new String("Hello Ad"))
↳ add ("XXX")
↳ add ("ACE")
↳ add ("AB")
```

System.out.println();

3

```
class myComparator implements Comparable {
 public int compare (Object obj1, Object obj2) {
```

```
 String s1 = obj1.toString();
```

```
 String s2 = obj2.toString();
```

```
 int n = s1.length();
```

```
 int m = s2.length();
```

```
 if (n > m) return 1;
```

```
 else if (n < m) return -1;
```

```
 else if (s1.equals(s2)) return 0;
```

```
 else return s1.compareTo(s2);
```

4

Note : if we use `use Comparator`, then that condition is

a. Object should be homogeneous.

b. object should be Comparable (class should implement Comparable()):

If we use `use Comparator<>` then nothing the condition?

a. object need not be homogeneous

b. object need not implement Comparable.

when to go for Comparable Interface and when to go for Comparator Interface?

Ans predefined Comparable classes like String, wrapper class  $\Rightarrow$  Default Natural Sorting is available

If we are not happy with Natural Sorting order we want Customization then we need to go for Comparable( $\pi$ )

For predefined Non-Comparable class like String, Comparable( $\pi$ ) is used for both Natural Sorting and Customized Sorting order.

For undefined class like Employee, Student  $\Rightarrow$  Developer if he comes up with logic then he should implement Comparable( $\pi$ ) Sorting, then he should implement Comparable( $\pi$ ) and give it as a ready made logic.

Ques Class Employee implements Comparable

```
int id;
String name;
int age;
public int compareTo(Object obj)
// sorting is done based on "id"
```

If the developer who is using Employee class if he is not interested with sorting based on "id" given by the api then he can use Comparator when he goes for Comparable and when goes for comparison.

### Comparable Vs Comparator:

- For predefined Comparable Classes (Like String) Default Natural Sorting Order is already available if we are not satisfied with that we can define our own sorting by Comparator Object.
- For predefined Non-Comparable Classes (like Stringbuffer) Default Natural Sorting Order is not already Available.

If we want to define our own sorting we can use Comparator Object.

- For our own Classes (Like Employee) the person who is using Employee class he is responsible to define Default Natural Sorting Order by implementing Comparable Interface.
- the person who is using our own class if he is not satisfied with Default Natural Sorting Order he can define his own sorting by using Comparator Object.

If he satisfied with Comparable then he can use his own class.

Use file to print Employee object  
into the TreeSet where DMSO is based on  
Ascending Order of EmployeeName & Customer  
Order is based on alphabetical order of Name

DMSO → Default Natural Sorting Order

Import java.util.\*;

Class Employee implements Comparable<

String name, int eid;

Employee (String name, int eid) {

this.name = name;

this.eid = eid;

public String toString () {

return name + " " + eid;

public int compareTo (Object obj) {

int eid1 = this.eid;

Employee e = (Employee) obj;

int eid2 = e.eid;

if (eid1 < eid2) return -1;

else if (eid1 > eid2) return 1;

else return 0;

class Test {

```
 public static void main (String [] args) {
 employee e1 = new Employee ("String1", 10);
 employee e2 = new Employee ("potrait", 11);
 employee e3 = new Employee ("Laxa", 9);
 employee e4 = new Employee ("Ran off", 17);
 employee e5 = new Employee ("Anuvi", 13);
 }
}
```

TreeSet t = new TreeSet();

t.add (e1);

t.add (e2);

t.add (e3);

t.add (e4);

t.add (e5);

System.out.println (t);

TreeSet t1 = new TreeSet (new myComparator ());

t1.add (e1);

t1.add (e2);

t1.add (e3);

t1.add (e4);

t1.add (e5);

System.out.println (t1);

class MyComposit implements Comparable & public int compareTo(Object obj, Object obj1){ Employee e1 = (Employee) obj1; Employee e2 = (Employee) obj; String s1 = e1.name; String s2 = e2.name; return s1.compareTo(s2); }

## Composition of Comparable and Comparable

### Comparable()

Present in java.lang package

It is meant for Default Natural Sorting Order

Defines only one method Comparable()

With wrapper classes and String class implements Comparable Interface.

### Comparable()

Present in java.util package.

It is meant for Customized Sorting Order.

Defines 2 methods Comparable() and equals()

The only implemented classes of Comparable are Collection and Queue interface.

## functional interfaces

- If an interface contains only one abstract method then such interfaces are called as "functional interface".

```
public interface java.util.function.Predicate<T> {
 public abstract boolean test(T);
```

### // default methods

```
public java.util.function.Predicate<T> and (java.util.function.Predicate<T> p1);
```

```
public java.util.function.Predicate<T> negate();
```

```
public java.util.function.Predicate<T> or (java.util.function.Predicate<T> p2);
```

### // static method

```
public static <T> java.util.function.Predicate<T> of (Object o);
```

### Usage of predicate

```
class my_predicate implements Predicate<Integer> {
```

#### @override

```
public boolean test (Integer i) {
 if (i > 0)
```

Instead of writing a separate class, we can write lambda expression as shown below:

class myPredicate implements predicate<String>

{

    @Override

    public boolean test (String name) {

        if (name.length() >= 3)

            return true

        else

            return false

class Test {

    public static void main (String [] args) {

        predicate<String> p = name → name.length() >

        System.out.println (p.test ("pwc"));

        System.out.println (p.test ("ics"));

default methods available as built-in methods for developer

public default predicate &gt; and (predicate)

public default predicate &gt; negation

public default predicate &gt; or (predicate, p)

import java.util.\*;

public class Test {

public static void main(String[] args) {  
int arr = {2, 10, 15, 20, 25, 30};

predicate <integers> p1 =  $i \rightarrow i > 10$ ;

System.out.println ("Elements greater than 10 are  
System.out.println ("

arr (p1, arr);

predicate <integers> p2 =  $i \rightarrow i \% 2 == 0$ ;

System.out.println ("Elements which are even  
System.out.println ("

arr (p2, arr);

System.out.println ("Elements which are odd  
System.out.println ("Should be even odd");

arr (p1, negation (p2), arr);

System.out.println ("Elements which are not  
even one is ");

arr (p1, negation (p2), arr);

public static void m1 (predicate <integers> p,  
arr (p1, arr);

```
for (int i = 0; i < 10; i++) {
 if (Composite.P. test (i)) //ele > 5
 System.out.println (i);
}
```

## function (T)

T → Input Type

R → return Type

```
public interface java.util.function.Function<T, R>
```

A 1 abstract method

```
public abstract R apply (T)
```

1 default methods

```
public <V> java.util.function.Function<T, V>
```

```
Compose<V> (java.util.function.Function<T, V> Super)
```

```
public <V> java.util.function.Function<T, V>
```

and then <V> (java.util.function.Function<T, V> Super)

Extends<V>)

// static method

```
public static <T> java.util.function.Function<T, T> id
```

Writing a code using Implementation Class  
class myfunction implements function<String, Integer>

    @override  
    public Integer apply (String name){  
        return name.length();  
    }

```
public class Test {
 public static void main (String [] args) {
 myfunction & function = new myfunction();
 int output = function.apply ("Sachin");
 System.out.println (output);
 }
}
```

Note = when to go for predicate and when to go for function?  
predicate → To implement some conditional checks we should go for predicate

function → To perform some operation and to return some result we should go for function.  
Method Reference (::) and Constructor Reference (::)  
:: ⇒ Scope resolution operator.

## Section 4.11: Method Reference

a. Static method  
    ClassName :: methodName

b. Instance method  
    object :: methodName

e.g:-

```
public class Test {
```

```
 public static void m1() {
```

```
 for (int i = 1; i < 10; i++)
```

```
 System.out.println("child thread")
```

```
 }
```

```
 public static void main (String [] args) throws Err
```

    //using method reference binded the method with

    run () of interface Runnable

```
 Runnable r = Test :: m1;
```

```
 Thread t = new Thread (r);
```

```
 t.start();
```

```
 for (int i = 1; i < 10; i++)
```

```
 {
```

```
 System.out.println ("main thread")
```

```
 Thread.sleep (1000);
```

```
 }
```

```
class Sample {
```

```
 private String s;
```

```
 Sample(String s) {
```

```
 this.s = s;
```

```
 System.out.println("Constructor executed");
```

```
 System.out.println("Sample object created");
```

```
@FunctionalInterface
```

```
interface Interface {
```

```
 public Sample get(String s);
```

```
public class Test {
```

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

```
 Interface i = s -> new Sample(s);
```

```
 i.get("from lambda expression");
```

```
 System.out.println(i);
```

```
 // Constructor output
```

```
 // constructor reference
```

```
 Interface i1 = Sample::new;
```

```
 i1.get("from Constructor reference");
```

## Example of Method References

### ② Functional Interface

interface Interface

```
 public void m1 (int i);
```

```
public class Test {
```

// logic coded by other developer

```
 public void m2 (int i) {
```

```
 System.out.println (i*2);
```

System.out.println ("logic coming from  
method reference ---");

```
}
```

```
public class Main {
```

String x = m2;

```
 m1 (10);
```

```
 System.out.println (x);
```

// method reference (binding the body of m1 to  
abstract method m1)

```
Interface i1 = new Test ()::m1;
```

```
 i1.m1 (20);
```

to demonstrate the usage of approach to print the elements of arraylist

```
import java.util.*;
import java.util.function.*;
// public void for each (java.util.function.Consumer<T> consumer)
// public abstract void accept (T t)
```

```
class myConsumer implements Consumer<String>
```

```
@Override
public void accept (String name) {
```

System.out.println ("accept method got called..");

```
System.out.println (name);
```

```
public class Test {
```

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

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add ("Sachin");
```

```
names.add ("Dhoni");
```

```
names.add ("Kohli");
```

```
names.add ("Dhoni");
```

```
/* traditional approach
```

```
Consumer<String> consumer = new myConsumer();
names.forEach (consumer);
System.out.println (list);
```

## 11 Lambda Expression

names.forEach(name -> System.out.println(name))  
System.out.println();

## Method reference

names.forEach(System.out::println);

## Stream API

Stream -> Channel through which there is a  
free flow movement of data.

Streams  
To process objects of the collection in its version.  
Streams concept introduced.

What is the difference between Java-8 streams  
and Java 10 streams?

Java-8 streams meant for processing objects from  
collection - i.e., it represents a stream of objects from  
collection but Java 10 streams meant for processing  
binary and character data with respect to file  
binary and character data of binary data of char  
i.e., it represents stream of binary data of char  
data from the file hence Java 10 streams and  
Java-8 streams both are different

- what is the difference b/w Collection and Stream?
- If we want to represent a group of individual objects as a single entity then we should go for collection.
  - If we want to process a group of objects from the collection then we should go for Stream.
  - we can create a Stream object to the collection by using Stream() method of Collection interface. Stream() method is a default method added to the collection.

### 1.3 Youthon

```
import java.util.*;
import java.util.stream.*;
public class Test {
 public static void main(String[] args) {
 ArrayList<Integer> al = new ArrayList<Integer>();
 al.add(10);
 al.add(5);
 al.add(10);
 al.add(15);
 al.add(20);
 al.add(25);
 System.out.println(al); // [10, 5, 10, 15, 20, 25]
 // In jdk 1.8
 ArrayList<Integer> evenList = new ArrayList<List<Integer>>();
```

```
for (Integer i : al)
```

```
 if ((i % 2 == 0))
```

```
 evenList.add(i)
```

```
System.out.println (evenList); // [6, 10, 20]
```

// From jdk 1.8 we have streams

1. Configuration  $\Rightarrow$  Stream

2. Processing  $\Rightarrow$  Filter (if i % 2 == 0) - collect (collection)

List<Integer> streamList = al.stream().filter(i -> i % 2 == 0).collect(Collectors.toList());

```
System.out.println (streamList);
```

```
StreamList.forEach (System.out::println);
```

3.

Import java.util.\*;

import java.util.stream.\*;

public class Test {

```
 public static void (String[] args) {
```

```
 List<Integer> al = new ArrayList<Integer>();
```

```
 al.add(0);
```

```
 al.add(5);
```

```
 al.add(10);
```

```
 al.add(15);
```

```
 al.add(20);
```

```
 al.add(25);
```

System.out.println();

// from Jdk 1.8

```
list <-- Integer > DoubleList = new ArrayList<
 Integer>();
for (Integer i : list) {
 DoubleList.add(i * 2);
}
System.out.println(DoubleList);
```

System.out

// from Jdk 1.8  
// for every object, if a new object has to  
// map -> for every object, if a new object has to  
// be created then offer map

```
list <-- Integer > StreamList = list.stream().map(obj-
> obj * 2).collect(Collectors.toList());
System.out.println(StreamList);
StreamList.forEach(i -> System.out.println(i));
System.out.println();
StreamList.forEach(System.out::println);
```

Stream is an interface present in `java.util.stream`.  
Once we got the Stream, by using that we can process  
objects of that collection  
we can process the objects in the following 2 phases  
1. Configuration  
2. processing

Q) Configuration: We can configure either by using filter mechanism or by using map mechanism.   
filtering: we can configure a filter-to-filter mechanism by using map mechanism.   
elements from the collection based on some boolean condition by using.

filter() method of Stream Interface.

public Stream filter (predicate  $\lambda t$ )  
Here (predicate  $\lambda t$ ) can be a boolean value  
function / lambda expression

Ex: Stream  $\leftarrow$  Stream

stream  $\leftarrow$  stream.filter ( $\lambda x \rightarrow x \geq 0$ )

Here to filter elements of collection based on some Boolean condition we should go for filter() method.

mapping: If we want to create a separate new object for every object present in the collection based on our requirement then we should go for map() method of Stream Interface.

public Stream map (function  $f$ )

It can be lambda expression also

Stream s = Stream();

Stream st = s.map(f -> f(0));

Here we performed Configuration for Com process  
objects by using several methods.

a) processing

processing by collect() method

processing by count() method

processing by sorted() method

processing by min() and max() methods

processing by

for each() method

interview() method

Stream of() method.

Ex :-

import java.util.\*;

import java.util.stream.\*;

public class Test {

public static void main (String[] args) {

ArrayList <String> names = new ArrayList<String>();

names.add ("Sachin");

names.add ("Sourav");

names.add ("Dhoni");

names.add ("Kohli");

names.add ("Yuvraj");

System.out.println("names");

```
List<String> result = names.stream().filter(
 name -> name.length() > 5).collect(Collectors.toList());
```

System.out.println(result.size());

```
long count = names.stream().filter(name ->
 name.length() > 5).count();
```

System.out.println("The no of objects, whose  
length is > 5 is " + count);

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

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

```
// Comparable (pre-defined API as natural ordering)
// Comparable (User-defined class for customized
// ordering Order) ->
```

```
comparable(obj1, obj2)
```

```
public class Test {
```

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

```
 ArrayList<Integer> al = new ArrayList<Integer>();
```

```
 al.add(10);
```

```
 al.add(0);
```

```
 al.add(15);
```

```
 al.add(5);
```

```
 al.add(0);
```

```
System.out.println("Before Sorting :: " + result);
//using stream API
List<Integer> result = al.stream().sorted()
.collect(Collectors.toList());
System.out.println("After Sorting :: " + result);
List<Integer> customizedResult = al.stream()
.sorted((i1, i2) -> i2.compareTo(i1)).collect(Collectors.toList());
System.out.println("After Sorting :: " + customizedResult);
```

3. Comparable

```
import java.util.*;
import java.util.stream.*;
//Comparable (predefined API for natural sorting order)
//Comparable (predefined API for natural sorting order)
//Comparable (ref: user-defined class API customized)
//Comparable (ref: Comparable (obj1, obj2))
Sorting order -> Comparable
```

```
public class Test {
 public static void main(String[] args) {
 ArrayList<Integer> al = new ArrayList<Integer>();
 al.add(10);
 al.add(5);
 al.add(15);
 al.add(5);
 }
}
```

odd-odd (20):

```
System.out.println("array list is :: " + list);
```

```
Object[] objarr = list.stream().toArray();
```

```
for (Object obj : objarr)
```

```
System.out.println(obj);
```

```
System.out.println();
```

```
Integer[] objarr = list.stream().mapToInt(Integer::
```

```
intValue).toArray();
```

```
for (Integer obj : objarr)
```

```
System.out.println(obj);
```

3

import java.util.\*;

import java.util.stream.\*;

public class Test {

public static void main(String[] args)

// Stream API ==> Collections (group of

Objects)

```
Stream s = Stream.of(9, 99, 999, 9999);
```

```
s.forEach(System.out::println);
```

```
System.out.println();
```

```
Double[] d = {10.0, 10.1, 10.2, 10.3, 10.4};
```

```
Stream s1 = Stream.of(d);
```

```
s1.forEach(System.out::println);
```

collect()  
The method collects the elements from the stream and adding to the specified to the collection initialized (specified) by argument.

eg:-

```
import java.util.*;
import java.util.stream.*;
public class Test {
 public static void main (String [args]) {
 ArrayList<String> names = new ArrayList<String>();
 names.add ("Sachin");
 names.add ("Satyav");
 names.add ("Akhon");
 names.add ("Yusfi");
 System.out.println (names); // [Sachin, Satyav, Akhon, Yusfi]
```

filter (T)

```
public abstract boolean test (T);
List<String> result = names . Stream () . filter (name
name.length (15)) . collect (Collector . toList ());
System.out.println (result);
```

function (T) filter

public abstract F apply (T);

```
List<String> mapResult = names . Stream () . map
name -> name . toUpperCase ();
System.out.println (mapResult);
```

Geant

This method returns number of elements present in the Stream `ps`.

public long Geof()

Import java.util.\*;

Import Taxa UK. 340000.00

## public class Test {

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

honey-odd ("sackin")

Homotopy = odd ("Smarandache")

honey, and (when?)

Womer odd ("yawn")

System.out.println("names"); // Father, Son  
System.out.println("names"); // Son, Father

Long count - name  
Length(175) - count(1)

System-cut prints (cont'd):

processing by selected method

If we sort the elements present inside `struc`,  
then we should get sorted by method.

The setting can either default (without setting a  
order) or customized setting order specified by compiler  
called **default natural setting** or  
set by (compiler c) - **Customized setting**.

```
import java.util.*;
import java.util.ArrayList;
public class Test {
 public static void main (String[] args) {
 ArrayList<Integer> list = new ArrayList<Integer>();
 list.add (10);
 list.add (20);
 list.add (30);
 list.add (40);
 list.add (50);
 list.add (60);
 System.out.println (list);
 List<Integer> result = list.stream ()
 .collect (Collectors.toList ());
 System.out.println (result);
 List<Integer> customizedResult = list.stream ()
 .sorted (Comparator.comparing (i1, i2) -> i1 - i2)
 .collect (Collectors.toList ());
 System.out.println (customizedResult);
 }
}
```

III processing by min() and max() methods  
min(Comparator c) returns minimum value  
according to specified Comparator.

max(Comparator c)  
maximum value according to  
specified Comparator.

import java.util.\*;

import java.util.stream.\*;

public class Test {

public static void main (String [] args) {

ArrayList<Integer> al = new ArrayList<Integer>();

al.add(10);

al.add(20);

al.add(0);

al.add(15);

al.add(5);

al.add(15);

System.out.println(al);

Integer minValue = al.stream().min((i1, i2) ->

i1.compareTo(i2)).get();

System.out.println(minValue);

Integer maxvalue = 100; int count = 0; for (int i = 1; i <= maxvalue; i++) {

4. compare To (fa) 11-9-06

## System - exit - problem (manövler)

## An English method

This method will not return anything.  
This method will take lambda expression as  
argument and apply that lambda expression to  
each element present in the Stream.

Impact factor article

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

```
public class Test {
 public static void main (String [] args) {
 ArrayList <String> names = new ArrayList<String>();
 names.add ("André");
 names.add ("Ric");
 names.add ("Cec");
 names.add ("Dido");
 }
}
```

however, `Stream` (2) for each `Client`  $\rightarrow$  `System` output function

www.silene.com/softouch (Systemsoft®) parallel 35

## copyTo method

We can use copyTo method to copy elements present in the stream into specified array.

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

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

```
public class Test {
```

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

```
 ArrayList < Integer > al = new ArrayList< Integer >();
```

```
 al.add (0);
```

```
 al.add (10);
```

```
 al.add (5);
```

```
 al.add (20);
```

```
 al.add (15);
```

```
 System.out.println (al);
```

```
 Integer [] array = al.stream().toArray (Integer[]::new);
```

```
 for (Integer element : array)
```

```
 System.out.println (element);
```

## 1. Stream API Methods

We can also apply a stream to group by values and then map.

```
Stream<String> stream = Stream.of("a", "b", "c", "d", "e");
```

```
stream.forEach(System.out::println);
```

```
Double[] d = {10.0, 10.1, 10.2, 10.3};
```

```
Stream<Double> stream = Stream.of(d);
```

```
stream.forEach(System.out::println);
```

## File Handling

What is persistence?

It is a mechanism of storing the data permanently on to the file.

In java persistence can be achieved through API's available inside package named "java.io".

## Input and Output Operations

Agenda

1. file
2. file writer
3. file reader
4. buffered writer
5. buffered reader

## file

```
File f = new File("abc.txt");
```

This line 1st checks whether abc.txt file is already available (or) not if it's already available then "f" simply refers that file

If it is not already available then it won't create any physical file just creates a java file object represents name of the file.

### Example :

```
import java.io.*;
```

```
Class fileDemo{
```

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

throws IOException {

```
File f = new File ("Cricket.txt");
```

```
f.createNewFile();
```

f.createNewFile();

```
System.out.println (!f.exists()); //true
```

1. Output

False

True

2. Output

True

True

A Java file object can represent a directory also.

Example:

```
import java.io.File;
import java.io.IOException;
class fileDemo {
 public static void main (String [] args) throws IOException {
 File f = new File ("TPL Teams");
 System.out.println (f.exists()); // false
 f.mkdirs (); // creates a new directory
 System.out.println (f.exists()); // true
 }
}
```

File Object and File class in Java

Note: In Unix everything is a file, Java "file" is based on Unix operating system. hence in Java also we can represent both files and directories by file object only.

## file class Constructors

① `file f = new file (String name);`

→ Creates a java file object that represents name of the file or directory in current working directory

ex:- `file f = new file ("abc.txt");`

② `file f = new file (String subdirname, String name);`

→ Creates a file object that represents name of the file in directory present in specified subdirectory

ex:- `file f = new file ("abc");`  
`file f1 = new file ("abc");`

`file f2 = new file ("abc", "demo.txt");`

③ `file f = new file (File subdir, String name);`

ex:- `file f = new file ("abc");`

`file f1 = new file ("abc");`

`file f2 = new file (f1, "demo.txt");`

## Requirement

→ Write code to create a file named with demo.txt in current working directory

Ans  
↳ abc.txt

## Program

```
import java.io.*;
class fileDemo {
 public static void main (String [] args) throws
 IOException {
 File f = new File ("Demo.txt");
 f.createNewFile();
 }
}
```

## Requirement

- write code to create a directory named with Ipl team in current working directory and create a file named with abc.txt in that directory
- create
- Ipl team
- abc.txt

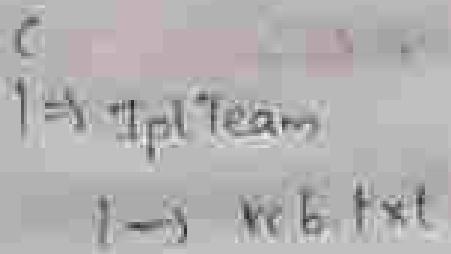
## Program

```
import java.io.*;
class fileDemo {
 public static void main (String [] args) throws
 IOException {
 File f1 = new File ("Ipl Team");
 f1.mkdir ();
 File f2 = new File ("Ipl Team", "abc.txt");
 f2.createNewFile();
 }
}
```

## Q. CreateNewfile()

3

Requirement: Write code to create a file name with web.txt present in C:\iplTeam folder.



### Program:

Import java.io.\*;

class Demo{

public static void main (String args) throws Exception

{File f=new file ("C:\iplTeam\","web.txt");

f.createNewfile();}

Assuming C:\iplTeam should be already available  
otherwise, it would result in 'file Not found'

Exception.

Important methods of file class:

boolean exists():

Returns true if the physical file directory  
available

boolean createNewFile():

This method 1st checks whether the physical  
file is already available or not. If it is  
already available then this method simply returns  
false without creating any physical file.  
If this file is not already available then  
it will create a new file and returns true.

④ boolean mkdir():

This method is checking whether the directory  
is already available or not if it is already  
available then this method simply returns  
false without creating any directory.  
If this directory is not already then it  
will create a new directory and returns true.

⑤ boolean isfile():

Returns true if the file object represents a physical  
file.

⑥ boolean isDirectory():

Returns true if the file object represents a  
directory.

## ④ String[] list()

It returns the names of all files and subdirectory present in the specified directory.

## ⑤ long length()

• Returns the no. of characters present in the file

## ⑥ boolean delete()

• To delete a file or directory

## Requirement :-

Requirement :- Write a program to display the names of all files and directories present in D:\ in Java. Job guarantee Batch

Requirement :- Write a program to display only file names.

Requirement :- Write a program to display only directory names.

import java.io.\*;

class Test {

```
 public static void main(String [] args)
```

```
 { int dirCount = 0;
```

```
 int fileCount = 0;
```

```
 int extFileCount = 0;
```

```
int zipFileCount = 0;
String location = "D:\\java\\job guarantee batch";
File file = new File(location);
String[] names = file.list();
for (String name : names) {
 if (name.equals(".")) {
 continue;
 }
 if (name.equals("..")) {
 continue;
 }
 if (file.isDirectory(name)) {
 dirCount++;
 }
 if (file.isFile(name)) {
 if (name.endsWith(".jpg")) {
 jpgFileCount++;
 }
 if (name.endsWith(".txt")) {
 txtFileCount++;
 }
 if (name.endsWith(".zip")) {
 zipFileCount++;
 }
 }
}
System.out.println("Count");
System.out.println("Number of zip files = " + zipFileCount);
System.out.println("Number of jpg files = " + jpgFileCount);
System.out.println("Number of txt files = " + txtFileCount);
System.out.println("Number of directory = " + dirCount);
```

```
System.out.println("Total no. of JPG files is: " + jpgfilecount);
System.out.println("Total no. of txt files is: " + txtfilecount);
System.out.println("Total no. of Zip files is: " + zipfilecount);
System.out.println("Total no. of Directory is: " + directorycount);
}
// Main Shutdown Hello
```

## FileWriter:

By using `FileWriter` - Object we can write character data to the file.

## Constructors

```
FileWriter fw = new FileWriter("String name");
```

```
FileWriter fw = new FileWriter(File f);
```

The above 2 constructors are used for creating the object for the file.

Instead of overriding if we want append operation then we should go for the following constructor.

filewriter -> new filewriter (String name, boolean append);

constructor of new filewriter (file &, boolean append);

If the specified physical file is not already available then these constructor will create that file.

methods :-

① write (int ch);

To write a single character to the file

② write (char [ ] ch);

To write an array of characters to the file

③ write (String s);

To write a string to the file

④ flush();

To give the guarantee the total data include last character also written to the file

⑤ close();

To close the stream.

import java.io.FileWriter;

import java.io.IOException;

```
public class TestApp {
```

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

```
 FileWriter fw = new FileWriter ("abc.txt");
```

```
 fw.write ("abc");
```

```
 fw.write ("new\nTechnology\nprivate
```

```
 fw.write ("new");
```

```
 char ch [] = { 'a', 'b', 'c' };
```

```
 fw.write (ch);
```

```
 fw.flush ();
```

```
 fw.flush ();
```

A new file will be created automatically

abc.txt

new  
Technology

private

limited

abc

## Note

- The main problem with Filewriter is we have to insert line separator manually which is difficult to the programmer. (\"n")
- And even line separator missing from System to System.

## file reader :

- By using file reader object we can read character data from the file.

## Construction

file Reader for = new file Reader (String name);  
file Reader for = new file Reader (file f);

## methods

### ① int read();

If return pls. to read next character from the file and return its unicode value. If the next character is not available then we will get -1.

### ② int i = read();

### ③ System.out.println((char i));

As this method returns Unicode value, while printing we have to perform type casting.

Q int readChar() {

It attempts to read enough characters of the file into char() array and returns the no of characters copied from the file into char() array.

④ file f=new file ("abc.txt");

⑤ char[] ch=new char [(int)f.length()];

⑥ void close();

class

import java.io.FileReader;

import java.io.IOException;

public class TestApp {

public static void main (String [args])

file reader fr=new FileReader ("abc.txt");

int i=fr.read();

while (i>-1){

System.out.println ((char)i);

i=fr.read();

}

file reader.close();

part 2

## Reading an array of characters

abc.txt

1000 characters are available

### Scenario 1:

file Reader fr = new FileReader("abc.txt")

char ch = fr.read();

char ch[] = new char[10];

int noOfCharactersCopied = fr.read(ch);

### Scenario 2:

file Reader fr = new FileReader("abc.txt")

char ch[] = new char[1000];

char ch[] = new char[1000];

int noOfCharactersCopied = fr.read(ch);

import java.io.FileReader;

import java.io.IOException;

import java.io.File;

public class TestApp {

public static void main (String args) throws IOException

File f = new File ("abc.txt");

file Reader fr = new FileReader(f);

char ch[] = new char [1000]; length;

fixed (ch);

```
String data = new String(ch);
System.out.println(data);
for close();
```

Usage of FileWriter and FileReader is not recommended because of following reason.

- ① While writing data by file writer compulsory we should insert line separator (\n) manually which is a bigger headache to the programmes.
- ② While reading data by file reader we have to read character by character instead of line by line which is not convenient to the programmes.

Assume we need to search for a 10 digit mobile no present in a file called mobile.txt.

⇒ Since we can read only character just to search one mobile no to searching and to search 10,000 mobile no we need to read 100,000 times. So performance is very low.

- ③ To overcome these limitations we should

so for BufferedWriter and Buffered Reader concept

### BufferedWriter

By using BufferedWriter we can write the characters  
data to the file

It can't communicate with the file directly  
can communicate only with writer object

### Constructor

BufferedWriter bw = new BufferedWriter (writer w);  
BufferedWriter bw = new BufferedWriter (writer w, int  
buffer size);

which of the following declarations are valid?

1. BufferedWriter bw = new BufferedWriter ("Ticket- //invalid: txt");
2. BufferedWriter bw = new BufferedWriter (newfile (txt));
3. BufferedWriter bw = new BufferedWriter (newfile ("Ticket.txt")); //valid
4. BufferedWriter bw = new BufferedWriter (new (newfile  
("Ticket.txt"))); //invalid  
("Ticket.txt")
5. It indicates 2 levels of buffering

## Methods

1. write (int ch);
2. writeline (String ch);
3. write (String s);
4. flush ();
5. close ();
6. newline ();

Inserting a new line character to the file

## Note

When compared with BufferedWriter which of the following capability (facility) is available in method (in BufferedWriter)

1. Writing data to the file;
2. Closing the writer;
3. flush of the writer;
4. Inserting new line character (newline ())

## Example

import java.io;

```
/* public void newline() throws IOException
Exception;
```

```
file writer file = new filewriter("file.txt", true);
bw.write("Hello");
bw.newLine();
char[] arr = {"平原", "北京", "上海"};
bw.write(arr);
bw.write("Hello");
bw.newLine();
bw.write("Good");
bw.flush(); // to make sure the operation is
// successful on file
bw.close(); // intentionally the close() call will
// happen
// when shutdown occurs
```

### Note.

1. bw.close() // recommended to use
2. bw.close(); // not recommended to use
3. bw.close(); // not recommended to use
 - bw.close()
4. When ever we are closing buffered writer
 automatically underlying writer will be closed

We are not close explicitly

## Buffered Reader

This is the most enhanced (better) Reader to read character data from the file.

Constructors

BufferedReader br = new BufferedReader(Reader)

BufferedReader br = new BufferedReader(Reader)

For buffering

### Note

→ Buffered Reader can not communicate directly with file. It should communicate via some Reader Object.

→ The main advantage of Buffered Reader over file Reader is we can read data line by line instead of character by character.

### Methods

1. int read()

2. int read(char[] ch)

3. String readLine()

it attempts to read next line and return it from file. If the next line is not available then this method returns null.

c. void close()

Read the data from the file called "table.txt".

```
public static long string readLine() throws IOException
```

```
import java.io.*;
```

```
public class TestApp
```

```
public static void main (String [] args) throws IOException
```

```
file Reader fr = new file Reader ("table.txt");
```

```
BufferedReader br = new BufferedReader (fr);
```

```
String line = null; }
```

```
System.out.println (line);
```

```
line = br.readLine();
```

```
br.close();
```

Note

1. br.close () // recommended to use

2. finalize () // not recommended to use

3. for `close()` it is recommended to use both  
after `close()`

→ whenever we use `Closing Buffered Reader`  
automatically underlying file reader will be  
closed it is not required to close explicitly  
→ Even this rule is applicable for `BufferedWriter`  
also

### PrintWriter

→ this is the most enhanced writer to write  
text data to the file  
→ By using `filenametext` and `bufferedwriter` we can  
write only character data to the file but by using  
PrintWriter  
we can write any type of data to the file

### Construction:

`PrintWriter pw = new PrintWriter (String name);`  
`PrintWriter pw = new PrintWriter (file f);`  
`PrintWriter pw = new PrintWriter (writer w);`  
`PrintWriter pw = new PrintWriter (writer w);`

`PrintWriter` can communicate either directly to  
the file or via some writer object also

## Methods

1. write (int ch);
2. write (char [] ch);
3. write (String s);
4. flush ();
5. close();
6. print (char ch);
7. print (int n);
8. print (double d);
9. print (boolean b);
10. print (String s);
11. println (char ch);
12. println (int n);
13. println (double d);
14. println (boolean b);
15. println (String s);

## Q1

import java.io.\*;

```
public void print (xxxx type)
public void print (xxxx type);
```

## Ques 2

public static void main (String [] args) throws IOException

File writer fw = new FileWriter ("file.txt");

PrintWriter out = new PrintWriter (fw);

out.write(100); // 100 unicode value will be written to a file.

out.write('w');

out.println(100); // 100 only will be written to the file

out.println(true);

out.println('s');

out.println("Dinesh");

out.flush();

out.close();

}

// JVM Shut down.

}

What is the difference b/w write(100) and print(100)?

- In the case of write(100) the corresponding character "d" will be added to the file but
- In the case of print(100), "100" value will be added directly to the file.

Notes:-

- a. The most enhanced Reader to read character data from the file is Buffered Reader.
- b. The most enhanced writer to write character.

data to the file is printed.

### Note:

In general we can use Readers and writers to handle character data whereas we can use

Input streams and output streams to handle binary data (like images, audio files, video files etc).

we can use output stream to write binary data to the file and we can use Input stream to read binary data from the file.

Character Data  $\Rightarrow$  Reader and writer

Binary Data  $\Rightarrow$  Input stream and output stream.

Requirement  $\Rightarrow$  file1.txt, file2.txt Copy all the

contents to file3.txt import java.io

contents to file3.txt

Class Test {

public static void main (String args) throws Exception

{

PrintWriter pw = new PrintWriter ("file3.txt")

// reading from first file and writing to file3.txt

BufferedReader br = new BufferedReader (new FileReader ("file1.txt"))

String line = br.readLine ()

while (line != null)

```
pu.println(line);
```

```
line = br.readLine();
```

```
}
// reading from second file and writing to first
br = new BufferedReader(new FileReader(Cfile2.txt));
```

```
line = br.readLine();
```

```
while ((line != null))
```

```
pu.println(line);
```

```
line = br.readLine();
```

```
}
// to write all the data to file3.txt
```

```
pu.println();
```

```
br.close();
```

```
pu.close();
```

```
System.out.println("Open file3.txt to see the
result");
```

```
}
// main function
```

Requirement :- file1.txt file2.txt copy one  
line from file1.txt and from file2.txt to file3.txt

```
import java.io.*;
class Test
{
 public static void main (String [] args) throws IOException
 {
 PrintWriter pw = new PrintWriter ("file2.txt");
 PrintWriter pw1 = new PrintWriter ("file1.txt");
 // reading from first file and writing to file2.txt
 BufferedReader br1 = new BufferedReader (new FileReader
 ("file1.txt"));
 String line1 = br1.readLine();
 BufferedReader br2 = new BufferedReader (new FileReader
 ("file2.txt"));
 String line2 = br2.readLine();
 while (line1 != null || line2 != null)
 {
 if (line1 == null)
 pw.println (line2);
 else
 pw.println (line1);
 line1 = br1.readLine();
 line2 = br2.readLine();
 }
 pw.println (line2);
 pw.print (line1);
 lines2 = br2.readLine();
 }
}
```

```
 //to write all the data to file 3.txt
 fw.flush();
 fw.close();
 fw = null;
 fw = new FileWriter("file3.txt");
 fw.write("Hello World");
 fw.flush();
 fw.close();
```

System.out.println("open file3.txt to see the result");

```
if (fw != null)
 fw.close();
System.out.println("File closed");
```

Requirement → Write a program to remove duplicate  
from the file.

Input: Data.txt

Class Test

```
{
 public static void main (String[] args) throws
 Exception
```

```
 BufferedReader br = new BufferedReader (new
 FileReader ("input.txt"))
 PrintWriter pw = new PrintWriter ("output.txt");
```

```
 String strout = br.readLine();
```

```
 while (strout != null)
```

```
 {
 boolean isAvailable = false;
```

buffered reader. BufferedReader bufferedReader = new BufferedReader(new FileReader("output.txt"));

String line = br.readLine();

// control comes out of while loop for smooth data  
without break

while (line != null)

{  
// if matched control should come out with

break;  
if (line.equals("target")) {

if Available = true)

break;

line = br.readLine();

if (Available == false) {

pw.println("target");

pw.flush();

target = br.readLine();

br.close();

pw.close();

Very Shallow now

Requirement => write a program to perform extraction of mobile no only if there is no duplicate.

import java.util.\*;

Class Test

```
public static void main (String[] args) throws
Exception
{
 BufferedReader br = new BufferedReader (new
 FileReader ("input.txt"));

 PrintWriter pw = new PrintWriter ("Output.txt");

 String target = br.readLine ();

 while (true != null)
 {
 boolean isAvailable = false;

 BufferedReader br1 = new BufferedReader (new
 FileReader ("Mobile.txt"));

 String line = br1.readLine ();

 // control comes out of while loop in smooth
 // fashion without break.
 while (line != null)
 {
 if (line.equals (target))
 {
 isAvailable = true;
 break;
 }
 line = br1.readLine ();
 }

 if (isAvailable)
 {
 pw.println (line);
 }
 }
}
```

if matched control should come out with break

if (line equals target))

if (available == true)

break;

else if (lno == ln) . read (line())

(if available == false) {

pw.println (target);

pw.flush ();

target = br.readLine ();

br.close ();

pw.close ();

if you shutdown here

with a code to read the data from the file and  
identify which data is longer in length (assuming  
the data is string)

import java.io.\*;

class Test {

public static void main (String [] args) throws

Exception {

BufferedReader br = new BufferedReader (new FileReader ("data.txt"));

String data = br.readLine();

int maxLength = 0;

String result = " ";

while (data != null)

{

int resultLength = data.length();

if (maxLength < resultLength)

maxLength = resultLength;

result = data;

data = br.readLine();

}

System.out.println ("The maximum string data in a file is :: " + result);

System.out.println ("The maximum string in a file is :: " + maxLength);

// JVM Shutdown hook

}

# Java

## Serialization

1. Serialization
2. Deserialization
3. Transient keyword
4. State vs transient
5. transient vs final
6. Object graph in serialization
7. Customized serialization
8. Serialization with respect to inheritance
9. Externalization
10. Difference b/w serialization & Externalization
11. Serialization ID

## Serialization

- The process of Saving (or) Writing State of an Object to a file is called Serialization but strictly speaking it is the process of Converting an Object from Java Supported form to either networkly Supported form (or) file Supported form.
- By Using file Output Stream and Object Output Stream Classes we can achieve Serialization process.  
    (=) `writeObject(Object obj)`

## De-serialization

- The process of reading State of an Object from file is called Deserialization but strictly speaking it is the process of Converting an Object from file

Supported form (in which supported form is)

Java: Supported form

→ By using `fileInputStream` and `ObjectInputStream` classes we can achieve `Serialization` (i.e. `readObject()`)

Import `java.io.*`

for

public `java.io.ObjectOutputStream` (Java To Output Stream)

throws

`java.io.IOException`

public `java.io.FileOutputStream` (Java, long String) throws

`java.io.FileNotFoundException`

public final void `writeObject` (java.lang.Object) throws  
java.io.IOException  
throws `java.io.IOException`

public `java.io.ObjectInputStream` (java.io.InputStream) throws  
`java.io.IOException`

public `java.io.FileInputStream` (java.lang.String) throws  
`java.io.FileNotFoundException`

public final `java.lang.Object readObject` () throws `java.io.IOException`,  
`java.lang.ClassNotFoundException`

24

class Dog, implements Serializable {

    Data:

        System.out.println("Static block gets executed.");

}

    Dog():

        System.out.println("Object created...");

    int i=10;

    int j=20;

class Test

    public static void main(String[] args) throws

        Exception:

    Dog d = new Dog();

    System.out.println("Serialization started...");

    String filename = "abc.dat";

    fileOutputStream fos = new FileOutputStream(filename);

    objectOutputStream oos = new ObjectOutputStream(fos);

    oos.writeObject(d);

    System.out.println("SerializationObject written to file");

    System.out.println("Serialization ended...");

// To pause the execution till we press some key from keyboard. System.in.read()

```
System.out.println("Deserialization started ---");
System.out.println("Deserialized Object reference is");
fileInputStream fis = new FileInputStream("abc.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
Object obj = ois.readObject();
Dog dh = (Dog) obj;
System.out.println("Deserialized Object Reference is");
System.out.println("Deserialized ended ---");
}
// JVM shutdown now.
```

---

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;
class Dog implements Serializable {
 int i=10;
 int j=20;
```

class and implements Serializable  
true  $\Rightarrow$  true  
but  $\Rightarrow$  false

```
public class TestApp {
 public static void main (String [] args) throws
 IOException, ClassNotFoundException {
 Dog d1 = new Dog ();
 Cat c1 = new Cat ();
 System.out.println ("Serialization Started");
 FileOutputStream fos = new FileOutputStream ("abc.ser");
 ObjectOutputStream oos = new ObjectOutputStream (fos);
 oos.writeObject (d1);
 oos.writeObject (c1);
 System.out.println ("Serialization Ended");
 System.out.println ("Deserialization Started");
 FileInputStream fis = new FileInputStream ("abc.ser");
 ObjectInputStream ois = new ObjectInputStream (fis);
 Dog d2 = (Dog) ois.readObject ();
 Cat c2 = (Cat) ois.readObject ();
 System.out.println ("Deserialization Ended");
```

```
System.out.println("Cat object data")
```

```
System.out.println("dr. id=" + id))
```

```
System.out.println("Cat object data")
```

```
System.out.println("dr. id=" + id))
```

### Output

```
Serialization started
```

```
Serialization ended
```

```
Deserialization started
```

```
Deserialization ended
```

```
Obj object data
```

```
20
```

```
Cat object data
```

```
100 200
```

### Note:

① we can perform serialization only on Serializable objects.

② an object is serializable if and only if the corresponding class implements Serializable interface.

③ Serializable interface present in java.io package and does not contain any abstract methods. It is an empty interface.

- ④ Serializable Interface present in Java to package and does not contain any abstract methods. If we implement this interface the required ability will be provided automatically by JVM.
- ⑤ We can add any no. of Objects to the file and we can read all those objects from the file but in which order we wrote objects in the same order only. The objects will come back that is Order is important.
- ⑥ If we are trying to serialize a non-serializable object then we will get runtime exception saying "NotSerializableException".

### Transient Key word:

- ① **transient** is the modifier applicable only for members, but not for classes and methods.
- ② While performing serialization if we don't want to save the value of a particular variable to meet security constraint such type of variable, then we should declare that variable with **transient** keyword.
- ③ At the time of serialization JVM ignores the original value of transient variable and save default value to the file.
- ④ That is "transient means not to serialize".

```
eg:-1 import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.Serializable;
```

```
class Dog implements Serializable {
```

```
 int i=10;
```

```
 transient int j=20;
```

```
public class TestApp {
```

```
 public static void main (String [] args) throws
```

```
 IOException, ClassNotFoundException {
```

```
 Dog d = new Dog();
```

```
 System.out.println ("Serialization Started");
```

```
 FileOutputStream fos = new FileOutputStream ("obj.ser");
```

```
 ObjectOutputStream oos = new ObjectOutputStream (fos);
```

```
 oos.writeObject (d);
```

```
 System.out.println ("Serialization ended");
```

```
System.out.println("Deserialization Started");
FileInputStream fis=new FileInputStream("abc.dat");
ObjectInputStream ois=new ObjectInputStream(fis);
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog) ois.readObject();
```

```
System.out.println("Dog Object data");
System.out.println(d2.toString());
```

## Output

```
Serialization Started
```

```
Serialization ended
```

```
Deserialization Started
```

```
Deserialization ended
```

```
Dog object data
```

```
to
```

## Static vs transient

1. Static variable to not part of object static hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

```
import java.io.FileOutputStream;
```

```
import java.io.IOException;
```

```
import java.io.ObjectOutputStream;
```

```
import java.io.FileInputStream;
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
class Dog implements Serializable {
 static transient int i=10;
 int j=20;
}
public class TestApp {
 public static void main (String [] args) throws
 IOException, ClassNotFoundException {
 Dog d1 = new Dog();
 System.out.println ("Serialization Started");
 FileOutputStream fbs = new FileOutputStream ("data");
 ObjectOutputStream oos = new ObjectOutputStream (fbs);
 oos.writeObject (d1);
 System.out.println ("Serialization ended");
 System.out.println ("Deserialization Started");
 FileInputStream fis = new FileInputStream ("data");
 ObjectInputStream ois = new ObjectInputStream (fis);
 Dog d2 = (Dog) ois.readObject ();
 System.out.println ("Deserialization ended");
 System.out.println ("Dog Object data");
 System.out.println (d2.i + " " + d2.j);
```

~~Output~~

Initialization started.  
Initialization ended  
Initialization started  
Initialization ended  
Finalization ended  
Object data  
0 0

Transient vs final:

final Variable will be participated into Serialization  
directly by their values.  
Hence declaring original Variable as transient there  
is no use.  
If the compiler comes the value to final variable  
eg: final int x=10;  
int y=20;

System.out.println(x); compiler will replace this  
as System.out.println(20) because x is final.  
System.out.println(y);

```
import java.io.FileOutputStream
import java.io.IOException
import java.io.ObjectOutputStream
import java.io.FileInputStream
import java.io.ObjectInputStream
import java.io.Serializable;
```

Class Dog implements Serializable

int i=10;

Transient final int j=20;

public class TestApp {

public static void main (String[] args) {

try {  
ClassNotFoundException e = new ClassNotFoundException("Dog");  
throw e;

Dog d1 = new Dog();

System.out.println ("Serialization Started");

FileOutputStream fos = new FileOutputStream("abc.dat");

ObjectOutputStream oos = new ObjectOutputStream(fos);  
oos.writeObject(d1);

System.out.println ("Serialization ended");

System.out.println ("Deserialization Started");

FileInputStream fis = new FileInputStream("abc.dat");

ObjectInputStream ois = new ObjectInputStream(fis);

Dog d2 = (Dog) ois.readObject();

System.out.println ("Dog Object data");

System.out.println ("d2.i=" + d2.i);

## Output

Serialization started

Serialization ended

Deserialization started

Deserialization ended

Des. object class

0 10

## Deserialization output

①

int i=10;

int j=20;

Output

0 10 20

②

transient int i=10;

int j=20;

Output

0 10 20

③

transient int i=10;

transient static int j=20;

Output

0 10 20

④

transient final int i=10;

transient int j=20;

Output

0

⑤ transient final int i=10;

transient static int j=10;

Output

10.....10

Note

We can serialize any no of objects to the file but in which order we serialized in the file only we have to de-serialize in the same order only we have to de-serialize, if we change the order then it would result in "Class Cast Exception".

Example =

```
Dog d1 = new Dog();
```

```
Cat c1 = new Cat();
```

```
Dot r1 = new Dot();
```

```
fileOutputStream fos = new FileOutputStream("abc.ser");
```

```
objectOutputStream oos = new ObjectOutputStream(fos);
```

```
oos.writeObject(d1);
```

```
oos.writeObject(c1);
```

```
oos.writeObject(r1);
```

```
fileInputStream fis = new FileInputStream("abc.ser");
```

```
objectInputStream ois = new ObjectInputStream(fis);
```

```
Dog d2 = (Dog) ois.readObject();
```

```
Cat c2 = (Cat) ois.readObject();
```

```
Dot r2 = (Dot) ois.readObject();
```

→ If we don't know the order of serialization then we need to use the following code

```

fileInputStream = new FileInputStream("abc.txt");
ObjectInputStream obj = new ObjectInputStream(inputStream);
Object obj = obj.readObject();
if (obj instanceof Dog) {
 Dog d = (Dog) obj;
 // perform operation related to Dog
}
if (obj instanceof Cat) {
 Cat c = (Cat) obj;
 // perform operation related to Cat
}

```

### Object graph in serialization

Whenever we are serializing an object, the other objects are also serialized. If one object has a reference to another object, then that object will be serialized automatically. The group of objects is nothing but object graph in serialization.

③ In object graph every object should be serializable otherwise we will get runtime exception saying "Not Serializable Exception".

```
import java.io.Serializable;
```

```
import java.io.FileOutputStream;
```

```
import java.io.ObjectOutputStream;
```

```
import java.io.FileInputStream;
```

```
import java.io.ObjectInputStream;
```

```
import java.io.IOException;
```

```
class Dog implements Serializable
```

```
 Cat reviewCat();
```

```
class Cat implements Serializable
```

```
 Dog reviewDog();
```

```
class Test implements Serializable
```

```
 int i=0;
```

```
public class Test
```

```
 public static void main (String [] args) throws
```

```
 IOException, ClassNotFoundException
```

```
 Dog d= new Dog();
```

```
System.out.println ("Serialization Started");
```

```
FileOutputStream fos = new FileOutputStream ("file100");
```

```
ObjectOutputStream oos = new ObjectOutputStream (fos);
```

```
oos.writeObject (d);
```

```
System.out.println ("Serialization ended");
```

```
System.out.println ("Is main thread running now?");
```

```
System.out.println("Deserialization Started");
fileInputStream file = new FileInputStream("a.java");
ObjectInputStream ois = new ObjectInputStream(file);
String s = (String) ois.readObject();
System.out.println(s);
System.out.println("Deserialization Ended");
```

## Output

Serialization Started

Serialization ended

java.lang.String@40000000

Deserialization Started

10

Deserialization ended

## Customized Serialization

During default Serialization there may be a chance loss of information due to transient keyword.

Example : remember strings and arrays inside it.

```
import java.io.Serializable;
```

```
import java.io.FileOutputStream;
```

```
import java.io.ObjectOutputStream;
```

```
import java.io.FileInputStream;
```

```
import java.io.IOException;
```

Properties of objectInputStream

class account implements serializable

String name = "sachin";

transient String password = "password";

3

private class Test3

public static void main (String [] args) throws

IOException, ClassNotFoundException

account acc = new Account();

System.out.println (acc.getName());

System.out.println ("Deserialization Started");

fileInputStream fil = new FileInputStream ("file1.txt");

ObjectInputStream ois = new ObjectInputStream (fil);

acc = (account) ois.readObject();

System.out.println ("Name is " + acc.getName());

System.out.println ("Deserialization Ended");

fileInputStream fil = new FileInputStream ("file2.txt");

ObjectInputStream ois = new ObjectInputStream (fil);

acc = (account) ois.readObject();

System.out.println ("Name is " + acc.getName());

System.out.println ("Deserialization ended");

In the above example before Serialization account object can provide proper nickname and password but after Deserialization Account object can provide only nickname but not password this is due to denoting password as transient.

Hence doing default serialization there may be chance of loss of information due to transient keyword.

We can recover this loss of information by using customized serialization by writing our own implements Customized Serialization by using the following two methods.

① private void writeObject ( ObjectOutputStream os )

throws Exception

→ this method will be executed automatically by JVM at the time of serialization.

→ Hence at the time of serialization if we want to perform any extra work we have to do is that in this method only - ( prepare encrypted password and write encrypted password separately to the file )

② private void readObject ( ObjectInputStream is )

throws Exception

→ This method will be triggered automatically by JVM at the time of Deserialization.

Hence at the time of Deserialization if we want the right

Only Extra Activity:

We have to define that in this method only  
(read encrypted password, perform decryption  
and assign decrypted password to the Account  
Object password variable.)

```
yes! import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
```

```
Class Account implements Serializable {
 String name = "Sachin";
 transient String password = "tmttkk123";
 private void writeObject(ObjectOutputStream os) throws IOException {
 os.defaultWriteObject(); // performing default
 String encr = DES.encrypt(password); // performing encryption
 os.writeObject(encr); // write the encrypted data to
 // file(abc.ser)
 }
 private void readObject(ObjectInputStream os) throws IOException {
```

01's - default ReadObject(), // performing de-serialization

String expod = casting of readObject(), // performing  
decompression

password expod - extracting (S) // extracting the extra data  
in Object

```
public class Test {
 public static void main (String [] args) throws
 IOException, ClassNotFoundException {
 Account acc = new Account ();
 System.out.println (acc.getName () + " " + acc
 .getPassword());
 System.out.println ("Serialization Started");
 FileOutputStream fos = new FileOutputStream ("obj");
 ObjectOutputStream oos = new ObjectOutputStream (fos);
 oos.writeObject (acc);
 System.out.println ("Serialization ended");
 System.out.println ("Now it is time to de-serialize");
 System.out.println ("Deserialization Started");
 FileInputStream fis = new FileInputStream ("obj");
 ObjectInputStream ois = new ObjectInputStream (fis);
 acc = (Account) ois.readObject();
 }
}
```

System.out.println("Acc. Name "+ acc.getName());

(polymorphism)

System.out.println("Deserialization mode");

- At the time of Account object Serialization, JVM will check if there any writeObject() method in Account class or not.
- If it is not available then JVM is responsible to perform serialization (default serialization).
- If account class contains writeObject() method then JVM feels very happy and executes that Account class writeObject() method.
- The same rule is applicable for readObject() method also.

Import java.io

```
/* public java.io.ObjectOutputStream(java.io.ObjectOutputStream)
```

Java To Java

```
public java.io.FileOutputStream(java.io.FileOutputStream) throws
java.io.IOException
```

```
public final void writeObject(java.lang.Object)
```

throws java.io.Deregester

public void noFileInputStreamException() throws IOException {  
 String line = "Hello World";  
 byte[] bytes = line.getBytes();  
 ByteArrayInputStream bis = new ByteArrayInputStream(bytes);  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 ois.readObject();  
}

throws java.io.IOException

public void fileNotFoundException() throws IOException {  
 String line = "Hello World";  
 byte[] bytes = line.getBytes();  
 ByteArrayInputStream bis = new ByteArrayInputStream(bytes);  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 ois.readObject();  
}

throws java.io.IOException

public void longObjectReadObject() throws IOException {  
 String line = "Hello World";  
 byte[] bytes = line.getBytes();  
 ByteArrayInputStream bis = new ByteArrayInputStream(bytes);  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 ois.readObject();  
}

throws java.io.IOException

public void longClassNotFoundException() throws IOException {  
 String line = "Hello World";  
 byte[] bytes = line.getBytes();  
 ByteArrayInputStream bis = new ByteArrayInputStream(bytes);  
 ObjectInputStream ois = new ObjectInputStream(bis);  
 ois.readObject();  
}

throws java.io.IOException

class Account implements Serializable {  
 String loginName = "sa@sa.com";  
 String password = "1234567890"; // loss of  
 transient String pin = "1234567890"; // loss of  
 // information  
 transient int pin = 0; // loss of information.  
 // write logic of serialization  
 private void writeObject(ObjectOutputStream oos) throws IOException {  
 System.out.println("writeObject method is called");  
 // performing default serialization  
 oos.defaultWriteObject();  
 // performing encryption on password  
 String encrypted = "1234567890"; // password // randomizing  
 // first encryption with pin // best  
 // write the encrypted data as object-to-serialized file  
 oos.writeObject(encrypted);  
 oos.writeObject(pin);  
 }  
}

```
// write the logic of Deserialization
private void readObject (ObjectInputStream ois)
throws Exception {
System.out.println ("readObject method recall");
```

```
// perform default deserialization
ois.defaultReadObject();
```

```
// read encrypted data from Serialized file
// read encrypted data from Serialized file
String encrypted = (String) ois.readObject();
String original = (String) ois.readObject();
int encPin = ois.readInt();
```

```
// perform decryption and calculate it to instance
// of EncryptedSubString
EncryptedSubString password = EncryptedSubString.getInstance (original);
password = EncryptedSubString.getInstance (original);
pin = EncryptedSubString.getInstance (original);
```

```
class Test {
public static void main (String [] args) throws
Exception {
```

```
Account account = new Account ();
System.out.println ("Serialization Started ----");
String filename = "File.ser";
FileOutputStream fos = new FileOutputStream (filename);
```

Object Output Stream on ~~new ObjectOutputStream(fis)~~;

acc.writeObject(account);

System.out.println("Serialization ended ---");

// to pause the execution till we press some key from keyboard  
System.in.read();

System.out.println("De-Serialization Started ---");

fileInputStream fis = new ~~ObjectInputStream~~(new FileInputStream("abc.ser"));

ObjectInputStream ois = new ObjectInputStream(fis);

Account acc = (Account) ois.readObject();

System.out.println("Username is : " + acc.getUsername());

System.out.println("password is : " + acc.getPassword());

System.out.println("acc is : " + acc);

System.out.println("De-Serialization ended ---");

Given Shutdown now

Output

D:\100operations>java C test.java

D:\100operations>java test

Serialization Started

write Object method is called

Serialization ended -

Be-Serializat<sup>ion</sup> Started

Default method is called

username is :- Sachin

password is :- franklin

bio :- isingle

Be-Serializat<sup>ion</sup> ended

Java

## Serialization with inheritance

Case 1 :-

If parent class implements Serializable then automatically every child class by default implements Serializable that is Serializable nature is inheriting from parent to child.

Hence even though child class doesn't implement Serializable, we can serialize child class object.

If parent class implements

Serializable interface

Import java.io.Serializable;

Import java.io.FileOutputStream;

Import java.io.ObjectOutputStream;

Import java.io.FileInputStream;

Import java.io.ObjectInputStream;

Import java.io.IOException;

class Animal implements Serializable {

    int i = 0;

}  
class Dog extends Animal {

    int j = 20;

}

public class Test {  
    public static void main (String args) throws  
        IOException, ClassNotFoundException {

        Dog d = new Dog ();

        System.out.println ("Serialization started");  
        FileOutputStream fos = new FileOutputStream ("obj.ser");  
        ObjectOutputStream oos = new ObjectOutputStream (fos);  
        oos.writeObject (d);  
        oos.close ();

        System.out.println ("Deserialization ended");

        System.out.println ("Press any key to exit");

        System.out.println ("Deserialization started");

        FileInputStream fis = new FileInputStream ("obj.ser");  
        ObjectInputStream ois = new ObjectInputStream (fis);  
        Dog d1 = (Dog) ois.readObject ();

        System.out.println (d1.i == d.i);  
        System.out.println ("Deserialization ended");

}

## Output

Serialization started

Serialization ended

Deserialization started

Deserialization ended

Even though Dog class does not implements Serializable interface explicitly but we can serialize Dog object because its parent class Animal already implements Serializable interface.  
Note: Object class doesn't implement Serializable interface.

## Case 2.

- Even though parent class does not implement Serializable we can serialize child object if child class implements Serializable interface.
- At the time of Serialization JVM together with the values of instance variable which are coming from non-Serializable parent then instead of original value JVM saves default value for those variables to the file.
- At the time of Deserialization JVM checks whether any parent class is non Serializable or not.

- ④ Only parent class is non-serializable and creates a separate object for every non-serializable parent and shares the instance variable to the parent object.
- ⑤ To create an object for non-serializable parent you always call no arg constructor (default constructor) of that non-serializable parent hence every non-serializable parent should compulsorily contain no arg constructor otherwise we will get runtime exception.
- Invalid class exception.

import java.io.Serializable;

For part 1: `java -io fileoutputstream`

import java.io.Objectoutputstream;

import java.io.FileInputStream;

import java.io.ObjectInputStream;

import java.io.IOException;

import java.io.Serializable;

class Animal {

int r=10;

Animal() {

System.out.println("No arg Animal constructor");

}

class Dog extends Animal implements Serializable {

int s=20;

Dog d=new Dog(); System.out.println("The Dog constructor");

```
public class Test {
 public static void main (String [] args) {
 throw new
 }
}
```

Exception: classNotFound Exception

Dog d = new Dog () ;

d. i = 1;

new d. i = 999;

```
System.out.println ("serializable started");
```

```
fileOutputStream fos = new FileOutputStream ("abc");
ObjectOutputStream oos = new ObjectOutputStream (fos);
oos. writeObject (d);
System.out.println ("Serialization ended");
```

```
System.out.println ("work in progress");
```

```
System.out.println ("Deserialization started");
fileInputStream fis = new FileInputStream ("abc");
ObjectInputStream ois = new ObjectInputStream (fis);

```

dog d = (Dog) ois. readObject ();

```
System.out.println (d. i == 1);
```

```
System.out.println ("Deserialization ended");
```

↑

Output

No. of animal constructed

No. of big bag constructed

Serialisation Started

Serialisation ended

Deserialisation Started

No. of animal constructed

10 ==> 997

Deserialisation ended

## Agenda

1. Externalization
2. Difference b/w Serialisation & Externalization
3. Serial Version of ID

Externalization :- (Java) and its working

1. In default Serialisation every thing taken care by JVM and programmer doesn't have any control
2. In Serialisation total object will be saved always and it is not possible to save point of the object, which creates performance problems at certain point
3. To overcome these problems we should go for Externalization where every thing taken care by programmer and JVM doesn't have any control.

24. The main advantage of Externalization are  
Serialisation process can save either whole object  
or part of the object based on our requirement

- ① To provide Externalizable ability any  
object comprising the corresponding class should  
implements Externalizable interface
- ② Externalizable interface is child interface of  
Serializable interface

Externalizable interface defines 2 methods -

1. writeExternal (ObjectOutput out) throws IOException

2. readExternal (ObjectInput in) throws IOException

Class Not found Exception

public void writeExternal (ObjectOutput out)  
throws IOException

- this method will be executed automatically  
at the time of serialisation with in this  
method , we have appropriate code to save required  
variables to the file . . . .

- public void readExternal (ObjectInput in) throws  
IOException , Class Not Found Exception

this method will be executed automatically at  
the time of deserialisation with in this method  
we have appropriate code to save read required  
variables from file and assign to the current  
object . . . .

at the time of deserialization from will create separate new object by creating publicly  
that object from will call readExternal() method.  
Any Externalizable class should implement interface  
public no-arg constructor otherwise we will get  
Runtime Exception saying "InvalidClassException".

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;
import java.io.Externalizable;
import java.io.ObjectOutput;
import java.io.ObjectInput;
```

class ExternalizableDemo implements Externalizable {

String s;

int i;

int k;

ExternalizableDemo(String s, int i, int k) {

this.s = s;

this.i = i;

this.k = k;

public Externalizable demo1

System.out.println("System.out.println from any constructor")

if performing serialization as per our requirement

public void readExternal(ObjectInput in) throws

IOException

System.out.println("Call back method used with")

Serialization")

ObjectInputStream in;

in.readObject();

if performing Deserialization as per our requirement

public void readExternal(ObjectInput in) throws

IOException, ClassNotFoundException {

System.out.println("Call back method used")

by serializable class")

String str = (String) in.readObject();

System.out.println(str);

public class Test {

public static void main (String [] args)

throws IOException, ClassNotFoundException {

ExternalizableDemo d = new ExternalizableDemo("John", 100, 200);

System.out.println("Serialisation Started");

```
FileOutputStream fos = new FileOutputStream("obj.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(d);
oos.close();
```

System.out.println("Serialisation ended");

System.out.println("Deserialisation Started");

```
FileInputStream fis = new FileInputStream("obj.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
d = (ExternalizableDemo) ois.readObject();
ois.close();
```

System.out.println("d is " + d + " and " + d);

System.out.println("Deserialisation ends");

## Output

Serialisation Started

ExternalizableDemo method used while serialisation

Serialisation ended

## Deserialization Started

### Desig. Consideration

Class based method used to do Deserialization

Within  $\approx 100 \rightarrow 0$

### Deserialization ended

① If the Class implements Serializable then  
then only part of the Object will be saved

In the case output is

② public no arg constructor

Within  $\approx 10 \rightarrow 0$

### Deserialization transient

If the Class implements Serializable Interface  
then the output is within  $\approx 10 \rightarrow 20$

③ In deserialization transient keyword won't  
play any role, hence transient keyword not  
requirement :-

### ④ Different b/w Serialization and Externalization

## Serialization

It is used for default serialization.

- ④ Here everything takes care by JVM and  
programmer doesn't have any control doesn't have  
any control
- ⑤ The total object will be saved always and  
it is not possible to save part of the object
- ⑥ Serialization is the best choice if we want  
to save total object to the file
- ⑦ Relatively performance is low.  
Serialization interface doesn't contain any method
- ⑧ It is a marker interface
- ⑨ Serializable class must implement public  
no-arg constructor
- ⑩ transient keyword play role in serialization  
by ignoring.

### SerialVersionUID

- In addition to serialization & deserialization automatically  
JVM will use a unique identifier which is working  
but Serializable.
- At the time of serialization JVM will save  
SerialVersionUID with object.
- At the time of Deserialization JVM will compare  
SerialVersionUID and if it is matched then it will accept

will be

Deserialized. Otherwise we will get Runtime Exception  
String

In Malformed Class Exception

the process is depending on default serialization  
code

1. After serializing object if we change the class  
file then we can't perform deserialization because  
of mismatch in serialization of local class and  
serialized object in this case at the time of  
deserialization we will get

Runtime Exception saying "Malformed Class Exception"

2. Both sender and receiver should use the same  
version of JVM if there any incompatibility in JVM  
version then receiver unable to deserializing because  
of different serialization. In this case  
receiver will get Runtime Exception saying "Invalid  
Class Exception"

③ To generate SerialVersionUID informally JVM will  
use timestamp with which many it wants to perform  
operations

Q. Can solve above problems by introducing own  
class Enumeration?

Ans: Import `java.io.Serializable`  
public class Dog implements Serializable  
private static final long serialVersionUID = 0L;  
int i=10;  
int j=20;

Import `java.io.*`  
public class Sender {  
public static void main (String args) throws  
IOException  
Dog d = new Dog();

FileOutputStream dos = new FileOutputStream ("abc.txt");  
ObjectOutputStream os = new ObjectOutputStream (dos);  
os.writeObject (d);

Import `java.io.*`  
public class Receiver {  
public static void main (String args) throws  
IOException, ClassNotFoundException {  
FileInputStream dos = new FileInputStream ("abc.txt");  
ObjectInputStream os = new ObjectInputStream (dos);  
Dog d = (Dog) os.readObject();

```
FileInputStream fileInputStream = new FileInputStream("file.txt");
ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
Dog d2 = (Dog) objectInputStream.readObject();
System.out.println(d2); // => "Hulk"
```

b: NTestApp>javac Dog.java

b: NTestApp>java & random

b: NTestApp>javac Dog.java

b: NTestApp>java RandomApp

Output => 20

⇒ In the above program option **Serializable** is not used although if we perform any change to Dog class even then we can serialize object

⇒ but we can configure our own Serializable Version

both Sender and Receiver don't required to maintain the same JVM Versions

Note: Some IDE's generates explicit

**Serializable**

## Usage of StringTokenizer

It is a part of java.util package.

It is used to split the entire string into multiple tokens based on the delimiter we supply.

e.g. String data = "sachin ramesh techukar";

StringTokenizer str = new StringTokenizer(data);

StringTokenizer str = new StringTokenizer(data, " "));

public boolean hasMoreTokens()

public String nextToken()

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

```
class TestAPP {
```

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

```
 StringTokenizer str = new StringTokenizer()
```

```
 "Sachin ramesh techukar", " "));
```

```
 System.out.println(str);
```

```
 int tokenCount = str.countTokens();
```

```
 System.out.println(tokenCount);
```

```
 while (str.hasMoreTokens())
```

```
 {
```

```
 String data = str.nextToken();
```

```
 System.out.println(data);
```

Code

One Java Completed.