CORE JAVA

NEW NOTES

KV.RAO

SATYA TECHNOLOGIES

MANOJ ENTERPISES XEROX

All software institute materials, Spiral – Binding, Printouts available

CONTACT: 9542556141

INDEX

S.NO	TITLE	P.NO
1.	Defination of software & types of softwares	1
2.	Standalone applications & distributed appl^ns	4
3.	History of Java	5
4.	Features of Java	8
5.	Data types	16
6.	Java fundamental datatypes	18
	i, Integer category datatypes	18
	ii, Float " "	19
	iii, Character " "	21
	iv, Boolean " "	23
7.	Variables	24
	i, Declaration	25
	ii, Initialization	26
8.	Methods	29
	i, Method overloading	34
	ii, Method overridding	35
9.	OOPS features	36
	a, Class	39
	b, Object	43
10.	Structure of Java program	51
11.	Method chaining	57
12.	Accepting values from command prompt	65

S.NO	TITLE	P.NO
13.	constructors in Java	86
	i, Types of constructors	88
	ii, Defaeult constructors	88
	iii, parameterized constructors	90
	iv, overloaded constructors	92
	v, Object parameterized constructors	92
14.	Data Encapsulation & Data Abstraction	93
15.	Static blocks in Java	99
16.	Instance blocks in Java	106
17.	Reading data from Keyboard	111
18.	"THIS" Keyword	113
19.	Factory Method	119
20.	Inheritance	123
21.	Types of relationships in Java	135
	a, is-A, b, has-A, c, Uses-A	
22.	"Super" Keyword	138
23.	Polymorphism	155
24.	Dynamic Binding	157
25.	Types of Classes	
	a, concrete classes	161
	b, Abstract classes	163
	c, Null body method	176

S.NO	TITLE	P.NO
26.	Interfaces	179
	i, Inner/Nested interfaces	195
	ii, Functional interfaces	201
	iii, Marker/Tagged interfaces	204
27.	Packages	206
28.	Access specifiers	222
29.	Exception Handling	226
	i, try, catch & finally blocks	237
	ii, "throws" keyword	245
	iii, "throw" keyword	254
30.	Multi threading	270
	i, steps of a thread	274
	ii, creating a thread	276
	iii, By using Java.lang.thread class	276
	iv, By using Java.lang. Runnable	290
31.	Synchronization	298
32.	Inter thread communication	307
33.	I-D CFW process	320
34.	Object type Casting	320
35.	I-D CFW interfaces	
	i, Method in Java.util.Collection	323
	ii, " " Java.util.List	327
	iii, " " Java.util.Set	330
	iv, " " Java.util.SortedSet	330

S.NO	TITLE	P.NO
36.	1-D CFW classes	
	i, LinkedList	333
	ii, ArrayList	343
	iii, HashSet & TreeSet	346
37.	2-D CFW process	350
38.	2-D CFW interfaces	
	i, Java.util.Map	352
	ii, Java.util.Map.Entry	356
	iii, Java.util.SortedMap	357
39.	2-D CFW classes	358
	i, HashMap & TreeMap	359
40.	Legacy CFW	
	i, Enumeration	367
	ii, Vector	368
	iii, Stack	369
	iv, Dictionary	373
	v, HashTable	375
	vi, property	376
41.	File programming	377
42	Serialization & De-Serialization	380
43.	String Handling	396
	i, By using Java.lang.String	410
	ii, By using Java.lang.StringBuffer	411
	iii, By using Java.lang.StringBuilder	421
	iv, By using Java.lang.StringTokenizer	426
		426

1/7/15

Def^n of S/w :-

A s/w is a collection of programs meant for performing same operations.

In real world we have so many no. of operations which are performed by many no. of s/w's for the simplicity of upcoming programmers operations & s/w's classified into 3 types. They are

1. System s/w
2. Application s/w
3. Internet s/w

2/7/15

System software :-

In real world we've 3 projects which are comes under s/s s/w development.

1. Development of device drivers (s/w for printer, scanner, tablet, digital pad etc)
2. Development of language compilers, interpreters etc
3. Development of RTOS (Windows OS's, UNIX, LINUX, Solaries, MAC etc...)

Inorder to develop the above projects which are classified under s/s s/w we use the following languages.

→ C, 8086

→ The above languages are popular for development of s/s s/w projects becoz these languages contains many no. of features which are useful to interact with H/w devices.

Application s/w :-

The purpose of appl s/w is to develop the project/appl to the organizations.

Some of the organizations are

* Financial Sector (Banking, Insurance etc)
* Health care sector (Hospitality, Diagnostic etc)
* Transportation sector (Railway, Airlines etc)
* Educational sector (Schools, Colleges etc) etc

Inorder to develop an appl s/w for an organization we use the following technologies. They are

1. Front End Technologies
2. Back End Technologies

Front End Technologies:-

The purpose of front end Technologies is that to develop GUI appls (look & feel appls).

In real world we've 2 types of front end techns. They are

(a) Platform dependent front end technologies :-

Ex:- Developer 2000 (oracle corp)

Visual Basic (Microsoft)

If we develop any GUI appl with the above 2 techns then they runs on Microsoft provided OS's only but not able to run by default on Unix. These techns are called "PDFET"

(b) Platform Independent front end technologies :-

Ex:- AWT, Applets, swings } Java Concept, — SUN micro S/S

AWT - Abstract window toolkit.

If we develop any GUI application with the above concepts then such GUI appl'ns runs on each & every OS. Hence, these techn'ls are called "PIDFET".

2. Back End Technologies :-

The purpose of back end techn'ls is that to achieve data persistancy (storing the data permanently).

We can achive the data persistency in 2 ways in world. They are

a, By using files

b, By using data base s/w's

If we store the data permanently in the form of files then any unauthorized user can modify the data of the file becoz files concept of any language doesn't contain any security in the form of user name & password. Hence, files are unsecured & industry not recomended to use.

If we store the data permanently in the form of data base s/w's then unauthorized users can't modify the data of data base becoz most of the data base s/w products provides high security in the form of

4 The following table gives data base s/w name & data base s/w vendor name.

DB s/w Name	DB s/w Vendor Name
Oracle	Oracle Corp
SQL server	Microsoft
DB2	IBM
⋮	⋮

8/7

Q:- Define stand alone appl^n & distributed appl^n?

Ans:- A stand alone appl^n ~~one~~ runs in the context of local disc & whose results can be accessable in the same machine but not accessable across the universe.

Ex:- All the projects of S/S s/w & Appl^n s/w.

Distributed Appls runs in the context of browser/ WWW & whose results are sharable accross the universe.

Ex:- 1. web sites (Yahoo, Gmail etc)

2. ATM based appls

3. Credit card based appls

4. Net banking appls

5. e-commerce related appls

Internet S/w :-

The purpose of Internet S/w is to develop distributed appls.

To develop distributed appls we use the following languages/technologies.

→ Java developed at SUN Micro system INC, USA (Handed over by oracle corp)

→ .Net developed at Microsoft INC, USA

Hence, Java is one of the internet S/w developed at SUN Micro system, released to the real world, used by all the programmers for developing each & every appl" & Java is more popular for developing distributed appls.

History of Java :-

→ Java is one of the programming language (or) technology treated by the programmers as "Internet S/w". & it is used for developing each & every appl" & it is more popular for developing distributed appls.

→ the Java S/w developed at SUN Micro System INC, USA (handed over by oracle corp) & developed under the guidence of James Gosley (Father/creator of Java).

→ The Java S/w started its development in the early of the year 1990 & released to the real industry in the middle of the year 1991 on the name of "OAK".

6 which is the original name of Java & scientifically it is tree name & it has taken 18 months to develop.

→ In those days the OAK S/w was popular for developing device level S/w's only (Electronic, mechanical & Automobile devices) but not popular for developing appls related to Internet S/w & appln S/w. This is the industry experts opinion. (1991)

→ To fullfil all the requirements of the industry SUN Micro System developers has perform them re-engineering operation on OAK S/w & released to the real world on the name of Java in the year 1995. Scientifically Java is the one of the Coffee seed name. At present the Java S/w is available in 3 categories. They are

a, J2SE (Java 2 Standard Edition)

b, J2EE (Java 2 Enterprise Edition)

c, J2ME (Java2 Micro/mobile Edition)

→ In the industry J2SE is used for developing client side appls, J2EE is used for developing server side appls & J2ME is used for developing wireless appls.

+17 Versions in Java:-

The following table gives Java techrology version & Java language version.

Java tech" version	Java language version
JAVA	JDK 1.0 JDK 1.1
JAVA2	JDK 1.2 JDK 1.3
	JDK 1.4
JAVA5	JDK 1.5 (Tiger s/w)
JAVA6	JDK 1.6
JAVA7	JDK 1.7
JAVA8	JDK 1.8

In the present industry we are using JDK 1.5 (or) 1.6.

Download JDK 1.5/1.6/1.7/1.8 freely from www.sun.com (or) www.oracle.com

NOTE:- In the present industry J2SE must be called as JSE (or) Java SE, J2EE must be called as JEE (or) Java EE and J2ME must be called as JME (or) Java ME.

Features / Buzzwords of Java :-

Features are ntng but the services/facilities/functionalities provided by language developers in the language which are used by programmers for developing real time appns.

Java programming contains 13 features. They are

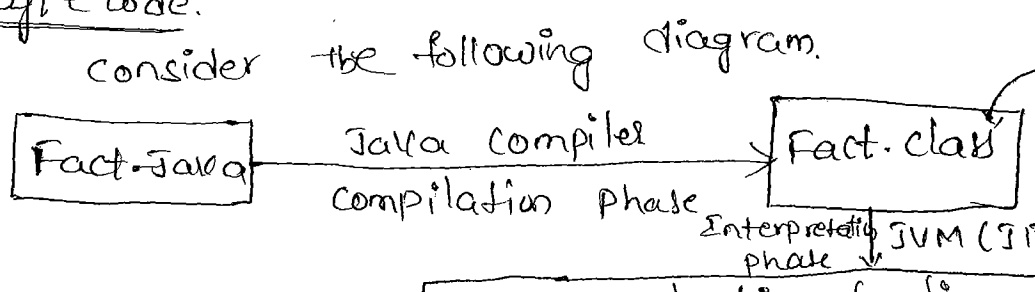
1. Simple
2. Platform Independent
3. Architectural Neutral
4. Portable
5. Multithreaded
6. Networked
7. Distributed
8. High Performance
9. Highly Interpreted
10. Dynamic
11. Robust (Strong)
12. Secured
13. OOPL

1) Simple :-

Java is one of the simple programming language becoz of the following 4 important technical factors.

1a) Java programming eliminates a complex concept called pointers. So, that appll development time is less & appll execution time is less becoz of magic of byte code.

consider the following diagram.



Def^n of Byte code :-

Byte code is the set of optimized instructions generated by Java compiler during compilation phase. & the nature of the byte code is much powerful than pointer code of C, C++ languages.

Def^n of JVM :- (Java virtual Machine)

JVM is one of the program developed SUN Micro sls developers, available as a part of Java slw & whose role is reading the line by line of byte code & converted into native understanding format of OS.

Hence, byte code is OS independent & JVM is OS dependent.

Def^n of JIT :- (Java Just in time compiler)

JIT is one of the program developed by SUN Micro sls developers, ~~available~~ added as a part of JVM & whose role is reading the line by line of byte code at once & converting into understanding format of OS.

In otherwords, the role of JIT is to speed up the interpretation phase.

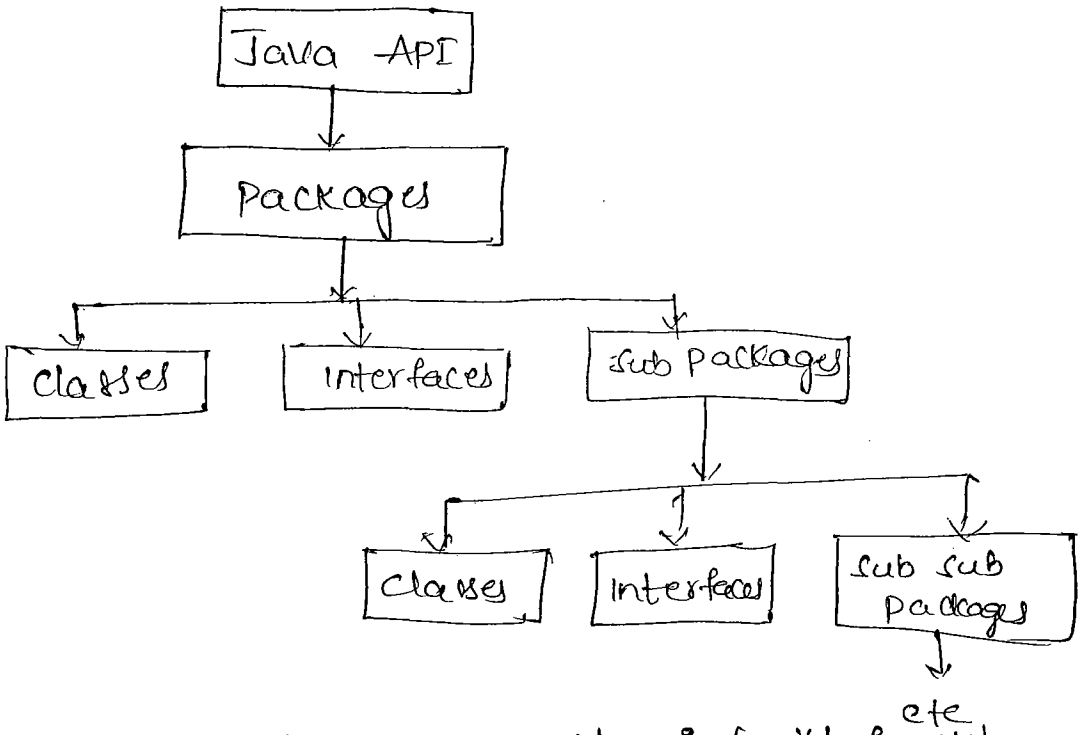
Hence, in this context Java is one of the compilation & interpretation based programming language.

Q.7 10

(b) Java programming provides rich set of API (Appl programming Interface).

Def of API :-

An API is a collection of pre defined Packages. A package is a collection of classes, interfaces & sub packages. A sub package interms contains collection of classes, interfaces & sub sub packages etc.



(c) Java programming language provides in build facility called "Garbage collector" which will collect un used memory space for improving the performance of Java based application.

Def of Garbage collector :- (GC)

A GC is one of the sls background Java program which running along with our regular Java program for improving the performance of Java

d, Java programming provides user friendly syntaxes which makes us to develop error free programs quickly.

Platform Independent :-

A language or technology is said to be platform independent if & only if whose applications runs on each & every OS.

In order to say a particular language is platform independent, it has to satisfy the following properties.

1. The language/technology related data types must take same amount of memory space on all OS's.
2. The language/technology must contain some special programs & whose role is converting native understanding format of one OS into native understanding format of another OS.

The languages like C, C++ etc & whose appls are treated as platform dependent becoz these languages never satisfies the above ①&② properties.

The language like Java & whose related appls are treated as platform independent becoz Java satisfies the above ①&② properties.

12

Multithreading :-

The aim of multithreading concept is to provide concurrent access/execution.

A flow of control is known as thread.

The purpose of thread is to execute logic of the java program which is written in the form of user defined methods.

If a java program is containing multiple flow of controls then it is known as multithreaded.

The languages like c/c++/pascal, COBOL etc are treated as single threaded modelling languages becoz their execution whose appl takes more environment provides single flow of control.

They provide sequential execution whose appl takes more execution time & does not contain any library for development of multithreading based appl

The languages/technologies like java & .Net are treated as multithreaded modelling languages becoz their execution environment provides multiple flow of control provides concurrent execution, whose appl takes less execution time & provides an effective library for development of thread based appl.

When we write a java program there exists 2 types of threads.

a. Foreground/Child thread :-

A foreground thread is one which is always executing logic of a java program which is written in the form of user defined methods.

b. Background/Parent thread :-

A background thread is one which is monitor execution

The real time implementation of multithreading concept is that to develop real world server softwares such as Tomcat, weblogic (BEA systems, websphere etc). In otherwords most of the real world server softwares developed by third party server vendors in java language by using multithreading concepts.

Hence multithreading of java is one of the specialized form of multitasking of operating systems.

Q:- How do you justify "Each & every java program is multithreaded"

When we execute any java program, the logic of the java program, the logic of the java program is executed by one of the thread known as foreground thread. To monitor the execution states of foreground thread, one more thread is created known as a background thread. So, java program is containing 2 threads hence every java program is multithreaded.

High Performance :-

Java is one of the high performance language becoz of following factors.

1. Magic of byte code (use less execution time)
2. Automatic memory management becoz of garbage collector (collects unused memory space for improving performance of java appl)

Highly Interpreted :-

In the initial versions of java compilation phase is very faster & interpretation phase is ~~also~~ slow which is one of the industry compliments & complaints

In the later versions of java SUN developers has developed a program called JIT (Just in time compiler) & added as part of JVM & whose role is to speed up interpretation phase by

(14) reading the entire section of the byte code & converting into native form of operating system.

In the current versions of java interpretation phase is very fast even compare with compilation phase. Due to this SUN developer has populated java is one of the highly interpreted programming language.

Robust (Strong):-

→ Java is one of the strong programming language becoz it contains a facility called exception handling.

→ The languages like C, C++, Pascal, COBOL etc & whose appls are treated as weak becoz these languages does not contain a facility called Exception handling.

→ Whereas java & whose appls are treated as strong becoz java contains a error handling facility called Exception handling.

Q:- Define Exception? what is meant by Exception Handling?

→ Runtime error of java program is known as exception.

→ Every exception of java program generates by default system/technical errors messages which are not understandable by appl users. Hence industry is not recomended to generate system error messages. But, recomended to generate user friendly error messages.

→ The process of converting system error messages into user friendly error messages is known as exception handling.

Dynamic :-

In any programming language we have two types of memory allocation technologies. They are

a. Static memory allocation
b. Dynamic memory allocation

→ In static memory allocation memory space is created/allocated for input of the program at compile time.

Due to static memory allocation we get the following limitations.

1. Waste of memory space
2. Loss of data
3. Overlapping of existing data

To avoid these problems industry is highly recommended to follow dynamic memory allocation.

→ In Dynamic memory allocation the memory space is allocated for input of program at runtime. Java programming follows only dynamic memory allocation. SUN Developers has populated that Java is one of the dynamic programming language.

Secured :-

→ Security is one of the principle used in the industry projects for protecting confidential information from unauthorized users.

→ In ~~Java~~ Non-Java programming language (C, C++, Pascal etc) there is no existing library for security.

→ Non Java language programmers needs to write their own code for providing the security & it may be complex hence every non Java programming language is unsecured.

→ In the case of Java programming we have a dedicated API (library) for dealing with a security i.e., In Java projects

(16) a java programmer need not to write any security program on their own & they can use readily available security program present in the existing API (like javax.security.*;)

Since it is containing readily available security API SUN developers has populated java is one of the secured programming language

Datatypes :-

The purpose of datatypes is to allocate sufficient amount of memory space for the input of the program in the main memory of the computer either by following static memory allocation (or) Dynamic memory location.

In every programming language data types are classified into 3 types. They are

1. Fundamental/ pre-defined datatypes (primary/ primitive/ core/ scalar/ built-in)
2. Derived data types
3. user-defined/ secondary/ custom/ programmer/ Referential datatypes

Fundamental datatypes are those which are developed by language developers & supplied as a part of their softwares & whose variables allows us to store single value but they never allows us to store multiple values of same type.

Ex :- int a;
a = 10; // valid
a = 10, 20, 30; // Invalid

(7) Derived data-types are those whose variables allows us to store multiple values of same type but they never allows us to store multiple values of different type

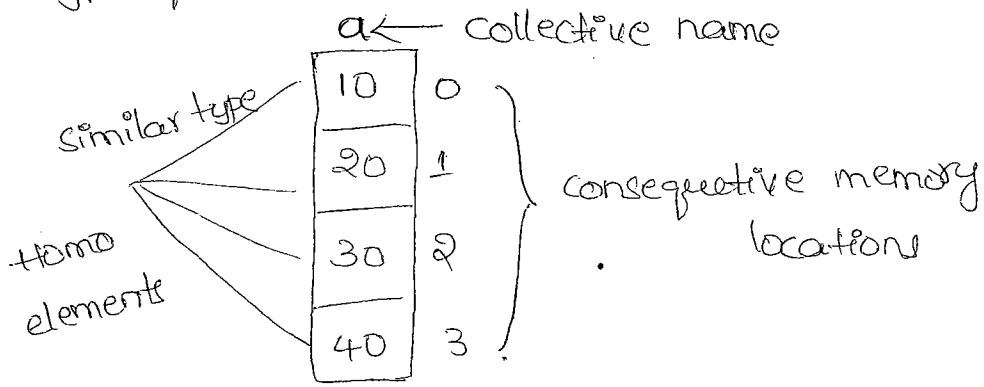
Ex:- int a[] = {10,20,30}; // Valid

int b[] = {10, 'A', 23.5}; // Invalid

In every programming language the concept of Arrays comes under derived data types.

Array :-

An array is a collective name given to a group of consequetive memory locations which are all refered by similar/homogeneous type of elements

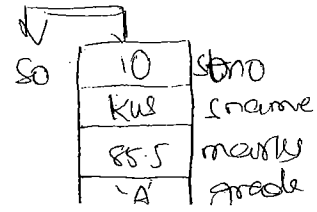


User defined data-types are those whose variables allows us to store multiple values either of same (or) different types (or) both types by using language facilities

Ex:- In 'c'-programming to create user defined datatype we use structures (struct), unions (union), Enumerations (Enum)

Similarly, in Java programming we use classes (class), interfaces (interface), Enumeration (Enum) etc. for development of user defined data types.

Ex:- Student s0 = new Student();



Java fundamental data types :-

In Java programming we have 8 fundamental data types which are classified into 4 categories. They are

1. Integer category datatypes
2. Float category datatypes
3. character category datatypes
4. Boolean category datatypes

Integer category datatypes :-

→ The aim of integer category datatypes is store integer data in the main memory of the computer by allocating sufficient amount of memory space.

→ Integer category datatypes contains 4 types & they are given in the below table.

S.NO	Datatype	size in byte	Range
1.	Byte	1	+127 to -128
2.	short	2	+32767 to -32768
3.	int	4	+x to -(x+1)
4.	long	8	+y to -(y+1)

→ If any category of data type contains multiple data types then it is mandatory to java program to choose an appropriate data type.

→ For choosing an appropriate data type we need to calculate the range of a data type. Range makes us to

(19) understand how much it can store at maximum side & how much it can store at negative side.

The following formula will be used for calculating range of any data type.

Range of any datatype = [The no. of bits available in the language which is understandable by computer] to the power of [No. of bits occupied by a perticular datatype]

Ex:-

Range of short datatype $= (2)^{16 \text{ bits}}$ 16 bits = 2x8

$= 1$ to 65536

$= 0$ to 65535

$= +32767.5$ to -32767.5

$= +32767$ to -32768

Float category Data types:-

The basic aim of float category data types is to store real constant values (π) in the main memory of the computer by allocating sufficient amount of memory space.

This category contains 2 types of datatypes. They are given in following table.

Datatype	Size (Bytes)	Range	No. of decimal places
Float	4	+x to (x+1)	7

(20) Float:-

If we store any real constant values in a variable of float data type then such real constant value stored in the main memory in such a way that after the dot(.) it takes 7 decimal places and the real constant value must be followed by a letter 'f' otherwise we get compile time error.

The following calculations shows (IEEE 754 : Floating Point calculations) how real constant values stored in the main memory in bit representation of float datatype

```
float f1 = 3.4f;
```

f_1

$\boxed{011.0110011}$

7-decimal places

for 0.4:

```
0.4
 *2
0.8   0
 *2
01.6  1
 *2
1.2   1
 *2
0.4   0
```

~~for~~ 3.4:

for 3

```
2 | 3
2 | 1   1
    0 1
    011
011.0110
```

Double:-

If we store any real constant value in a variable of double data type then the double datatype stores the real constant value in such a way that after the dot(.) it takes 15 decimal places & a real constant value may/may not be followed by a letter 'd'.

Ex:- double d1 = 3.4d;

(or)

(21) d1

$\underbrace{011.011001100110011}_{\text{15 Decimal Places}}$

In our Java program if we use any real constant value directly then such real constant value is by default treated as highest datatype in float category that is double datatype.

Float F1 = 3.5;
⇒ error

Becoz 3.5 is by default treated as double & it can't be directly stored in a variable of float type

NOTE:-

In 'C' language

EX:-

1. Float a=3.4;
If (a==3.4) double
{ PF("OK");}
else
{ PF("CANCEL");}
PF ("OVER");

ANS: CANCEL OVER

2. Float a = 3.5;
If(a==3.5) double
{ PF ("OK"); }
else
{ PF ("CANCEL");}
PF ("OVER");

ANS: OK OVER

This happens due to floating point calculations.

3.4:-
011.01100011 Float
011.011001100110011 double
which are not equal

3.5:-
011.1000000
011.1000000000000
which are equal

Character data type:-

→ A charater is an identifier which is enclosed within single quotes.

EX:- 'A', 'g', 'K', '$' etc.

→ A collection of / sequence of characters enclosed within double quotes is known as strings.

EX:- "swap", "Jesus my lord", etc

(29) → To store the character data in the main memory of the computer in C, C++, Java programming we use a data type called 'char'.

→ Char datatype in C, C++, takes 1 byte becoz C, C++ follows ASCII character set.

→ In Java programming char datatype takes 2 bytes becoz Java follows UNICODE character set. & Java programming is available in 18 International languages (English, Latin, Greek, etc)

Q:- Define ASCII & UNICODE

ASCII is a character set followed by those programming languages which are available in one International language called English & whose size is 256 ($2^8 = 256$) languages like C, C++, COBOL, follows ASCII character set as they are available in only english language.

UNICODE character set is one which is followed by those programming languages which are available in more than 1 International languages including English & whose size is 65536 characters ($2^{16} = 65536$)
2 bytes

Ex:- Java, .NET etc.

NOTE:- (Drawback of Java)

As per as English Java programmer is concerned for storing one character it requires one byte & rest of one byte is simply wasted which is one of the negligible drawback of Java language

(23) Boolean category data type :-

→ The aim of this category data type is to store logical values (Feeling that they are existing but in reality they won't take physical memory space in main memory)

Ex :- True, False.

→ In Java programming logical values are stored by using boolean

→ Boolean datatype of Java implemented by using a general purpose register called flipflop which will store one bit of memory space (1 for true & 0 for ~~flo~~ false)

→ Boolean data type of Java takes 0 bytes of main memory space becoz boolean values are not storing in main memory but they are storing in registers called flipflops.

NOTE :-

→ Each & every keyword in Java must be written in small letters.

→ The default value of Integer category variables is zero (0)

→ The default value of Float category variables is 0.0.

→ The default value of Character category variables is nothing.

→ The default value of Boolean category variables is false

Q :- Why character category data types requires 1 byte memory space in C while 2 byte in Java.

As 'C'-language supports only english language as it follows ASCII character set it requires single byte to store.

while Java uses UNICODE character set which supports is International languages including english.

consider in latin A is represented as A ← 1 byte ← 1 byte

To store such letters 2 bytes are required

Variables

Importance of variables:-

WKT by using data byte we can allocate sufficient amount of memory space in the main memory of computer & values are also stored in the memory space. To process the values which are stored in the memory space, it is mandatory to the programmer to give or qualify some distinct name to the memory spaces. These distinct names are used for retrieving values from the memory space & these names are called identifiers.

Identifier values are varying from one point of time to another point of time. So that identifiers are also known as variables.

Def'n of variable:-

A variable is an identifier whose values are changing during execution of the program.

Hence, to write a meaningfull program we require to use variables in other words without variables we can't store the data in the main memory of the computer

Rules for writing variables:-

Out of so many no. of. rules of variables, the following rules are suggested for writing variables

1. First letter of variables must be an alphabet.
2. In Java programming the length of variable can be upto 32 characters.
3. No special symbols are allowed except '_' (underscore)
4. No keywords to be used as variable names (if, while, else, final)

Variable Declaration :-

Def:- The process of allocation sufficient amount of memory space by giving distinct names to memory space is called "Variable Declaration".

* Without declaring a variables we can't store the data

SYNTAX :- datatype $V_1, V_2, \ldots V_n$;

Here datatype represents either fundamental datatype (or) derived (or) programer defined data type. $V_1, V_2 \ldots V_n$ represent JAVA valied variables names treated as name of the variables.

JAVA programming follows the declaration mechanisms of C & C++ languages.

Ex:- Write the set of instructions for computing sum of 2 numbers.

C-Program declaration

```
int a,b,c;
a=10;
b=20;
c=a+b;
```

a	b	c
10	20	30

Declared at begining of program

C++ program declaration

```
int a;
a=10;
int b;
b=20;
int c;
c=a+b;
```

a	b	c
10	20	30

Before using variable just declare before one line.

* The both languages must followed by JAVA.

What happens when a variable is declared

int a;

1. Memory space created 4 bytes
2. Memory space giving name
3. Memory space giving address
4. Memory space zero is replaced by value.

Variable declaration cum Initialization:-

Def:- The process of placing our own value without placing default value is known as "variable declaration cum Initialization".

The purpose of initialization concept is that to place common values which are suitable for all the programmers.

SYNTAX :- datatype $V_1 = val_1, V_2 = val_2, \ldots V_n = val_n$

Ex:- float pi = 3.1417f

String crs = "JAVA";

Constants in JAVA :-

Constant

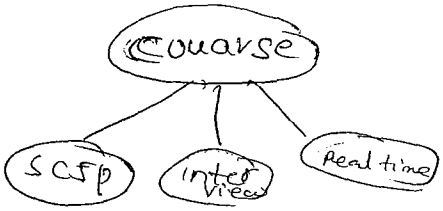
def: A constant is a variable whose value can't be changed during execution of the Program

In JAVA programming to make anything as constant we use a keyword called "final".

The keyword final can be applied in 3 places

They are

1. At variable level
2. At method level
3. At class level



(a) Final at variable level :-

At variable level it has two sub levels

(a) Final at variable initialization level :-

SYNTAX :- `final datatype v1=val1, v2=val2 --- vn=valn;`

once the initialized variable is made as final then modifications & assignments are not permited.

Ex:- final float pi = 3.14177f;

pi = pi+1; // Invalid

pi = 3.14178f; // Invalid

(b) Final at declaration level :-

SYNTAX :- `final datatype v1, v2 --- vn;`

Once the declared variable made as final then it allows first time assignment but it will not allow second & further subsequent assignments. & never allows any single modifications.

Ex:- final float pi;

```
pi = pi + 1; // Invalid
pi = 3.1417f + 0; // Invalid
pi = 3.1417f; // valid
pi = 3.1416f; // Invalid
```

Final variable values can't be modified.

2. Final at method level:-

In slw development if we devlope any fun which is common for all the programers then such funs are called "constant funs" & whose defn must be made as final. Hence one the method is final it is not possible to re define/over written.

3. Final at class level:-

If we don't want to give features properties of base class to the derived class then the defn of the base class must be made as final.

Final classes never participates in inheritance

1916

Development of (29) methods in JAVA :-

In every prog.g language for performing each & every type of operations we use the concept of functions.

In JAVA prog.g the concept of funns is known as methods.

Def.n of method :-

A part of main program is called method. (or)

A subprgm of main prgm is called method.

Parts of the methods :-

Methods development contains 3 parts. they are

1. Method call
2. Method deffination
3. Method declaration / Method prototype

The purpose of method is performing the operation which is required in our project & it can be defined once & used by various programers in many no. of place. Hence methods contains a slogan called

"Write once call anywhere".

No. of approaches to define the methods in real time :-

1. A method can take the values & a method can return the values.
2. A method will not take the values & a method will not return the values

3. A method will not take the values & a method can return the values.

4. A method can take the values & a method will not return the values.

SYNTAX for defining a method in JAVA:-

```
Returntype methodName (list of formal parameters if any)
  {
    Block of statements (S) - Business logic
  }
```

Explanation:-

1. Returntype represents type of value being returned by the method. If the method is not returning the any type of value then its return type must be void

> Every datatype is a return type but return type is not a data type

2. Method name represents a JAVA valid variable name treated as a name of the method. If a method name contains either one word or more than one word then first word 1st letter is small & rest of the subsequent words 1st letter must be capital.
(Ex:- getEmpDetails())

3. List of formal parameters represents the variable names used for holding the values which are sending from method calls.

4. Block of stmts represents set of executable stmts & they provides result of the method & it is known as business logic.

~~Function~~ Method def || = ~~Fun~~ Method || heading + ~~Fun~~ Method body

* write a method in JAVA for calculating sum of 2 numbers.

To define the above method we've 4 approches

Approach 1: A method T.V & a method R.V

Method deffination	Method call

int sum(int x, int y)
{
int z; // local variable
z = x+y;
return(z);
}
	① int res1 = sum (6,7);
	② int res2 = sum (10,20);
	③ int res3 = sum (8,9);

20/6/15

Approach 2: A method is N.T.V & a method is N.R.V

Method deffination	Method call

void sum()
{
int a=10;
int b=20;
int c=a+b;
print a,b,c;
	sum();
	sum();
	sum();

Approach 3:- A method is not taking values &
A method is returning the value

Method def"	Method call

int sum()
{
int a=10;
int b=20;
int c=a+b;
return (c);
}
	int res1= sum();
	int res2= sum();
	int res3= sum();

Approach 4: A method is taking values &
A method is not returning the values

Method def"	Method call

void sum (int x, int y)
{
int z = x+y;
print z;
}
	sum (10, 20);
	sum (5, 6);
	sum (-5, -9);

* Define a method in JAVA for interchanging of two values or swapping of 2 values.

Sol: To define the above method one is not recomended to choose those approaches which are returning the values. Recomended to choose those approaches which are not returning the values.

Approach 2 :- A method is NTV & a method is N.RV

Method def	Method call

NRV M NTV
void swap ( )
{
int x = 10;
int y = 20;
x = x+y; int t=x;
y = x-y; (6) x=y;
x = x-y; y=t;
Print (x);
print(y);
}
	swap ( );
	swap ( );

Approach 4 :- A method is TV & NRV

Method def	Method call

int swap (int x, int y)
{
x = x+y;
y = x-y;
x = x-y;
Print x, y;
}
	swap (10, 20);
	swap (2, 3);

The following table gives no. of approaches to define the methods vs phases within the methods.

No. of Approaches to define methods / No. of Phases in methods	Method T.V & R.V	Method N.T.V & N.R.V	Method N.T.V & R.V	Method T.V & N.R.V
Input	out side the method	Inside the method	Inside the method	outside the method
processing	Inside the method	Inside the method	Inside the method	Inside the method
output	outside the method	Inside the method	outside the method	Inside the method

NOTE :- For each & every method call their must exist a method def" otherwise we get compile time error

Method overloading :-

Method overloading = Method name is same + Signature is different

Signature represents the following points

→ No. of parameters
→ Type of parameters
→ order of parameters

} atleast one thing must be different

Ex: 1. Sum (10, 20, 30);
2. Sum (10, 20);
4. Sum (10, 1.5f);
5. Sum (1.5f, 20);

Here sum (...) is called overloaded method.

NOTE: At the time of overloading the method we never consider the return type but we consider always same method name with different signature.

Method overriding :-

Method overriding = Method heading is same + Method body is different

Ex:-

```
void op(int x, int y)
{
    int z = x+y;
    Print x,y,z;
}
```

```
void op (int x, int y)
{
    int k = x*y;
    Print k,x,y;
}
```

→ Defined by one programer

→ It is called original method

→ Defined by another

→ overriden method

"The process of re-defining original method method multiple times for performing multiple operations is called method overriding."

* In SW development if we develop any common method for most of the programmers then they can use the common method but they should not change the defn of method. and here common methods must be final. In otherwords final methods can't be overridden.

NOTE:- A overloaded method may/may not be final

Ex:- Define a method for calculating simple interest.

```
final float
void si (float p, float t, float r);
{
    float si = (p*t*r)/100;
    return (si);
}
```

Homework:-

→ Define a method for calculating area of a circle (area = πr^2)

→ Define a method for calculating area of rectangle (area = lxb)

→ Define a method for calculating perimeter of circle & rectangle

→ Define a method for ~~calcul~~ converting the temparature Celcius to Farnheat.

→ Define a method for finding biggest of two numbers.

22/6 OOps features / principles:-

In real world to develope any project we need some type of langl tech which will satisfies some principles either procedure oriented principles (POPS) or object oriented principles (OOPS). Based on the principles followed by langueages/ technologies, the languages are classified into two types.

1. Procedure oriented programming languges (POPL)

POPL :-

Ex:- Pascal, COBOL, ADA, Visual Basic, PERL, 8086, C, Developer 2000 etc.

If any PL satisfies POPS then it is called POPL. In real time we use POPL for developing projects related to S/S S/W & projects related to Appll S/W but not projects related to internet S/W (distributed appls).

If we choose POPL for development distributed appls then we get the following limitations.

1. POPL posses (gives) platform dependency
2. In POPL the data is visiting b/w client & server side appls in the form of plain text but not in the form of ciphere text.
3. POPL gives poor security
4. In POPL the data is visiting b/w client & server side appls in the form of byte by byte & results in poor com.n
5. In POPL the data is available around the funls & leads to redundancy (repetation) problems.

To overcome the above problems industry is highly recomended to use those PL's which will satisfy OOPS.

OOPL :-

Ex:- Small talk C++ Refill
Lisp JAVA Ruby etc
object pascal .net
object COBOL Simula

If ~~A~~ pl satisfies oops then it is called OOPL.

Advantages of OOPL In developement of distributed Applications (JAVA)

1. In oopl gives platform independency
2. In oopl appls the data is visiting btw client & server side appls in the form of cipher text/ encrypted form.
3. In oopl appls we get fully security.
4. In oopl appls data can be visiting btw client & server side appls all at once & results in effective com.n
5. In oopl app development the is around object & solves redundancy problem.

Inorder to say a particular language is object oriented it has to satisfy oops principles & they classified into 8 types. they are.

1. Classes
2. Objects
3. Data Encapsulation
4. Data abstraction
5. Inheritance
6. Polymarphism
7. Dynamic Binding
8. Message passing

All the above oops principles are common to all oopl but their SYNTAX'S/Implementations are changing from one oopt to another OOPL.

23/6

Class :-

→ classes concept is always used for development of user defined/programmer defined data types.

→ While we are using classes concept for development of user defined data type we use a keyword called "class".

→ Each & every Java program must start with a concept of class. In otherwords without classes concept there is no Java program.

→ Each & every class name in Java can be treated as user/Programmer defined data type.

Definition:-

The process of binding data members & associated methods in a single unit is known as class

(OR)

A class is a collection of data members & methods.

Whenever we define a class there is no memory space is created for the data members & methods ~~must~~ but memory space will be created for the data members & methods when we create an object wrt a class. Hence, all the classes contains logical existance & object contains physical existance

In object oriented programming we've 2 types of methods they are.

A. Member methods

B. Non-member methods.

→ A Member method is one which is defined inside the scope of the class & it can access the data of the class.

→ A Non-member method is one which is not available within the scope of the class & it can't access the data of the class.

→ Java programming allows only member methods but not Non-member methods.

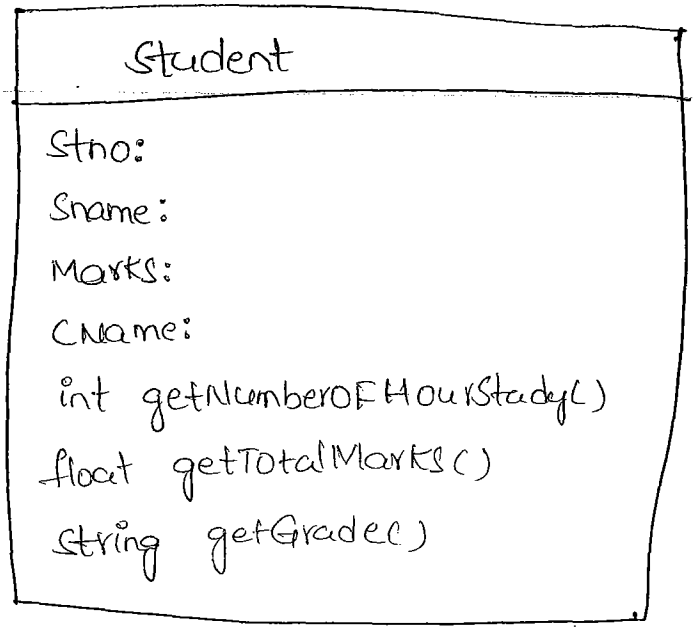
→ Data members of a class are known as properties (or) fields (or) attributes. & methods are also known as Behaviour (or) Accessories.

→ In real world appls development classes can be expressed by using class diagrams & they can be developed by using UML softwares like Rational Rose, Turbo Analyst (IBM products).

The structure of class diagram is

Class Name
Data members
methods

Ex :- Draw a class diagram for student. (41)



Syntex for defining a class in Java:-

```
class classname
 {
    variable declaration (data members)
    methods defination
 }
```

→ class is a keyword used for developing user/ programmer defined data type.

→ classname represents a Java valid variable name treated as the name of class. Each & every classname in Java is treated as userdefined datatype whenever we define a class there is no memory space is created for the data members & methods of a class but whose memory space is created when we create an object wrt class.

→ variable declaration represents data members of a class & they will be selected depends on the name of a class.

→ Method deffination represents set of methods meant for performing some specific operations & they will be selected depends on the name of the class. Each & every method in Java must be defined inside the class i.e, no method of Java to be defined outside the class.

→The defination of the class must be incorporated within {and} & the defination of the class may/may not be terminated by (;) Semicolon.

NOTE

If a class name contains either one word (or) more than one word then all words first letters must be captical.

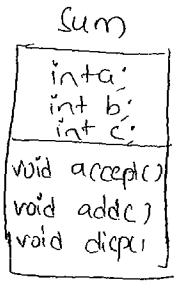
Ex:- ActionEvent, NumberFormatException, Button etc.

Q:- Write a class defination for Student.

```
Class Student
{
  int sno;
  String name;
  float m1,m2,m3;
  int getNumberOfHourStudy()
  {
     return(2);
  }
  float getTotalMarks()
  {
     float tot=m1+m2+m3;
     return(tot);
  }
  String getGrade()
  {
     return("Destination");
  }
} //Student
```

Q:- Write a class defination for computing sum of two numbers.

```
Class Sum
{ int a,b,c;
  void accept()
  {
     a=10;
     b=20;
```



```
void add()
{
  c=a+b;
}
void disp()
{
  sop("val of a="+a);
  sop("val of b="+b);
  sop("val of c="+c);
}
}//Sum
```

Object:-

→ WKT when we define a class there is no memory space created for data members & methods of class but whose memory space is created when we create an object w.r.t class.

→ In otherwords if we want to enter values of data members of the class first we have to allocate memory space for data members of class by creating its object.

→ To create an object there must exist a class defination. Without object creation data processing is not possible in Java based applications.

Defns of object:-

1. Instance of a class is known as an object (Instance is nthg but allocating sufficient amount of memory space for data members & methods of a class).
2. Each & Every class variable is known as an object.
3. Each & every grouped item is known as an object.
4. Real world entities are called objects.
5. Blueprint of a class is known as an object.

1, 2 & 3 are recommended during interfaces & 4 & 5 are not recomended

Creating an object in Java:-

Creating an object is nothing but allocating memory space for data members & methods of a class by following dynamic memory allocation by using new operator & it is known as dynamic memory allocation operator.

Functions of new operator :-

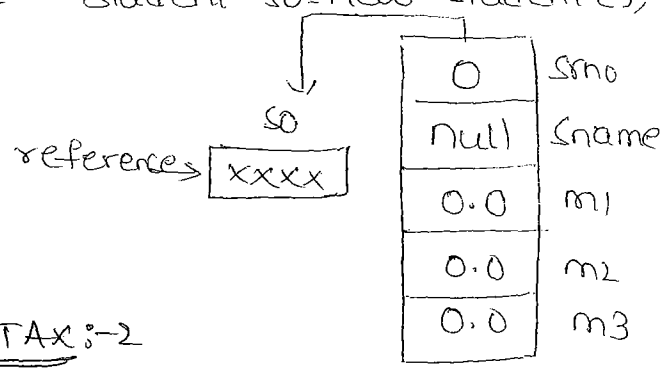
1. It allocates sufficient space for datamembers & methods of a class.
2. It takes an address/reference of a loaded class & place it into LHS variable i.e., object name.

TO create an object with new operator we have 2 syntaxes.

SYNTAX :- 1

<clsName> objname = new <clsname()>;

Ex :- Student so=new Student();



SYNTAX :-2

<clsName> objname;

Objname = new <clsName()>;

Here statement 1 represents object declaration when ever an object is declared, memory space is not created for data members & methods of a class & whose default value is null.

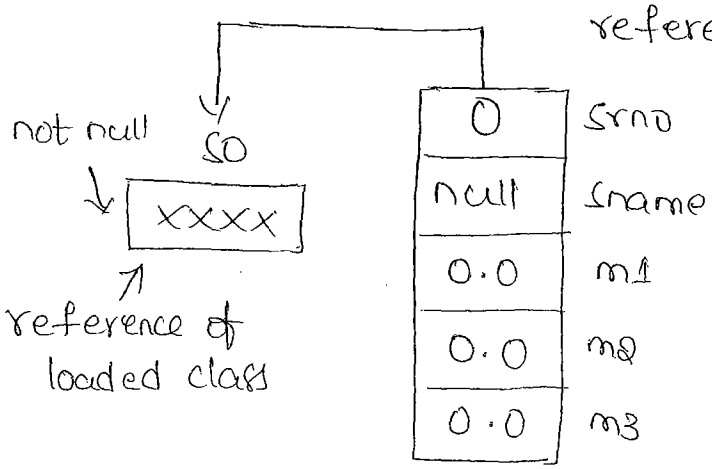
Student so; // Student Object declaration

so

null

Statement 2 represents object referencing. whenever an object is referenced memory space is allocated for data members of the class & whose default value is not null (which is ntng but reference/address of loaded class)

So = new student(); // Student Object referencing memory space



Q:- Define a class loaded subsystem.

It is one of the program developed by SUN MS & added as a part of Java S/w & whose role is to transfer or load the class files from secondary memory (Hard disk) into main memory (RAM).

Class	object
1. A class is a collection of data members & associated methods.	1. Incidence of a class called an object.
2. Whenever we define a class memory space is not created for the data members & methods of a class.	2. When we creat an object memory space is created for the data members & methods.
3. classes contains logical existance	3. object contains physical existance.
4. Defination of a particular class will exists only once	4. Wrt one class def^n one can creat multiple objects.
5. Whenever we execute the JAVA program the def^n of the class will be loaded in to main memory only once. with the help of class loader sub. system.	5. After loading the class def^n in the main memory whose objects will be created later.
6. The def^n of the class resides in secondary memory in the form of .class file.	6. Objects of JAVA resides in main memory.

Note:-1

All the objects of JAVA programing resides in heap memory :

Note:-2

All the methods of class/objects which are

Note:3

All the content/figurative values resides in associative memory.

* All the above 3 memories are the parts of main memory.

Types of data members in JAVA:-

In JAVA programing we've 2 types of data members.

They are 1. Instance/Non-static Data Members

2. Static Data members

IDM	SDM
1. IDM's are those whose memory space is created each & every time whenever an object is created.	1. SDM's are those whose memory space is created only once when the class defn is loaded in main memory irrespective of no. of objects are created.
2. IDM's are used for storing specific values.	2. SDM's are used for storing common values which are suitable for all the programs
3. Programatically IDM's declaration should not be preceded by "static" keyword.	3. Programatically, SDM's declaration must be preceeded by "static" keyword.
4. SYNTAX:- datatype $v_1, v_2 \cdots v_n$;	4. SYNTAX:- static datatype $v_1, v_2, \cdots v_n$;
5. Each & every IDM must be accessed w.r.t. object name.	5. Each & every SDM must be accesed w.r.t. class name (even with object also) clsname. static SDM name

6. IDM values are not sharable	6. SDM values are sharable
7. IDM's are also known as "object level data members" ~~OB~~ (OLDM)	7. SDM's are also known as "class level data members" (CLDM)
8. EX:- int PIN; int sno; String name;	8. EX:- Static String CRS = "JAVA"; Static String capital = "Delhi"; Static float PI = 3.141ff;

Types of methods in JAVA:-

In JAVA programing we've 2 types of methods. They are 1. Instance/non static methods
2. Static methods

IM	SM
1. IM's are meant for performing repeated/specific operations such as reading the records from files, reading the records from the database etc.	1. SM's are used for performing one time/common operations. Such as opening the file, obtaining the data base connection.
2. programatically, IM's defn should not be preceeded by static keyword.	2. programatically, SM's defn must be preceeded by static keyword.
3. SYNTAX:- Returntype methodname (List of formal parameters if any) {	3. SYNTAX:- static Rettype methodname (List of formal parameters if any) { Block of statements;

4. Each & every IM must be accessed w.r.t. object name.

Objname.IM name ( )

5. The resut of instance method is not sharable.

6. IM's are also known as "object level methods".

7. EX :-

```
void calTotal()
{
total = m1+m2+m3;
}
```

4. Each & every SM must be accessed w.r.t. class name. (Even with object also).

clsname. SM name ( )

5. The result of SM is sharable

6. SM's are also known as "class level methods"

7. EX :-

```
Public static void main ( )
{
  ==
  S1.calTotal ( );
  S2.calTotal ( );
  ==
}
```

* A class of JAVA can contain IDM's, SDM's, IM's & SM's.

* IDM's & IM's ~~can~~ of a class must be called wrt object name whereas SDM's & SM's must be called wrt class name.

26/6/15

System.out.println ( ) → ① (new line)

system.out. print ( ) → ②

C++ [OOPL]

To display - result

cout - object - ostream (iostream.h)

To read

cin - object - istream

→ Statement ① is used for displaying the result of the JAVA program on the console line by line

→ Statement ② is used for displaying the result of JAVA program on the console in the same line

Technical Description :-

→ Here Println (-) & print (-) are two pre defined overloaded instance methods present in a pre-defined class called printStream.

→ To access Println (-) & Print(-) methods we need an object of PrintStream class.

→ An object of printStream class called out is created at the static data member in an another Pre-defined class called system. Hence, to access Println (-) & print(-) methods we use the following statements.

(i) system.out.Println ( )
(ii) system.out print ( )

consider the following code segment we gives view about technical description of above ①&② statements

```
class system
{
  ___
static PrintStream out=
        new PrintStream
  ___
```

```
class PrintStream
{
  void Println(-)
  {
    ___
    ___
  }
  void print (-)
  {
    ___
```

EX :- Display "Hello Java world" msg

```
System.out.println("Hello Java world");
```

EX :-2 int a=10

```
System.out.println(a);          // 10
System.out.println("val of a = "+a);   // val of a = 10
System.out.printlb("val of a = "+a);
System.out.println(a+" is the val of a");   // 10 is the value of a
```

EX :-3 int a=10, b=20;

int c= a+b;

Public – universal access modifier

```
sop ("sum = "+c");   // sum=30
sop (c+ " is sum");   // 30 is sum
sop ("Sum of "+a+" and "+b+" = "+c);   // sum of 10 & 20 = 30
```

Structure of the Java program :- Main – Psuedo Method compiler

The standard format developed language developers which is used by programmers is called "structure of the program".

The following standard format used for developing JAVA appls

```
Packages info;
class <clsname>
{
    Data members
    User defined methods
```

```
Public static void main (String K[ ])
    {
        Block of Stmt (s)
    }
}
```

(1) → Packages info represents collection of classes & interfaces & sub packages. A sub package internally contains collection of classes, interfaces & sub sub packages etc. If we use any pre-defined class (or) interface as a part of our Java prgm then it is mandatory to the Java programer to specify in which package the classes & interfaces presents otherwise we get compile time error. Out of many no-of packages in Java programming one package is by defaeult imported known as "Java.lang.*".

(2) → Each & every Java prgm must starts with a concept of class. Here, class is a keyword used to develope prgmr defined data type.

(3) → <clsname> represents a Java valid variable name treated as name of the class.

(4) → Data members represents either IDM or SDM & they will be decided based on the cls name.

(5) → user defined methods represents either IM or SM & they will be decided based on the clsname

⑥ → Each & every Java program starts executing from main method & it is called Program driver. (Psuedo method)

⑦ → Since main method of Java is not returning any value & hence its rettype must be void.

⑧ → Since main method of Java executes only once in the entire life of the Java program & hence its nature must be static.

⑨ → Since main method of Java accessed by all the Java programers & hence whose access modifier must be public.

⑩ → Every main method of Java takes array of objects of strgs

⑪ → Block of stmts represents set of executable stmts which are interns calls user defined methods of the class.

⑫ → The file naming convention in Java programing is that whatever the class is containing the main method, it should be given as a file name with an extension

• Java.

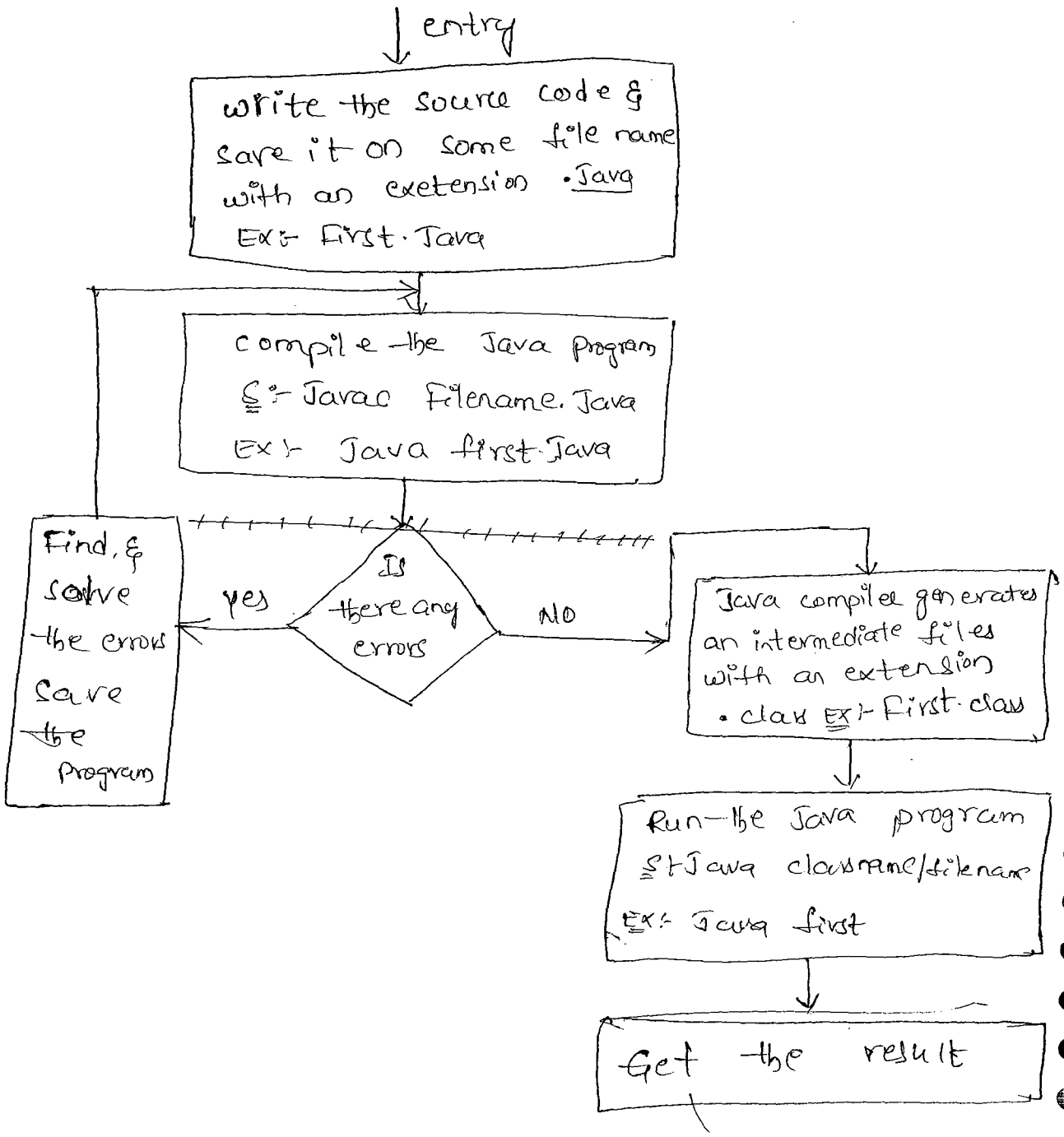
* write a Java program which will print "Hello Java world".

```
// First.Java
class First
{
public static void main (strg ats k[])
    {
      sop("Hello Java world");
      sop("This is my first program");
    }
}
```

27/6

Compiling & Running the Java program :-

The following diagram gives sequence of steps for compiling & executing the Java program.



In the above diagram Javac & Java are called tools (or) appl programs (or) exe files developed by SUN micro sys developers & supplied as the part of JAVA-HOME/bin directory & whose purpose is

Requirement analysis for developing Java project/appn :-

1. Software Requirement

Download JDK 1.5/1.6 or 1.7/1.8 from www.sun.com/ www.oracle.com freely.

2. OS Requirement

Any OS becoz Java is platform Independent language.

3- IDE Requirement (Integrated Development Environment)

An IDE is the one of the 3rd party s/w developed by 3rd party s/w vendors, released to the real industry & used by the all programmers for development of different appls by making use of inbuild comfortabilities.

EX

a. DOS Editor, Note pad, word pad etc } Poor IDE Industry will not use

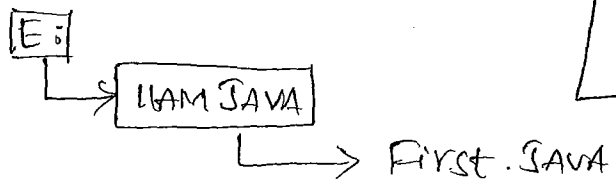
b. Edit PLUS- ES Enterprise } Poor IDE Industry will not use

c. Net Beans, Eclipse, MyEclipse, WSAD, Intelli JIDEA, RAD etc } Effective IDE's used in Real Industry

Setting the path for compiling the Java program:-

1. Creat a directory (E: 11AM JAVA) & Save the a filename in this directory

Ex:-



JVM - Java virtual Machine

2. Compile the Java program

E:\ 11AM JAVA > JAVAC First.Java

⇒ we get error becoz Javac doesn't exist in 11AM JAVA. Inorder to use Javac tool as the part of our directory we must set the path to the location where Javac present.

E:\ 11AM JAVA > set path = D:\ Java\ jdk1.6\ bin

⇒ Recompile the above program after setting the path

E:\ 11AM JAVA > Javac First.Java.

⇒ Ensure that Java compiler generates First.class. In general .class file will be generated based on class defn but not on file name

3. Run the Java program

E:\ 11AM JAVA > Java First

⇒ when the above stmt is executed internally the following 4 operations will takes place.

i class loader sub sys loads/transfers First.class into main memory from secondary memory.

ii, JVM takes loaded class (First-class). (54)

iii, JVM looks for main method to starts its execution.

iv JVM will call the main method w.r.t the loaded class first.class as first.main()

→ 29/6

* Write a Java program which illustrate the concept of methods chaining.

Def^n of Method chaining :-

The process of establishing the com^n btw multiple methods for performing some operation is called "method chaining".

```
class Third
{
    void show()
    {
        System.out.println("show()- instance");
        write(); //Rule-2
    }
    static void disp()
    {
        System.out.println("disp()- static");
    }
    void write()
    {
        Third. disp(); // Rule-4
```

```
System.out.println ("write ()-instance");
}
public static void main (String [] args)
{
    System.out.println ("i am from main()-beg");
    Third t1= new Third();
        t1.show (); //Rule-3
        disp (); //Rule-1
    System.out.println ("i am from main ()-End");
}
}
```

Method chaining Rules :-

Rule : 1

one static method can call another static method directly provided both the static methods belongs to same class.

Rule : 2

one IM can call another IM directly provided both the IM's belongs to same class.

Rule : 3

one SM can call another SM directly w.r.t object without considering wheather they belongs to same class.

Rule :- 4

one IM can call another SM w.r.t. class name without considering wheather they belongs to same class (or) different class.

Q:- Can we write multiple main methods in Java

Ans:- Yes,

In one Java program we can write multiple main methods. where name of the method is same but whose signature is different i.e, overloaded main methods can be defined.

NOTE:- The following code gives overloaded main methods

```
//Exam.java
class Exam
{
  public static void main(float x) //---(3)
  {
    System.out.println("i am from main()-taking float param");
  }
  public static void main(String x[]) //---(1)
  {
    System.out.println("i am from main()- taking array of objects");
    main(10); // control goes to (2)
  }
```

```
public static void main (int[]) // —(2)
  {
    System.out.println("i am from main()-taking int Parameter");
    main(1.5f);// control goes to (3)
  }
} // Exam- class
```

<u>NOTE</u>:- It is not possible to define duplicate method defns which is already existing with same Rettype & same signature.

* write a Java Program which will compute multiplication of 2 numbers.

```
// MulDemo.Java
class Mul
{
 int a,b,c;
 void read()
 {
    a=6;
    b=5;
 }
 void multiply()
 {
   c=a*b;
```

```
void disp ()
  {
    System.out.println (a+" * "+b+" = "+c);
  }
} // Mul-BLC
class MulDemo
{
  public static void main (String [] args)
  {
    System.out.println ("i am from main()-beg");
    Mul m= new Mul();
      m.read ();
      m.multiply();
      m.disp();
    System.out.println ("i am from main() - end");
  }
} // MulDemo
```

If we run the above program we get same o/p in all the times. If we want to get different results we must do the following cyclic steps.

1. go to the program
2. Change the required values
3. Save the program
4. compile the program

If any program satisfies the above charts then it is called "static programing/hard coded approach". which is not a recomended process in sw development becoz appl users may not be able to perform the above steps.

Hence, programmers must always develop "dynamic programming/flexible approach".

Q:- Define Hard coded Approach & Flexible Approach.

Ans:- In HCA i/p of the program mentioned in the program participated during compilation time & used at execution time.

In FA/DA i/p of the program supplied at run time & used at run time & it is the highly recomended process.

30/6

Accepting the Values dynamically to the Java Program

In Java programming we can accept the data dynamically by using 4 approaches. They are

1. By using command prompt
2. By using Keyboard
3. By using properties file/resource bundle file
4. By using xml Document.

Length:- Length is one of the Implicit variable created by JVM & supplied to each & every Java program for 2 purposes. They are

1. It is used to find no. of elements present in arrays i.e., size of the array.
2. To treat fundamental arrays (int[], float[] etc) as objects internally by the JVM.

SYNTAX :- arrayName. length

Ex :- 1

```
int a[] = {10, 20, 30}
sop ("No. of elements = " + a. length); // 3
for ( int i=0 ; i< a.length; i++)
{
    sop (a[i]);
}
```

Ex :- 2

```
float x[] = {1.9f, 2.5f}
sop ("No. of elements in x = " + x. length); // 2
for (int i=0; i< x. lenght; i++)
{
    sop(x[i]);
}
```

Ex :- 3 array of objects of string class

```
String S[] = {"c", "cpp", "Java"}
sop (" No. of courses = " + S. length); // 3
for (int i=0; i< s.length; i++)
{ sop (S[i]);
```

1. Accepting the data dynamically from command prompt :-

Consider the following statement

E:\\ 11AM Java > Java MulDemo 10 20

command prompt — Java program — values

The no. of values passing to the Java program from cmd prompt are called cmd line arguments

When the above stmt is executed the following steps takes place internally.

(i) class loader sub s/s loads / transfer MulDemo into main memory along with cmd line arguments (Ex:- 10 20)

(ii) JVM takes loaded class MulDemo along with cmd line arguments (Ex:- 10 20) & counted them. [Ex: length = 2]

(iii) JVM looks for the main ~~method~~ (String K[length])

(iv) JVM calls the main method (String K[2]) w.r.t loaded class MulDemo. as MulDemo.main(String K[2])

10	0
20	1

Ex:-

E:\\ 11AM Java > Java StudentDet 10 sathya JNTU 99

calling / w.r.t

StudentDet. main(String K[4])

```
for(int i=0;i<K.length;i++)
{
  Sop(K[i]);
```

10	0
sathya	1
JNTU	2
99	3

* write a Java prgm which will accept cmd line argum & display those values on the console.

```
// DataRead.Java
class DataRead
{
  Public static void main (String K[])
  {
    sop("no. of cmd line args = "+ K.length);
    for (int i=0; i< K.length; i++)
    {
      sop(K[i]+ " ");   for our own spacing
    }
  }
}
```

→ compile the above program

Javac DataRead.Java

→ Ensure DataRead.class file must be generated

→ RUN the Java program by passing cmd line cargs

Ext Java DataRead 10 Sathya JNTU

→ output :

```
No. of cmd line args=3
10
Sathya
JNTU
```

1/7 converting string type data into numerical/fundamental type data :-

WKT whatever the cmd line arguments we pass to the Java program, they are taken by JVM & placed in main method & available in the form of array of objects of String class. WKT on string type data it is not possible to perform numerical operations & hence it is highly desirable to convert string type data into fundamental/numerical type data.

The following table gives Fundametal data type, corresponding wrapper class name & conversion method which will convert numerical string type data into numerical/Fun type data.

Fund Datatypes	wrapper class Name	conversion method whic will convert numerical string type data into Num/Fun type
byte	Byte	public static byte parseByte (String)
short	Short	public static short parseShort (String)
int	Integer	public static int parseInt (String)
long	Long	public static long parseLong (String)
float	Float	public static float parseFloat (String)
double	Double	public static double parseDouble (String)
char	Character	—
boolean	Boolean	public static boolean parseBoolean (String)

In general each & every wrapper class (except character) contains the following predefined generalized method

wrapper class name

public static xxx parseXxx (String)

Here xxx represents any fundamental data type (except char data type)

Q:- What r the 3 points to be remember for using Java methods from Java API in your Java programme.

1. Decide what type of method we r using (either IM or SM) & present in which class.
2. Decide what type of parameter a method is using & pass its value
3. Decide what type of return type a method is returning & hold the result in a variable of that type. (ret. type)

Ex:- String S_1 = "10";
String S_2 = "20";
Sop(S_1+S_2); // 1020
int x = Integer.parseInt (S_1);
int y = Integer.parseInt (S_2)
int z = x+y;
Sop ("Sum = "+z); //30

```
Sop("sum= "+x+y); //1020
Sop("sum= "+(x+y)); //30
Sop("sub= "+x-y); // error
Sop("sub= "+(x-y)); //10
Sop("mul= "+x*y); //200
Sop("DIV= "+x/y); //0
Sop("Rem= "+y%x); //0
Sop("Rem= "+x%y); //10
```

* write a Java program which will calculate multiplication of 2 numbers from by accepting 2 integer values from command prompt.

```
// MulDemo1.Java
class Mul
{
  int a,b,c;
  void read(int x, int y)
  {
    a=x;
    b=y;
  }
  void multiply()
  {
    c=a*b;
  }
```

```
void display()
{
  System.out.println(a+" * "+b+" = "+c);
}
} // MUL-BLC

class MULDemo1
{
    public static void main (String [] args)
    {
      // accepting cmd line args & converting into int type
      int x = Integer.ParseInt (args[0]);
      int y = Integer.parseInt (args[1]);
                                  -> sop ("i am from main() - beg");
    // create an object of BLC
        Mul m = new MULL();
    // send the dynamic values
        m.read (x,y);
        m.multiply();
        m.disp();
    System.out.println ("i am from main() - 'end");
```

→ compile above program

Javac MULDemo1.Java

→ Run the Java program by passing 2 cmd line args

Java MULDemo1 5 6

FAQ :- Define a wrapper class? What is its purpose.

Ans :- For each & every fundamental data type there exist pre-defined class & it is known as wrapper class.

The purpose of wrapper classes in Java is that converting numerical string type data int numerical fundamental type data.

2/7

* Write a Java program which will accept 2 integer values & swap them.

```
// SwapDemo.Java
class swap
{
  int a,b;
   void read()
   {
     int a=x;
     int b=y;
   }
   void swapvalue()
   {
      a=a+b; // int t=a
      b=a-b; // a=b
      a=a-b; // b=t
   }
```

```
void disp()
{
  sop("Values of a = "+a);
  sop("values of b = "+b);
}
} // BLC

class SwapDemo
{
  public static void main(String K[])
  {
    if (K.length != 2)
    {
      sop("plz enter two values: ");
    }
    else
    {
      // accepting cmd line args & converting into fund values
      int x = Integer.parseInt(K[0]);
      int y = Integer.parseInt(K[1]);
      Swap So = new Swap();
        So.read(x,y);
      sop("original values");
      sop("_____");
        So.disp();
      sop("_____");
      So.SwapValues();
```

```
    sop(" Swapped values");
    sop(" ____");
    So.disp();
    sop(" ____");
  }// else
}//main()
}// Swap Demo - EL
```

* Write a Java program which will compute square of a given number. Pass the number from the cmd prompt.

```
// SquareDemo.Java
class square
{
  int n;
  void set (int x)
  {
    n=x;
  }
  void squareCal()
  {
    int res = n*n;
    return res;
  }
  void disp()
  {
    int r= squarecal();
```

```
} // Square- BLC
class SquareDemo
{public static void main (String [] args)
 {
   if (args.length != 1)
   {
     Sout ("plz enter one value");
   }
   else
   {
    int x= Integer.parseInt (args[0]);
     Square so= new Square();
         so.set (x);
         so.disp();
   } // else

 } // main
} // ELC
```

* Write a Java program which will decide wheather the given number is +ve or -ve or zero.

```
// NumberDecideDemo1.Java

class NumberDecide
{
  int n;
  void set(int x)
  {
    n=x;
```

```
String decide()
{
  String s = " ";
   if (n>0)
   {
     s = n+" is +ve";
   }else
  ~~if (n~~
    else
   {
     if (n<0)
     {
       s=n+" is -ve";
     }
     else
       s=n+" is zero";
   } // else
   return s;
  } //decide()
  void disp (String z)
  {
    sop (z);
  }
} // NumberDecide-BLC
```

```
class NumberDecideDemo1
{
    public static void main(String [] args)
    {
      if(args.length != 1)
      {
        sop("plz enter only one value");
      }
      else
      {
        int m = Integer.parseInt(args[0]);
        NumberDecide no = new NumberDecide();
        no.set(m);
        String res = no.decide();
          no.disp(res);
      } // else
    } // main()
} ELE
```

NOTE:- A method of Java is notonly returning fundu types (values) but also returns class name as a return type.

NOTE:- A method of Java is notonly taking fundu type as parameter but also takes class ~~typ~~ names as parameter types (objects as parameters)

* write a Java program which will calculate area of a circle by accepting radius from cmd prompt
* write a Java program which will calculate area of rectangle by accepting length & bredth from cmd prompt

```
//AreaDemo.Java
class Area
{ int r;
  void set (int x)
  {
    r=x;
  }
  void areaCircle ()
  {
    float a = 3.1417f * r * r;
  }
  void disp ()
  {
    Sop(" area of circle = " +a);
  }
} //Area- BLC
class AreaDemo
{
  Public Static void main (String K[])
  {
    if (K.length !=1)
```

```
{
  sop("plz enter one value");
}
else
{
    int x= Integer.parseInt ( K[0]);
      Area A0 = new Area();
          A0. set (x);
          A0. areacircle ();
          A0. disp ();
  } // else
} // main
} // Areademo - ELC
```

```
* // Areademo1.Java
class Area
  {
    int l,b,a;
    void set (int x, y)
    {
      x=l; l=x;
      y=b; b=y;
    }
  void  arearect()
  {
      a = l*b;
  }
```

```
void display()
  {
    Sop("a+" *"+b+"= "+a);
  }
} // Area-BLC
Class AreaDemo1
  {
    public static void main(String K[])
    {
      if(K.length != 2)
      {
        sop("plz enter two values");
      }
      else
      {
        int x = Integer.parseInt(K[0]);
        int y = Integer.parseInt(K[1]);
        Area A0 = new Area;
          A0.set(x,y);
          A0.arearect();
          A0.display();
      } //else
    } //main
} // ELC.
```

3/7

* Write a Java program which will print 1 to n numbers by accepting n-value from cmd prompt.

NOTE:- In every programming language we use control structures for performing either same operation or some other operation only once dependents on the condition evaluation or perform some operation repeatedly until condition becomes false.

control structures condition contains

a, conditional stmts(if, if else, switch) are used for for performing either one operation or some other operation only once.

b, looping stmts (while, do while, for) are used for performing some operation repeatedly until condition becomes false

To deal with looping kind of prgms we need to ensure 3 parts must present in our program. They are

a, Initialization part

b conditional part

c updation part(Incrementation / Decrementation)

Prgm :- ~~NumGen~~

```
//NumGenDemo.Java

class NumGen
{
   int n;
   void set(int x)
   {
      n=x;
   }
   void generate()
   {
     if (n<=0)
     {
       System.out.println("plz enter +ve value");
     }
     else
     {
      for (int i=0; i<=n; i++)
      {
       s.o.p(i);
      }
     } //else
   }
} //NumGen_BLe
```

```
class NumGenDemo
{
  public static void main (String [] args)
  {
    if(args.length != 1)
    {
      sop("plz enter one value");
    }
    else
    {
      int x = Integer.parseInt(args[0]);
      NumGen no = new NumGen();
          no.set(x);
          no.generate();
    } // else
  } // main
} // NumGenDemo — ELC
```

* Write a Java prgm which will accept a numeric integer from the cmd prompt & generate multiplication table.

```
// MultiplicationDemo.Java
class Multiplication
{
  int n;
```

```
void set(int x)
{
    n=x;
}
void table()
{
  if (n<=0)
  {
    sop (" plz enter +ve value);
  }
  else
  {    ~~int a=n*i~~; sop ("multiplication table for i "+n);
       sop("= = = = = = = = = =");
    for (int i=1; i>=10; i++)
    {
      sop(+n"*"+i " = "+a);
    }
    sop("
  } // else
}
} // BLC
class MultiplicationDemo
{
  public static void main(String [] args)
  {
    if(args.length[] =1)
    {
      sop(" plz enter one value");
```

```
else
{
    ~~int x = Integer.parseInt(args[0]);~~
    int x = Integer.parseInt(args[0]);
        Multiplication m = new multiplication();
                m.set(x);
                m.table();
    } // else
} // main
} // ELC
```

* Write a Java Program which will convert ordinary number into Roman number.

Logic Analysis :-

10 - X - W	100 - C - W	1000 - M - W
9 - IX - if	90 - XC - if	900 - CM - if
5 - V - if	50 - L - if	500 - D - if
4 - IV - if	40 - XL - if	400 - CD - if
1 - I - W		

```
// RomanDemo.Java
class Roman
{
    int n;
    void set(int x)
    {
        n=x;
    }
```

```
void convert ()
{
  if(n<=0)
  {
    sop("No. Roman Eqv");
  }
  else
  {
    while(n>=1000)
    {
      s.o.print(" M");
        n=n-1000;
    }
    if (n>=900)
    {
      s.o.print(" CM");
        n=n-900;
    }
    if(n>=500)
    {
      soprint(" D");
        n=n-500;
    }
    if (n>=400)
    {
      soprint ("CD");
```

```
while( n>=100)
{
  soprint( "c");
      n=n-100;
}
if( n>=90)
{
  soprint( "xc");
      n=n-90;
}
if (n>=50)
{
  soprint("L");
      n=n-50;
}
if (n>=40)
{
  soprint("xL");
      n=n-40;
}
while ( n>=10)
{
  soprint("x");
      n=n-10;
}
if ( n>=9)
{
  soprint("ix");
      n=n-9;
}
if(n>=5)
{
  soprint ("v");
      n=n-5;
}
if(n>=4)
{
  soprint("iv");
      n=n-4;
}
if( n>=1)
{
  soprint("i");
      n=n-1;
}
} // else
}//convert
} // BLC
class RomanDemo
{
  Public Static void main(String [] args)
  {
    if (args.length != 1)
    {
      sop("plz enter one value");
    }
  else
  {
    int x=Integer. ParseInt(args[0]);
      Roman r= new Roman();
      r.set(x);
```

```
r.convert();
} // else
} // main
} // ELC
```

4.7 Concepts Constructors in Java :-

In Java programming constructors will have a seperate role for in writing the programs.

Def :- A constructor is one of the special member method & it is automatically/ implicitly called by the JVM for placing during object creation time for placing our own values without placing default values.

Advantages of constructors :-

When we develop the Java application with the concept of constructors then such appls will get the following advantages.

1. constructors eliminates default values
2. constructors eliminates in calling ordinary methods.

In otherwords, the purpose of constructors concept is to initialize the object.

Rules/ Properties/ Characteristics of constructors :-

1. constructors will be called implicitly/ automatically during object creation time.

2. Each & every constructor name must be similar to class name.

3. constructors will not return any value even void also (if we write any return type then it will be treated as an ordinary method.)

4. Constructors should not be static (Becoz whenever an object is created constructors will be called each & every time)

5. Constructors of Java programming will not participate in inheritance process. (Becoz every constructor is meant for initializing its own class data members)

6. The access modifier (AM) of the constructor may or may not be private

a, If the AM of the constructor is <u>private</u> then an object of the corresponding class can be created in the same class context & it is not possible to creat in the context of some other class

(b) If the AM of the constructor is <u>not private</u> then we can creat an object of corresponding class in the same class context & in another class context.

Types of constructors in Java :- (88)

In Java programming we've 2 types of constructors They are 1. Default / Parameter less / zero argument constructor
2. parameterized constructor

6/7/15

Default constructor :-

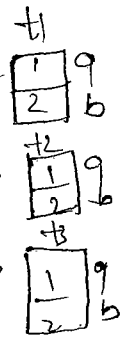
A constructor is said to be default if & only if it will not take any parameters. The purpose of The purpose DC is to creat multiple objects wrt Same class for placing same values.

SYNTAX :-

```
class (clsName)
{
    <classnam () // default constructor
    {
        Block of stmt(s)- -- initialization
    }
}
```

* Write a Java program which illustrate the concept of default constructor.

```
//TestDemo.Java
class Test
{
    int a,b;
    Test() //DC
    {
        sop("Test — DC");
        a=1;
        b=2;
        sop("val of a="+a);
        sop("val of b="+b);
    }
} // BLC
class TestDemo
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();
    } // main
} // ELC
```



Rules :-

1. Whenever we creat an object only with DC then defining the DC is optional.
2. whenever we creat an object with default constructor & if we are not defined a DC then JVM will call its own DC known as "sys defined DC"

and whose role is to place default values.

b, If we creat an object with DC & if we define the DC then JVM will execute programmer defined default constructor & places programmer defined values.

Parameterized Constructor :-

A constructor is said to parameterized if & only if constructor name is same always takes parameters.

The purpose of PC is that it always takes parameters & creat multiple objects wrt same class for placing different values.

SYNTAX :-

```
class <clsnames
{
    ------
    <clsname> (list of formal parameters)
    {
      Block of stmts(s) - initialization
    }
}
```

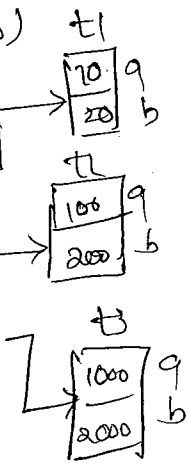
* Write a Java program which illustrate the concept of PC.

```
// TestDemo.Java
class Test
{
  int a,b;
  Test(int x, int y)
```

```
{
    sop("Test_DPC");
    a=x;
    b=y;
    sop("\val of a="+a);
    sop("\val of b="+b);
  } // Test

} // BLC
class TestDemo
{
  public static void main (String[] args)
  {
    Test t1 = new Test (10,20);
    Test t2 = new Test (100,200);
    Test t3 = new Test (1000,2000);
  } // main

} // ELC
```



Rule :- 2

When we creat an object with PC then defining PC is mandatory otherwise we get compile time error.

Rule :- 3

Whenever we creat multiple objects with both DC & PC then it is mandatory to the Java programmer to define both DC & PC. otherwise we get compile time error

NOTE :- If we are not initializing data members of any class wrt corresponding class constructors then the uninitialized data members will be initialized by its defined default constructor with default values.

Overloaded constructor:-

A constructor is said to be overloaded if & only if constructor name is same but signature is different.

Signature represents the following points.

a, No. of parameters
b, Type of parameters
c, Order of parameters

} Atleast one thing must be different

EX:-

```
Test t1 = new Test(10, 20);  -> 1
Test t2 = new Test(100);     -> 2
Test t3 = new Test(1.5f, 2.5f); -> 3
Test t4 = new Test(10, 1.5f);   -> 4
Test t5 = new Test(1.5f, 20);   -> 5
```

Here Test(...) is known as overloaded constructor.

Object parameterized constructor :-

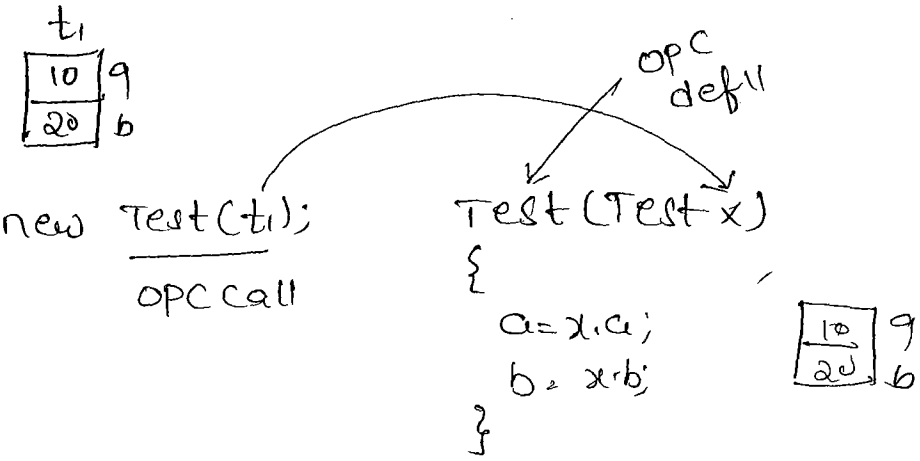
A constructor is said to be object parameterized if & only if it always takes object as a parameter.

The purpose of object parameterized constructor is that to copy the content of one object into another object where both the objects belongs to same type.

The functionality of OPC is resembling the functionality of copy constructor of C++.

Ex:-

Test t1 = new Test(10, 20);



*Write a Java program which illustrate the concept of DC, OPC & OPC

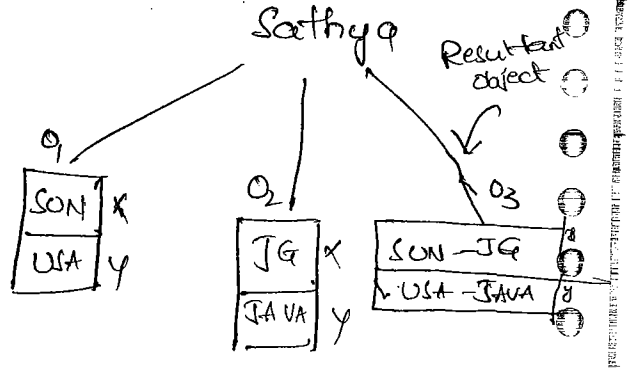
```
//TestDemo.Java
class Test
{
  int a,b;
  Test()
  {
    sop("Test ____ DC");
    a=1;
    b=2;
    sop("val of a = "+a);
    sop("val of b = "+b);
  }
```

```
Test (int x)
{
    SOP("Test ---- SPC");
    a=x;
    b=x;
    SOP("val of a="+a);
    SOP("val of b= "+b);
}
Test (int x, int y)
{
    SOP("Test ---- DPC");
    a=x;
    b=y;
    SOP("val of a= "+a);
    SOP("val of b= "+b);
}
Test(Test x)
{
    SOPC "Test ---- SingleOPC");
    a= x.a;
    b= x.b;
    SOP("val of a =" + a);
    SOP("val of b= "+b);
}
```

```
Test(Test x1, Test x2)
{
    sop("Test ——— Double op()");

    a = x1.a + x2.a;
    b = x1.b + x2.b;
    sop("val of a = "+a);
    sop("val of b = "+b);
}
} // BLC
class TestDemo
{
    public static void main(String [] args)
    {
        Test t1 = new Test();
        Test t2 = new Test(10,20);
        Test t3 = new Test(100);
        Test t4 = new Test(t2);
        Test t5 = new Test(t2,t3);
    } //main
} // ELC
```

* Write a Java program for compatinating the content of 2 objects which are containing string data.




```
//Sathopcd.Java
class Sathya
{
  String x, String y;
  Sathya(String s1, String s2) // PC
  {
    x=s1;
    y=s2;
  }
  Sathya (Sathya x1, Sathya x2) //OPC
  {
    x=x1.x+"_"+x2.x;
    y=x1.y+"_"+x2.y;
  }
  void disp()
  {
    sop("value of x="+x);
    sop("val of y="+y);
  }
} //BLC

class opcd
{
  Public static void main (String [] args)
```

```
{
    Sop("First object values");
        Sathya O1 = new Sathya("SUN", "USA");
            O1.disp();
    Sop("Second object values");
        Sathya O2 = new Sathya("JG", "JAVA");
            O2.disp();
    Sop("The Resultant object values");
        Sathya O3 = new Sathya(O1, O2);
            O3.disp();
} //main
} //ELC
```

Where
DC - Default constructor
PC - Parameterized constructor
OLC - Overloaded constructor

Points to be Remembered:-

1. By default every class of Java contains single default constructor (DC) known as "s/s defined DC".
2. Programatically, a class of Java can contain single DC defined by the programmer & multiple PC's known as "OLC's."
3. WKT a class of Java can contain collection of data members, constructors, methods etc & these details of the class are called profile. To view the profile/interface of a class we use a tool called Javap which is present in JDK\bin folder

SYNTAX:-

a. Javap . class file name

b. Javap fully qualified name of a class/ interface

Use Javap tool after compilation of the Program.

Def^n of Data Encapsulation:-

The process of hiding the confidential data from external programmers is called "data encapsulation."
(Data encapsulation provides Security to the confidential data)
Programatically, to implement data encapsulation we apply a keyword called Private. Private features of a class will not participate in profiling process.

Def^n of Data abstraction:-

The process of retreiving/ extracting essential details without considering hidden details is called "Data abstraction."

Q:- Some of the classes of Java does not contain constructor (True/ False)?

Ans:- False, becoz every class of Java contains constructors but the programmer of that class made the constructors defined as private.

8/7

Static Blocks in Java:- (99)

The purpose of static block is to initialize static data members only once.

→ Static block will execute only once when the class is loaded into main memory, before executing the main method.

→ SYNTAX :-

```
static
{
    block of stmt(s). initialization of static
                                data members
}
```

→ When we write static block & main method in a single Java program then JVM will execute static block first & later main method once.

→ Writing static block is optional

→ When we write SB & constructors in a Java program, then JVM will execute SB only once & constructors will be executed each & every time.

→ In SB, we can access/initialize only static data members but not instance data members where as both IDM & SDM are accessed in constructor.

→ In one Java class we can write any no. of static blocks but all those types of static blocks will be treated as single static block & gives the op in that order in which we write.

→ Whenever we initialize a SOM in the class then by default that class will contain system defined static block.

→ We can execute a Java program only by writing SB without writing main method upto JDK1.6 version. But it is not possible from JDK1.7 version.

* Write a Java program which illustrate the concept of static block in execution logic class.

```
// SBEX1.Java
class SBEX1
{
  static
  {
    SOP("I am from static block");
  }
  public static ~~main~~ void main(String [] args)
  {
    SOP("I am from main()");
  }
}
```

Output :-

I am from static block

I am from main()

* Write a Java Program which illustrate the concept of only the SB in a class without main method. (valid upto 1.6 & gives error in 1.7 onwards)

```
// SBEX2.Java
class SBEX2
{
  static
  {
    sop("I am from static block");
    ~~sop~~ system.exit(0);
  }
}
```

* Write a Java program which illustrate the concept of SB present in ELC & in BLC. by comparing with constructor & main method.

```
//SBEX3.Java
class Test
{
  static
  {
    sop("static block --- Test");
  }
  Test()
  {
    sop("Test --- DC");
  }
} //Test
```

```
class SBEX3
 {
  static
   {
     SOP(" static block ___ SBEX3");
   }
  Public static void main(String [] args)
   {
     SOP(" I am from main()");
       Test t1 = new Test();
       Test t2 = new Test();
   }
 }
```

<u>output :-</u>

```
Static block ___ SBEX3
 I am from main()
 Static block _ Test
  Test _ DC
  Test _ DC
```

* write a Java program which illustrate the concept of multiple Static blocks

```
// SBEX6.Java
 class SBEX6
  {
    SBEX6()
    {
      System.out.
```

```
static
 {
   sop("static block-1 — SBEX6");
 }
 public static void main(String[] args)
 {
   sop("i am from main()");
   SBEX6 s1 = new SBEX6();
   SBEX6 s2 = new SBEX6();
 }
 static
 {
   sop("static block-2 — SBEX6");
 }
 static
 {
   sop("static block-3 - SBEX6");
 }
} // SBEX6
```

<u>output</u>:-

```
static block-1 - SBEX6
static block-2 - SBEX6
static block-3 - SBEX6
i am from main()
SBEX6 — DC
SBEX6 — DC
```

*Write a Java program which illustrate the concept of initializing static data members in SB & IDM in constructor

```
//SBEX8.Java
class student
{
  int sno;
  String name;
  static String crs;
  static
  {
    sop(" student— static block");
    crs = "JAVA";
    sop("coarse for all = "+crs);
  }
  student (int sno, string sname)
  {
    sno = sno;
    name = sname;
    sop("Student Number = "+sno);
    sop (" student Name = "+name);
  }
} // student
class SBEX8
{
  public static void main (String [] args)
```

```
{
    Sop("First student details");
    Sop("=========================");
    Student S1=new Student(1, "Sathya");
    Sop("=========================");
    Sop("second student details");
    Sop("=========================");
    Student S2=new Student(2, "URAJU");
    Sop("=========================");
  }
}
```

* consider the following class deffination

```
class Sathya
{
  int a;
  Static int b=20;
}
```

The above class def" contains static data members initialization. Due to this the def" of Sathya class will contain default static block which is added by Java compilation environment.

Instance Block :- (IB)

→ The purpose of IB is exactly similar to DC of a particular class.

→ IB will execute each & every time whenever an object is created before executing the constructor & after executing the SB.

→ SB will execute only once when the class is loaded in main memory for initializing the SDM's whereas IB will be calling each & every time whenever an object is creating before the constructor & it is used for initializing IDM's of a class.

→ SYNTAX :-

```
{
    block of stmt(s) - initializing of IDM
}
```

→ we can write any no. of IB's but all of the IB's will be treated as single IB & they will be executed in which order we write.

* Write a Java program which illustrate the concept of SB, IB & DC present in ELC.

```
// IBEX1.Java
class IBEX1
{
```

```
{
  SOP("SB- IBEX1");
}
{
  SOP("IB — IBEX1");
}
IBEX1()
{
 SOP("IBEX1 —DC");
}
Public static void main(String [] args)
{
  SOP("i am from main()");
  IBEX1 io1 = new IBEX1();
  IBEX2 io2 = new IBEX1();
}
}
```

output :-

```
SB — IBEX1
~~IB - I~~
i am from main()

IB - IBEX1
IBEX1 - DC
IB - IBEX1
IBEX1 - DC
```

*Write a Java program (108) which will initialize the DMVs of a particular class with IB & constructor in the BLC.

```
// IBEX2.Java

class Test
{
  {
   static
   {sop("Test-SB");
  }

  {
   sop("Test-IB")
   a=10;
   b=10;
  }
 Test()
 {
  sop("Test-DC");
  a=100;
  b=200;
 }
 void disp()
 {
  sop("val of a =" +a);
  sop("val of b =" +b);
 }
} //Test-BLC
class IBEX2
{
 public static void main(String [] args)
```

```
{
  sop( "i am from main ()");
  Test t1= new Test();
  t1.disp();
}
}
```

output :-

```
i am from main()
Test -SB
Test -IB
Test -DC
a=100
b=200
```

* write a Java program which illustrates multiple SB's, multiple IB's & multiple constructors in both BLC & ELC.

```
// IBEX3.Java
class Sathya
{
  {
    sop("IB-1 - Sathya");
  }
  Static
  {
    sop("SB - Sathya");
  }
  {
    sop("IB-2 - Sathya")
```

```
{
  SOP("IB-3 ___ Sathya");
}
Sathya(int a)
 {
  SOP("Sathya-PC");
 }
} //Sathya-BLC
class IBEX3
{
 Static
  {
   SOP("SB-IBEX3");
  }
  {
   SOP("IB-IBEX3");
  }
 IBEX3()
  {
   SOP("IBEX3-DC");
  }
 public static void main(String [] args)
 {
   SOP("i am from main");
    Sathya S1=new Sathya(10);
    Sathya S2=new Sathya(20);
```

```
    IBEX3 io = new IBEX3();
  }
}
```

Output:-

```
SB - IBEX3
i am from main ( )
SB - Sathya
IB-1 - Sathya
IB-2 - Sathya
IB-3 - Sathya
Sathya - DC
IB-1 - Sathya
IB-2 - Sathya
IB-3 - Sathya
Sathya - DC
IBEX3 - IBEX3
IBEX3 - DC
```

NOTE:-

A class of Java can contain 5 parts they are

a, Data members
b, methods
c, Constructors
d, Static Block
e) Instance Block

2. Reading the data from the Keyboard :-

To read the data from the keyboard we use one pre-defined class called Scanner & it present in Java.Util Package. Star & it ~~pre~~ is available JDK1.5 Version onwards.

Reading the data from the keyboard is nthg but creating an object of scanner class

Profile of Java.Util.Scanner :-

Constructor

① Scanner (InputStream)

IM

② public xxx nextXxx ()

③ public String nextline

→ Constructor ① is used for creating an object of scanner class by passing an object of i/p Stream. An object of InputStream class call in Created as a SDM in another pre-defined class called System.

Ex:- Scanner S = new Scanner (System.in)

Here S : read actual data from keyboard

Provides an environment to read the data from keyboard

→ In method ② xxx represents any fundamental data type

method ① is used for reading fundamental data from keyboard.

→ Method ③ is used for reading any type of data in the form of string type.

* Write a Java program which will read 2 values from keyboard & find its sum.

```
//DataRead.Java
import Java.util.Scanner;
class DataRead
{
  Public Static void main(String []args)
  {
    Scanner s=new Scanner(system.in);
    Sop("Enter First Value");
     int a=S.nextInt();
    Sop("Enter Second Value");
     int b=S.nextInt();
     int c=a+b;
    Sop("Val of a= "+a);
    Sop("val of b= "+b);
    Sop("sum= " ~~a+b~~c);
  }
}
```

10.7 THIS :-

"THIS" is a keyword / implicit object / object created by JVM & supplied to each & every Java program for 2 purposes. They are

1. It always points to current class object
2. Whenever the formal parameters & data members of the class are same then JVM get ambiguity problems. (There is no clarity btw multiple duplicate things). Inorder, to differentiate btw data members of the class & formal parameters, the data members of the class must be preceeded by "THIS" keyword

SYNTAX

this. current class data member name

EX :-

* Write a Java program which illustrate the concept of "This" keyword.

```
// ThisDemo.Java
class Test
  {
    int a,b;
    Test( int a, int b)
    {
      this.a = a; // Here we are assigning formal parameter values to data members of class
```

```
this.a = this.a+2;
this.b = this.b+2;   } //Here we are incrementing the values of
                       data members of the class

a = a+1;
b = b+1;             } //Here we are incrementing the values of formal
                       Parameters

Sop("val of a in constructor (FP) = "+a);
Sop("val of b in constructor(FP) = "+b);   } //Here a,b refers to
                                              formal parameters
}
void disp()
{
Sop("val of a = "+this.a);
Sop("val of b = "+this.b);   } //Here writing "this" keyword
                                 is optional
}
} //Test-BLC
class ThisDemo
{
Public static void main (String Dargs)
{
~~This~~ t1 =
Test t1 = new Test(10,20);
    t1.disp();
Test t2 = new Test (100,200)
    t2.disp();
} //main
}//ThisDemo — ELC
```

output:-

```
val of a in constructor (FP) = 11
val of b in      "      (FP) = 21
val of a = 12
val of b = 22
```

```
val of a in constructor (FP) = 101
val of b in      "      (FP) = 201
val of a = 102
val of b = 202
```

Let us consider the following statement

```
obj1. methodName(obj2);
```

source/invoking object

Destination/target object

Whenever the stmt is executed method def^n receives source object values in the form of "this" & destination object values are available in the form of formal object.

Programatically, source object values can be reffered as

this. Data member of Source object

& destination object values are reffered as formal object. Data member of destination object

* Write a Java program which illustrate the concept of printing multiple object values with single method.

```
//ThisDemo1.Java
class Test
{
  int a,b;
  Test(int a, int b)
  {
    this.a=a;
    this.b= b;
  }
```

```
void disp(Test x) // x is called formal object
{
  sop("val of a of t1 = "+this.a);
  sop("val of b of t1 = "+this.b);
  sop("val of a of t2 = "x.a);
  sop("val of b of t2 = "x.b);
}
} //Test - BLC
class ThisDemo1
{
  public static void main (String [] args)
  {
    Test t1 = new Test(10,20);
    Test t2 = new Test(100,200);
      t1.disp(t2);
```

source object

destination object

```
  } //main
} // ELC
```

117

* Write a Java program which will add the content of 2 objects where each object contains 2 integer values

```
// objsum1.Java
import Java.util.Scanner;
class Test
{
  int a,b;
  Test()
  {
```

```
    a=b=0;
  }
  Test(int a, int b)
  {
    this.a=a;
    this.b=b;
  }
  Test sum(Test x)
  {
      Test t44=new Test();
      t44.a=this.a+x.a;
      t44.b=this.b+x.b;
        return t44;
    }
  void disp()
  {
    sop("val of a="+a);
    sop("val of b="+b);
  }

} //Test- BLC

class objsum1
  {
    Public static void main(String [] args)
```

```
// void sum(Test x1, Test x2)
// {
// this.a=x1.a+x2.a;
// this.b=x1.b+x2.b;
// }
```

"(or)

```
{
Scanner S=new Scanner(System.in);
sop("Enter First Value for t1");
  int x1=S.nextInt();
sop(" Enter second value for t1");
  int x2=S.nextInt();
sop("Enter First Value for t2");
  int x3=S.nextInt();
sop(" Enter Second Value for t2");
  int x4=S.nextInt();
Test t1=new Test(x1,x2);
Test t2=new Test(x3,x4);
Test t3=new Test();
  // t3=t1+t2;
  t3=t1.sum(t2); //Sol-1 (OR) // t3.sum(t1,t2); // Sol-2
sop("T1 value");
sop("=========");
  t1.disp();
sop("=========");
sop("T2 Values");
sop("==========");
  t2.disp();
sop("==========");
sop("T3 values");
sop("==========");
  t3.disp();
sop("==========");
} //main
} // ELC
```

NOTE :- Java programming never supports operator overloading but the aim of operator overloading is achieved in Java programming with the simple concept of methods.

NOTE :- Any type of operator can be applicable by default on fundamental data types & their values but not applicable by default on derived data types (arrays) & user defined data types.

Consider the following statement

$t_3 = t_1 / t_2$ — Error

Becoz operator / will not function btw t_1 & t_2

To address the above issue we've 2 solutions

sol : 1 t_3.div (t_1, t_2);

sol : 2 $t_3 = t_1$.div (t_2);

* Factory Method :-

Defn :- A method is said to be a factory method if & only if whose return type is similar to the class name in which class it presents.

The purpose of factory method is to creat an object without using new operator.

we've 2 types of factory methods. They are

1. Instance Factory method
2. Static Factory method

Return type Test ← Class name

↳ Test sum(---)

↑ Factory Method

* write a Java program which will multiply 3 objects contents where each object contains 3 values.

$t_4 = t_1 * t_2 * t_3$ — Invalid

Sol-1

t_4.Mul(t_1, t_2, t_3)

Sol-2

$t_4 = t_1$.Mul(t_2, t_3)

```
// ObjMul.Java
class Test
{
  int a,b,c;
  Test (int a, int b, int c)
  {
    this.a = a;
    this.b = b;
    this.c = c;
  }
  Test()
  {
  a=b=c=0;
  }
Test mul (Test x1, Test x2)
  {
Test ts5 = new Test();
  ts5.a = this.a * x1.a * x2.a;
  ts5.b = this.b * x1.b * x2.b;
```

```
void disp ()
{
  sop("Val of a="+a);
  sop("Val of b="+b);
  sop("Val of c="+c);
}
}//Test_BLC
clas objMul
{
 public static void main(String D args)
{
 Scanner s=new Scanner(system.in);
  sop("Enter First value for t1");
     int x1= s.nextInt();
  sop("Enter second values for t1");
     int x2=s.nextInt();
  sop("Enter First value for t2");
     int x3=s.nextInt();
  sop("Enter second value for t2");
     int x4=s.nextInt();
  sop("Enter First value for t3");
     int x5=s.nextInt();
  sop("Enter second value for t3");
     int x6=s.nextInt();
```

```
Test t1 = new Test(x1, x2);
Test t2 = new Test(x3, x4);
Test t3 = new Test(x5, x6);

  // t4 = t1 * t2 * t3;
  t4 = t1.Mul(t2 * t3);
```

← without using new operator we can creat an object

```
Sop("T1 Values");
Sop("=======");
    t1.disp();
Sop("=======");
Sop("T2 Values");
Sop("=======");
    t2.disp();
Sop("=======");
Sop("T3 Values");
Sop("=======");
    t3.disp();
Sop("=======");
Sop("T4 Values");
Sop("=======");
    t4.disp();
Sop("=======");

}

}
```

* Inheritance :-

The importance of inheritance in s/w development is to build Scalable appls (high performance, consistent results, Effective memory management).

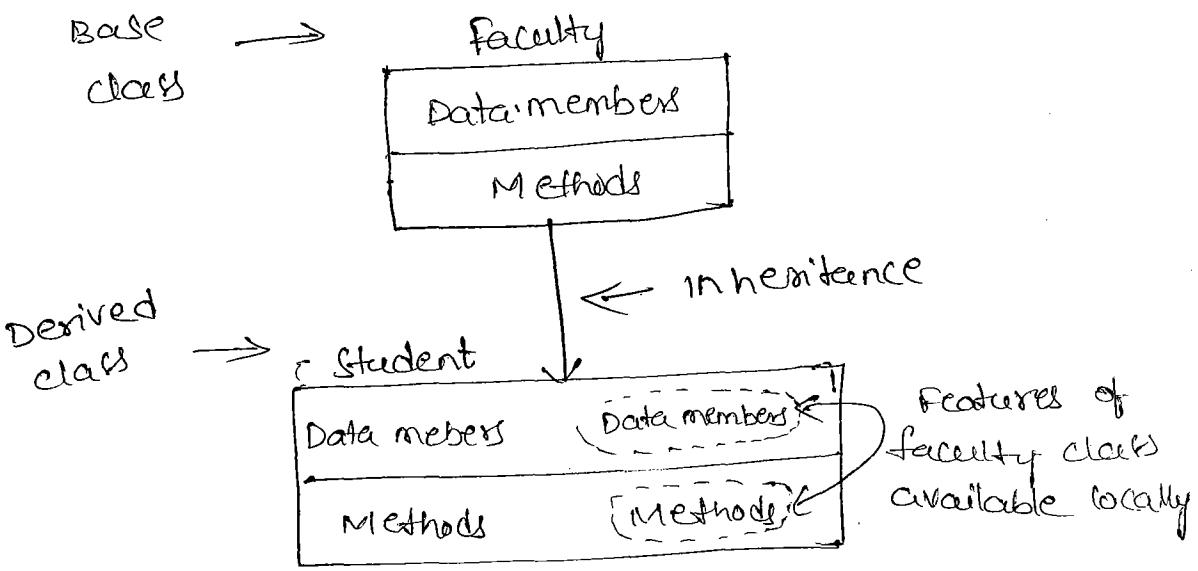
Def :- The process of obtaining datamembers & methods (Features) of one class in to another class is called "Inheritance".

→ The class which is giving data members & methods to another class is called base/parent/super class.

→ The class which is getting data members & methods from another class is called derived/sub/child class.

→ The inheritance concept is also known as re-usability/sub classing/derivation/extendable classes.

consider the following diagram.



In general, all the features of base class available logically in the context of derived class.

* Inheritance concept always follows logical memory management.

in the derived class ~~class~~ without taking any physical memory space. In otherwords, logical memory management says "Feeling that base class features are existing without taking any memory space".

*Advantages/ Benifits of Inheritance/ Reusability:-

If we develop any Java appl with the concept of inheritance then such appls will get following advantages.

1. Application development time is less
2. Application memory space is less
3. Application execution time is less
4. Application performance is enhanced (improved)
5. Redundancy (Repeat) of code is minimized
6. We are able to get the slogan of Java i.e. WORA.

*Inheritance types/ Reusable Techniques:-

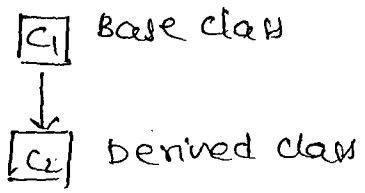
The patterns / models which makes us to understand how to inherite the features of base class into derived class. we've 5 inheritance types. They are

1. Single Inheritance
2. Multilevel Inheritance
3. Hirarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

1417 * Single Inheritance :-

Def :- In this inheritance there exist single base class & single derived class.

Diagram :-

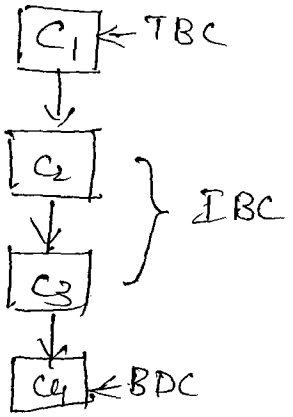


Inheritance Path :- $C_1 \rightarrow C_2$

* Multilevel Inheritance :-

Def :- In this inheritance there exist single base class, multiple intermediate base classes & single derived class.

Diagram :-



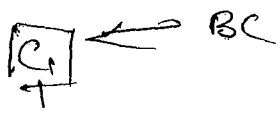
Inheritance Path :- $C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$

Def of IBC :- An IBC, in one context it acts as BC & in another context same class acts as DC.

* Hierarchial Inheritance :-

Def :- In this inheritance there exist single base class & multiple derived classes.

Diagram :-



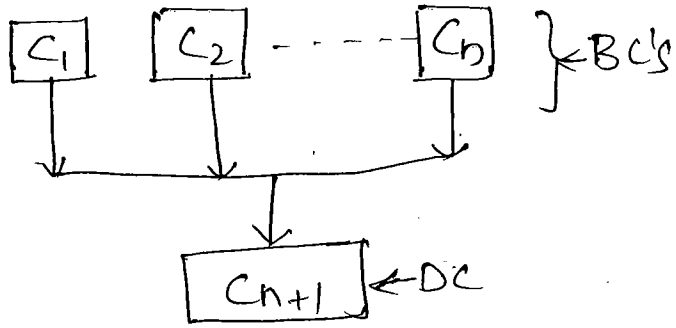
Inheritance path :- $C_1 \rightarrow C_2$, $C_1 \rightarrow C_3$ ---- $C_1 \rightarrow C_n$

Single, multilevel & Hierarchial inheritances are supported by In Java. By the concepts of classes & interfaces.

*Multiple Inheritance :-

Def :- In this inheritance there exist multiple base classes & single derived class.

Diagram :-



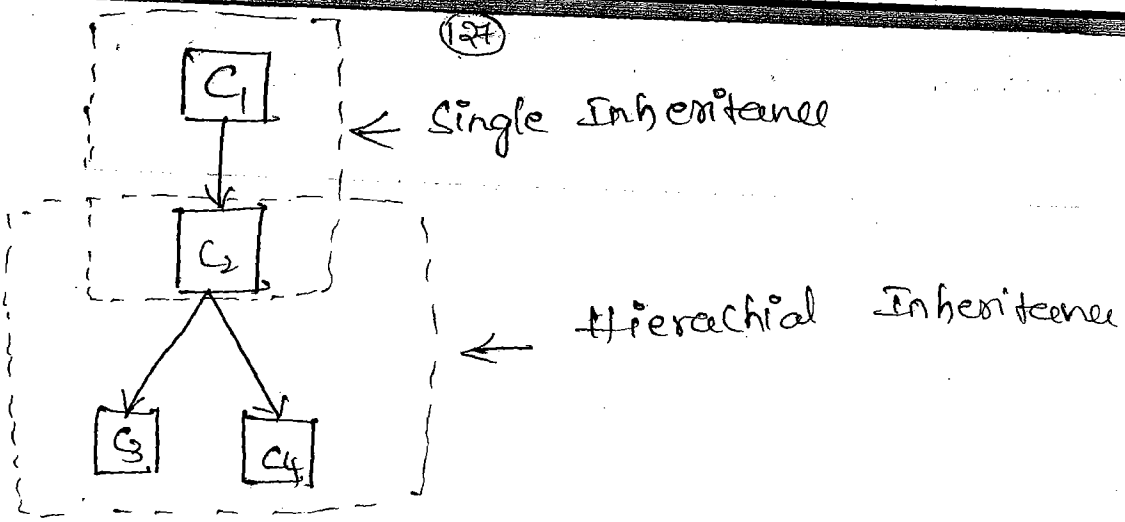
Inheritance path :- $C_1 \rightarrow C_{n+1}$, $C_2 \rightarrow C_{n+1}$ ------ $C_n \rightarrow C_{n+1}$

Multiple Inheritance is not supported in Java by the concept of classes becoz of ambiguity problems. But, this inheritance type is supported by the concept of interfaces.

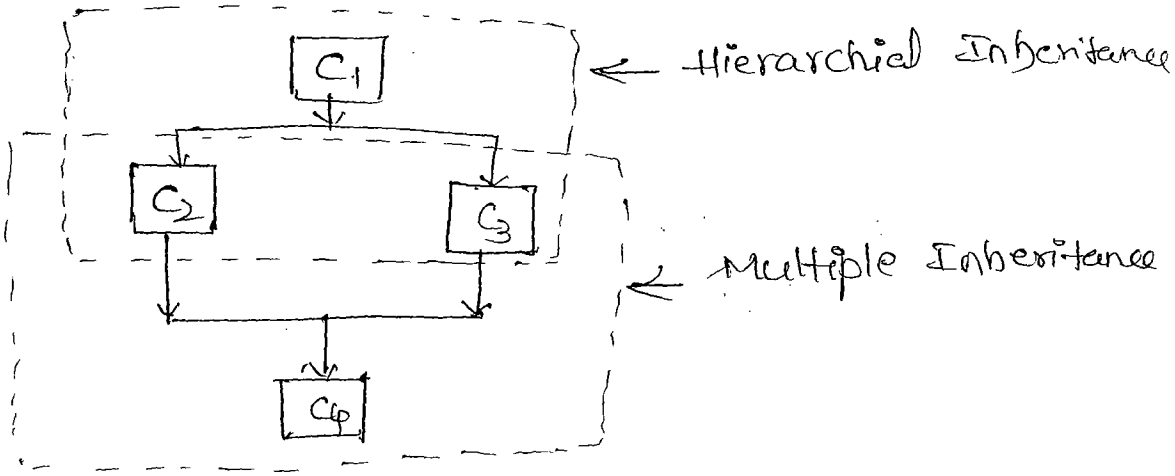
*Hybrid Inheritance :-

Hybrid Inheritance = combination of any available Inheritance types

In the combination, if the one of the combination is multiple inheritance then the entire combination (Hybrid Inheritance) is not supported by classes but it can be supported by interfaces.



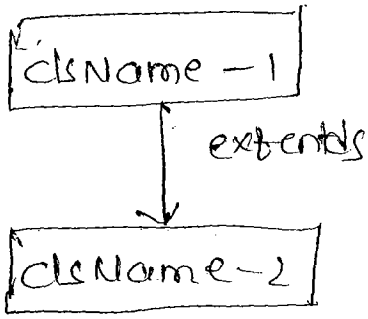
Valid by classes & interfaces



Invalid by classes & valid by interfaces

Inheriting the features of base class into derived class:-

To inherite the features of BC into DC we use the following SYNTAX.



SYNTAX :-

class <clsName-2> extends < cls Name-1>
{ variable declaration;

In the above SYNTAX (28)

1) <clsName-1> & <clsName-2> represents name of the base & derived classes.

2) Extends is a keyword which is used for inheriting the features of base class into derived class plus it provided richest functionality to derived class.

3) In Java programming one DC can extends only one BC becoz Java programming never supports multiple inheritance ~~trae~~ ~~the~~ through the concept of class as but it can be supported through the concept of interfaces.

4) Whenever we develop any inheritance based appn, it is always recomended to the Java programmer to creat an object of BDC becoz it contains the features of TBC (top most base class) & intermediate base class (IBC)

5) Whenever we creat an object of bottom most derived class (BDC), first we get the memory space for data members of TBC, second we get the memory space for data member of IBC & at last we get the memory space for Data members of BDC. All the features of TBC & IBC available logically in BDC. → the context of (logical memory management).

15/17

6. If we don't want to give the features of base class to the derived class then the defn of the base class must be made as final. Hence, final base classes never participates in inheritance. Final features of the class can be inherited into derived class

But those values can not be modified.

7. If we don't want to give some of the features of base class then such features of the base class must be made as private. Hence, Private features of base class never parcepates in inheritance (or) never visible/accessable in the context of derived class. Java programming does not contain just private classes (Inner private classes can be permitted)

8. In Java programming, data members & methods of a base class can be inherited into derived class but constructors, static blocks & instance blocks of base class will not be inherited into derived class.

9. <u>Scope of base class object</u>:-

An object of base class can access the features of same class but an object of base class can't access the special features which are present in the context of its derived class.

10. For each & every class in Java there exist an implicit pre-defined super class called Java.lang.object becoz it provides garbage collection facility (finalize()) for collecting un used memory space for improving the performance of Java based appls.

* Write a Java program which illustrate the concept of inheritance.

```
//Inhprog1.Java
 class BC
 {
    int a;
    void disp( )
    {
      sop("base__disp( )");
    }
 } //BC
 Class DC  extends BC
 {
   int b;
   void show( )
   {
    sop("derived __ show( )");
     a=10;
     b=20;
    sop("Val of a from BC="+a);
    sop("Val of b from DC="+b);
   }
 } //DC
Class inhprog1
{
  Public static void main (String [] args)
  {
    sop("w.r.t. DC");
    DC dol= new DC( );
      dol.disp( );
```

```
    sop("w.r.t BC");
      BC bo1 = new BC();
        bo1.disp();
    // bo1.show(); Invalid, beoz show() does not exist in BC
  }
}
```

NOTE:- If any class of Java contains Private features (Data members & methods) then those features can be accessable in the context of same class with the same class object but not accessable in the context of some other class.

* Develop the Java Program for the following problem statement.

1. There exist a university which contains university name, location & coarse. Use the appropriate methods for reading & displaying university details.

2. There exist a college which contains college Name, location & Students details like no, name & doing which coarse. Use appropriate methods for getting & displaying clg & student details along with unillersity details.

```
// Inhprog2.Java
import Java.util.Scanner;
class univ
{
  String uname;
  int cs;
```

```
Scanner s = new Scanner(System.in);
void getUnivDetails()
{
  sop("Enter univ Name");
    uname = s.nextLine();
  sop("Enter univ Loc");
    loc = s.nextLine();
  sop("Enter univ course");
    crs = s.nextLine();
}
void dispUnivDetails()
{
  sop("=========");
  sop("univ Details");
  sop("=========");
  sop(uname+"  "+crs+"  "+loc);
  sop("=========");
}
} // univ — BC
class college extends univ
{
  String cname;
  String cloc;
  int stno;
  String sname;
```

```
Void getCollegeDetails()
{
  Sop("Enter UR college Name");
    cname=S.nextline();
  Sop("Enter UR college Loc");
    cloc=S.nextline();
  Sop("Enter UR Name");
    Sname=S.nextLine();
  sop("Enter UR Number");
   Stno=Integer.parseInt(S.nextline());
   get UnivDetails();
}

Void dispcollegeDetails()
{
   dispUnivDetails();
   Sop("= = = = = = = = = = = = = = = =");
   Sop("college Details");
   sop("= = = = = = = = = = = = = = = =");
   sop("UR Name="+Sname+" UR Number="+Stno);
   Sop("UR college Name="+cname+" UR loc="+cloc);
   Sop("= = = = = = = = = = = = = = = =");
}

} // college — DC
```

```
class Inhprog2
{
    Public static void main(String [] args)
    {
        college co= new college();
            co.getCollegeDetails();
            co.dispCollegeDetails();
    }
}
```

* Devolop a Java program for the following problem stmt

Let us assume there exist a company & it contains company Name, location & type of operations & use the appropriate methods for getting & displaying company details.

let us assume there exist a permanent employee which contains employ no, employ name, basic salary & designation. company gives ~~DA~~ DA has 40% basic salary, ~~DA~~ TA as 20% of basic salary CCA = 500 & deductions like GPF as 5% of basic salary, LIC as 5% of basic salary. Calculate gross salary of employee. [Gross salary = (basic salary + DA + TA + CCA) − (GPF + LIC)]

Let us assume there exist adhoc employee & it contains employ no, name, no. of days working & wage rate per day - Calculate the total salary of adhoc employee)

[Total salary = No of working days * rate per day]

Generate the pay slips of permanent employee & adhoc employee with employee details along with company details.

* Types of relationships in Java:-

The purpose of relationships in Java is that how to use the features of one class into another class.

In Java programming we've 3 types of relationships. They are

1. IS-A Relationship
2. has-A Relationship
3. uses-A Relationship

* IS-A Relationship :-

In this relationship one class is obtaining the features of another class by using inheritance concept with extends keyword.

Ex:-

```
class C1
{
  ____
  ____
}
class C2 extends C1
{
  ____
  ____
  ____
}
```

Here the relationship always between C_1 & C_2 classes is "IS-A".

IS-A relationship always follows "logical memory management".

The limitation of "IS-A" relationship is that unable to eliminate the ambiguity problems which are occuring in multiple inheritance. To avoid the above problem we use the concept of "has-A" relationship.

*has-A Relationship :-

An object of one class is created as a data member in the context of another class.

```
EXT class C1
    {
    ___
    }
    class C2 <-- has-A
    {
    C1 O1 = new C1();
    ___
    }
```

Here the relationship btw C1 & C2 is "has-A".

"Has-A" relationship always follows physical memory management

we refers to use "has-A" relationship when their exist.

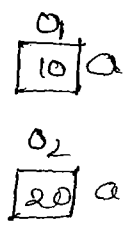
Ambiguity problems in multiple inheritance.

Consider the following invalide code segment

```
class C1
{
int a=10;
}
class C2
{
int a=20;
}
class C3 extends C1, C2   // Invalid (or) error
{
___
}
```

To overcome the above problem we apply "has-A" relationship and we can rewrite the above code as follows.

Ex:-
```
class c1
{
 int a=10;
}
class c2
{
 int a=20;
}
class c3
{
 int c;
 c1 o1=new c1();
 c2 o2=new c2();
 ==
 c = o1.a+ o2.a;
 ==
}
```



*Uses-A relationship:-

A method of one class is using an object of another class

Ex:-
```
class c1
{
 ==
}
class c2
{
 void disp()
 {
  c1 o1=new c1();
  ==
 }
```

uses-A

∴ c2 is uses-A

Note:1

The default relationship in Java is "IS-A" becoz for each & every class in Java there exist an implicit pre-defined super class called Java.lang.Object.

Note:2

The universal example for "has-A" relationship is System.out (Here out is an object of PrintStream) & it is created as a static data member in another class called System. The relationship btw system & printstream class is known as "has-A" relationship.

Note:3

The universal example for "uses-A" relationship is that execution logic method of execution logic class is using an object of business logic class.

* "Super Keyword":-

Whenever we inherite the base class features into derived class, there is a possibility that base class features are similar to derived class features & JVM gets an ambiguity.

In order to overcome the ambiguity problems (or) to differentiate btw base class features & derived class features. In the context of derived class the base class features must be precided by super keyword. Otherwise such features will be treated as current class/derived-class features.

SYNTAX:-

super. Base class Feature Name

Super Keyword is playing an important role in 3 places They are

1. At variable level
2. At method level
3. At constructor level

* Super at variable (Data member) level :-

Whenever we inhirite the base class data members into derived class, there is a possibility that base class data members are similar to derived class data members & JVM gets an ambiguity

In order to differentiate btw BC data members & DC datamembers in the context of DC the BC data members must be preceded by a "Super" keyword.

SYNTAX :- Super <Base class data members name>;

If we don't write super keyword before the BC data member name then it is considered as current class data member & it is hidden in the context of DC.

EX :- Write a java program which illustrate the concept of super keyword at variable level.

```
//SDemo.Java
class BC
{
   int a;
}
class DC extends BC
{
   int a, c;
   void set(int x, int y)
   {
      super.a = x;
      this.a = y;
   }
   void add()
   {
      c = super.a + this.a;
   }
   void disp()
   {
      sop("val of a (BC) = "+ super.a);
      sop("val of a (DC) = "+ this.a);
      sop("sum = "+c);
   }
}
class SDemo
{
   public static void main(String [] args)
```

```
    DC dO1 = new DC();
    dO1.set(Integer.parseInt(args[0]), Integer.parseInt(args[1]));
    dO1.add();
    dO1.disp();
  }
}
```

Super at method level :-

Whenever we inherit base class methods into DC, there is a possibility that base class methods are similar to DC methods & JVM gets an ambiguity.

To differentiate btw BC methods & DC methods, in the context of DC methods BC methods can be reffered by using super keyword.

Syntax :- super<base class method name>;

If we don't write super keyword before the base class method name then JVM will call the DC method infinitely & base class methods are hidden in the context of DC.

NOTE :- Any method called by itself infinitely then stack memory becomes full & such type of error is known as Stack OverFlow Error.

* Write a Java program which illustrate the concept of super keyword at method level.

```
// SDemo1.Java
class BC
{
  void disp()
  {
    sop("Base class-disp()");
  }
}

class DC extends BC
{
  void disp()
  {
    sop("Derived class - disp()");
    super.disp(); // call super class disp() from the context of DC
  }
}
```

```
class SDemo1
{
  Public static void main(String [] args)
  {
      DC do1= new DC();
          do1.disp();  // will call derived class disp()
  }
}
```

Q:- What is the difference btw 'this' & 'super' keyword?

Ans:- 'this' keyword always points to current class features
'super' keyword always points to super class features.

Super at method level:-

Whenever we inherit base class methods into DC, there is a possibility that base class methods are similar to DC methods & JVM gets an ambiguity.

TO differentiate btw BC methods & DC methods in the context of DC methods base class method can be reffered by using 'super' keyword

SYNTAX:- super < baseclass method name >;

If we don't write 'super' keyword before the BC methodname then JVM will call the DC method infinitely & BC methods are hidden in the context of DC.

NOTE:- Any method called by itself infinitely then stack memory becomes full & such type of error is known as StackOverflowError.

* Write a java program which illustrate the concept of 'super' keyword at method level.

```
//Sdemo.java
class BC
{
  void disp()
  {
```

```
    sop("base class: disp()");
  }
}
class DC extends BC
{
    void disp()
    {
      sop("Derived class: disp()");
        super.disp(); // Call super class disp() from the context of DC.
    }
}
class Sdemo
{
  Public static void main(String args[])
    {
      DC dc1 = new DC();
          dc1.disp(); // will call DC disp()
    }
}
```

Super at method overridden level:-

WKT method overridding is equal to method heading is same plus method body is different.

To implement method overridding concept we need to use inheritance concept.

Original methods always presents in BC's & overridden methods always presents in DC's.

* Write a Java program which illustrate the concept of super Keyword at method overridden level.

```
//Sdemo.java
class BC
{
    void op(int x, int y)
```

```
    sop("op() : original : BC");
    int z=x+y;
    sop("sum in BC="+z);
  }
}
class IBC extends BC
  {
    void op(int x, int y)
     {
       super.op(x,y);
       sop("op() : overridden : IBC");
       int z=x-y;
       sop("sub in IBC="+z);
     }
  }
class DC extends IBC
 {
   void op(int x1, int x2)
    {
      super.op(x1, x2);
      sop("op() : overridden : DC");
      int z=x1*x2;
      sop("mul in DC="+z);
    }
 }
class Sdemo
 {
  public static void main(String [] args)
   {
     DC do1=new DC();
     do1.op(10,20);
       (or)
     //new DC().op(Integer.ParseInt
      (args[0]), Integer.parseInt(args[1]));
   }
 }
```

In the above program new DC() is known as nameless/anonymous class object

Q. How do you call BC methods from the context of DC methods where both the methods are same?

Ans:- super<base class method name>

The above stmt must be written in the context of DC.

Q:- How do you call original method from the context of overridden methods?

Ans:- The super<original method name>

The above stmt must be written in the context of overridden method of DC

Whenever we apply super keyword at variable & method level then it can be used btw immediate BC & DC's but not btw bottom most DC. super keyword either at variable level (or) method level can be applied at any one level.

super.super.methodname is not valid.

Super at constructor level:-

Whenever we develop any inheritance appl it is highly recommended to create an object of bottom most DC becoz it obtains features from top most BC & intermediate BC.

When we create an object of bottom most DC, first we get the memory space for data members of top most BC, 2nd we get memory space for data members of intermediate BC & at last we get memory space for data members of bottom most DC

In whatever the order memory space is created in the same order data members will be initializes (otherwise we get compile time error) i.e., first top most BC data members will be initialized second intermediate BC data members will be initialized & at last bottom most DC data members will be initialized. WKT initializing the data members requires constructors concept.

Hence, in any inheritance appl development, constructors are calling from bottom to top & execution is from top to bottom

In order to establish communication btw BC constructors & DC constructors we have two implicit functions. They are

1. super()
2. super(- - - -)

<u>super()</u> :- It is used for calling super class default constructor from DC constructors.

<u>super(- - - -)</u> :- It is used for calling super class parameter as constructor from DC constructors.

Rules :-

Whenever we use either super() (or) super(- - - -) in the DC constructors, they must be used as a first executable stmt. Otherwise we get compile time error (becoz before executing DC constructors first we must execute super class constructors)

Possibilities of using super() & super(- - - -) :-

The following diagram gives possibilities of using super() & super(- - -)

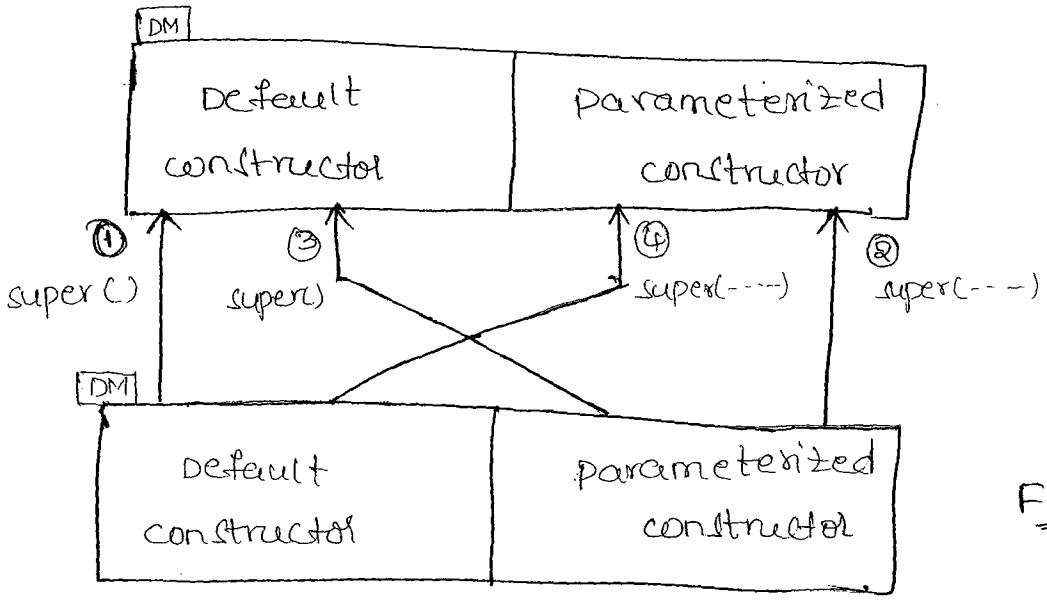


Fig :-1

Rules ① & ③:-

Whenever we want to call base class default constructor from the context of DC constructor in the context of DC constructor we use super().

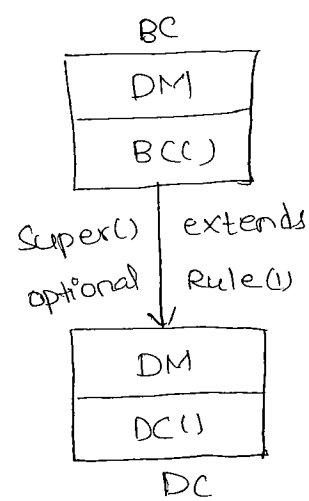
Using super() in the context of DC constructor is optional becoz BC will contain single form of default constructor

Rules ② & ④:-

Whenever we want to call BC parameter as a constructor from the context of DC constructor in the context of DC constructor we must use super(----).

Using super(----) in the context of DC constructors is mandatory becoz a BC may contain multiple parameterized constructor

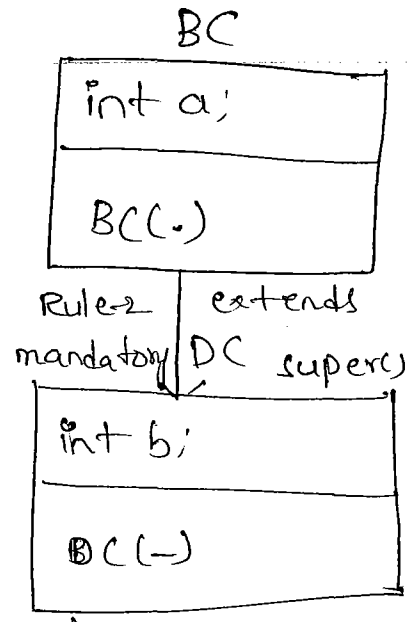
*Write a java program which will illustrate Rule① of fig①.




```
class BC
{
   int a;
   BC()
   {  // ---- ②
   {
      sop("BC ⇒ Default constructor");
      a=10;
      sop("val of a from BC ="+a);
   }
}
class DC extends BC
{
   int b;
   DC() // ---- ①
   {
      super();//optional - control goes to ②
      sop("DC ⇒ Default constructor");
```

```
    b=20;
    sop(" val of b from DC = "+b);
  }
}
class sdemo
{
  private static void main(String [] args)
  {
    new DC();  // control goes to ①
  }
}
```

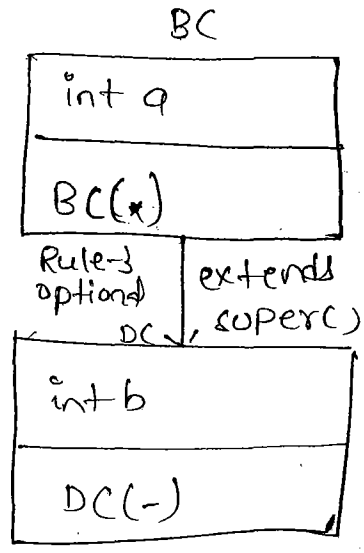
*write a Java program which illustrate rule-2 of fig (a)




```
//Inhprog6.Java
class BC
{
  int a;
  BC(int a) // ------(2)
  {
   sop("Base ----PC");
   this.a=a;
   Sop("val of a from BC="+this.a);
  }
} //BC
class DC extends BC
{
  int b;
  DC(int a,int b) // ------(1)
  {
   super(a); //mandatory ---control goes to (2)
   sop("derived ----PC");
   this.b=b;
   sop("val of b from DC="+this.b);
  }
} //DC
class Inhprog6
{ public static void main(String[] args)
```

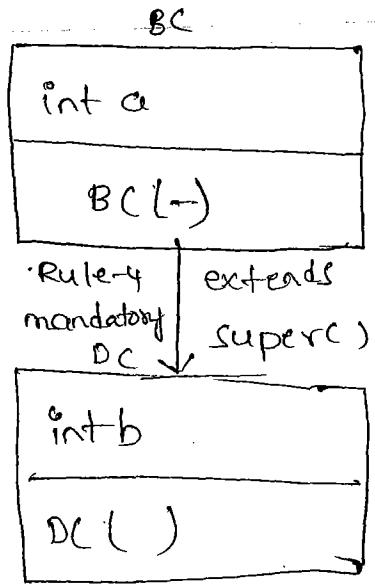
21/7

*Write a Java program which illustrate rule (3) of fig (1).




```
//Inhprog7.Java
class BC
{
  int a;
  BC()// ------ (2)
  {
    sop("Base ----BC");
    a=100;
    sop("Val of a from BC="+a);
  }
}//BC
class DC extends BC
{
  int b;
  DC(int b) // ----- (1)
  {
    super(); // optional --- control goes to (2)
    sop("derived ---- DC");
    this.b=b;
    sop("Val of b from DC="+this.b);
  }
} // DC
class Inhprog7
{
  public static void main(String [] args)
  {
    DC dc1=new DC(200); // control goes to (1)
```

* Write a Java program which illustrate rule-4 of fog()




```
// Inh prog & Java
class BC
{
  int a;
 BC(int a) // ---- (2)
 {
   sop("Base --- pc");
   this.a = a;
   sop(" val of a from BC = "+this.a);
 }
} //BC
class DC
{ int b;
  DC() // ------ (1)
  {
   super(); // mandatory ... control goes to (2)
   sop("derived --- DC");
   b=2;
   sop(" val of b from DC = "+b);
  }
} //DC
class Inhprog 8
{
  Public static void main(String [] args)
  {
    DC do1= new DC();
```

In all the above examples constructors are calling bottom to top[increasing order(1) ---> (2)] & executing from top to bottom [Decreasing order (2) ---> (1)].

*this() & this(...) :-

this() (or) this(...) are the two implicit fcns created by JVM & supplied to each & every java program for the purpose of establishing the com.n btw current class constructors.

1. this() :- It is used for calling current class default constructor from current class parameterized constructor
2. this(...) :- It is used for calling current class parameterized constructor from current class other category constructors.

*Rules :-

① When we use this() (or) this(...) in the current class constructors, they must be used as first executable statment. Otherwise we get compile time error.

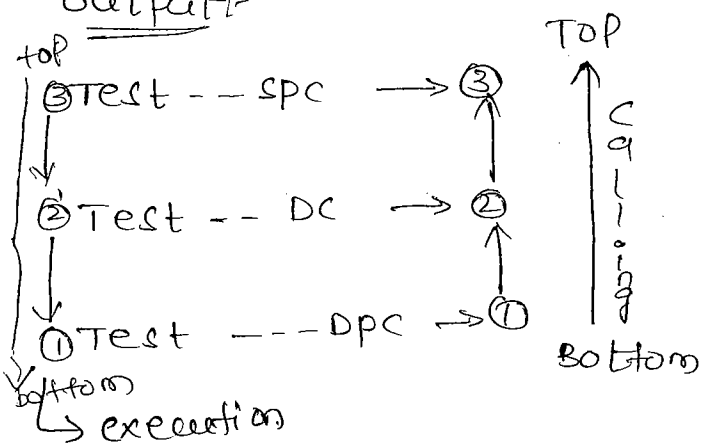
② When we use this() or this(...) in the current class constructors, it is recomended to form path but not recomended to form cycles. Becoz cycles always generates ~~constr~~ recursive constructor invocation problem.

* Write a Java program (12) which illustrate the functionality of this() & this(...).

```
//ThisDemo2.Java
class Test
{
 int a,b;
 Test() //-----(2)
 {
  this(100); //control goes to (3)
  sop("test --de");
  a=1;
  b=2;
  sop("val of a = "+a);
  sop("val of b = "+b);
 }
 Test(int x) //-------(3)
 {
  sop("test----spc");
  a=b=x;
  sop("val of a = "+a);
  sop("val of b = "+b);
 }
 Test(int a, int b) //--------(1)
 {
  this();//control goes to(2)
  sop("test -- DPC");
  this.a=a;
  this.b=b;
```

```
    SOP("val of a ="+this.a);
    SOP("val of b = "+this.b);
  }
} //Test
class ThisDemo2
{
  Public static void main(String [] args)
  {
    Test t1 = new Test(10,20); // control goes to (1)
  }
}
```

output:-



→ Hence even in the current class constructors com.n, const current class constructors are calling from bottom to top & executing from top to bottom

Q:- What is the difference btw this & super Keyword.

Ans:- this Keyword is always used to reffer current class object & differentiating btw data members & formal paramenters. where as super Keyword is always pointing base class features & it differentiates

BC features & DC features are same

Q2:- What is the difference b/w this(), this(...) & super(), super(...)?

Ans:- this() & this(...) are used for establishing com.n b/w current class constructors where as super() & super(...) is used for establishing com.n b/w BC & DC constructors.

NOTE:- Whenever we come across multiple this() (or) this(...) then we get compile time error. according to rule of this & with my intelligence I can address multiple this functions issue.

NOTE:- one can also solve multiple this functions & super functions & multiple super functions but we can't solve super & this functions becoz super class constructors can't call derived class constructors.

this - this — allow
this - super — allow
super - super — allow
super - this — Not allow.

23/7

Polymorphism :-

Polymorphism is one of the distinct principle of OOPS & whose basic aim is to build scalable apps (high performance + less memory + consistant result)

Def :-

The process of representing "one form in multiple forms" is called Polymorphism.

→ In the defn of polymorphism, one form represents original method & always resides in BC. Multiple forms represents over ridden methods & always resides in DC's.

→ Polymorphism is one of the principle but not a programming concept. All the OOPL's will implement polymorphism principle by various programming concepts

→ In Java programming, polymorphism principle implemented by using 2 approaches. They are

a. Method over riding
b. Method over loading

→ The polymorphism principle is having 2 sub principles. They are

a. Static / compile time polymorphism
b. Dynamic / run time polymorphism

} Types of polymorphism

→ Java programming follows only dynamic polymorphism but not static polymorphism due to its limitations.

→ In real world appls Polymorphism principle is always used for developing <u>business logic</u>

consider the following diagram

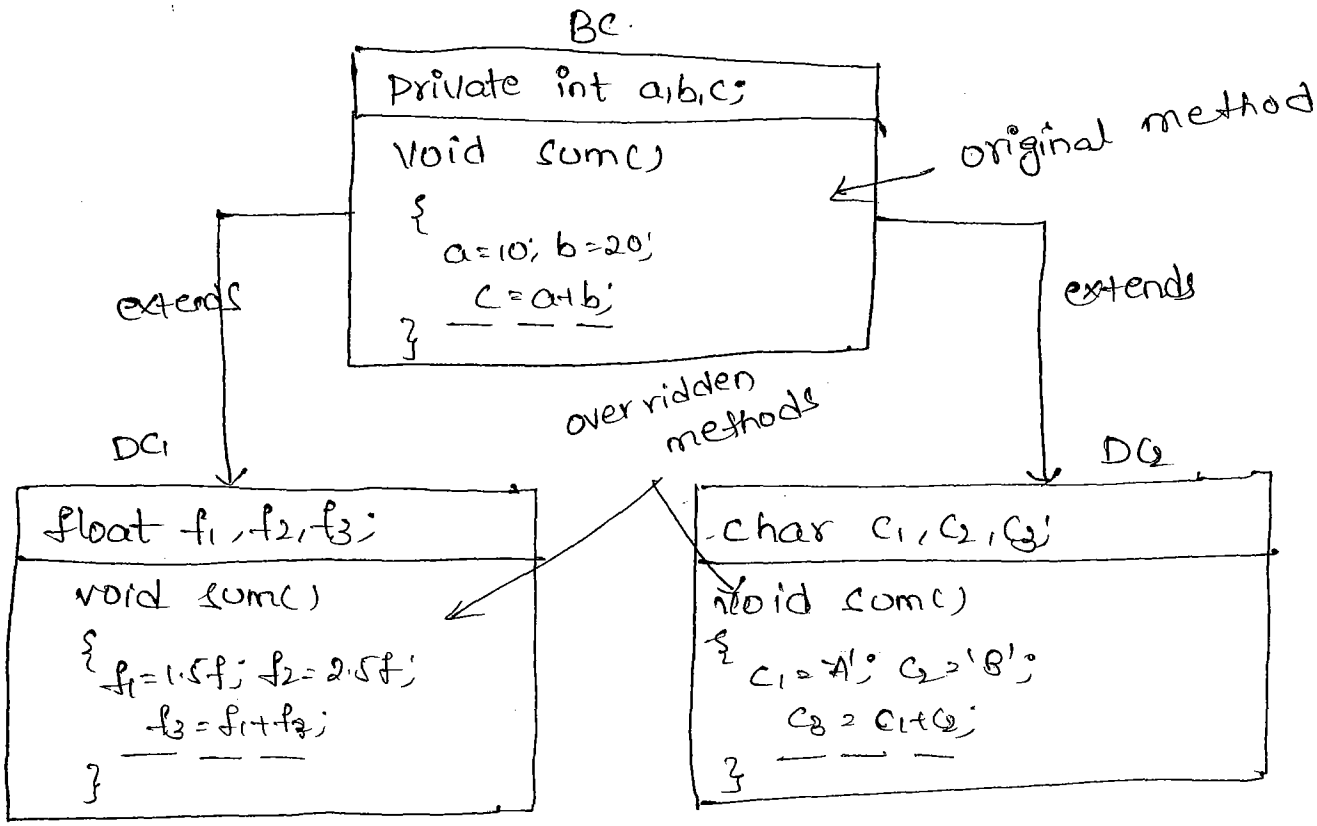


Fig (1)

In the above diagram the sum method is available <u>in one form originally</u> & it is further implimented in multiple forms by over riding original method & hence sum method is called polymorphic method.

The classes DC1 & DC2 are inheriting the features from BC class & hence the business logic is developed by following polymorphism principle along with method over riding in the form of dependent business logic classes (BLC)

Dynamic Binding :-

Dynamic binding is one of the distinct principle in oops which helps us to develop effective execution logic.

→ Dynamic binding always says " don't creat the object of DC's but creat the object of BC".

→ We always use dynamic binding for executing polymorphic appls.

→ The advantages of polymorphism along with method overriding & dynamic binding principle are

(a) application takes less memory space.

(b) application takes less execution time.

(c) application performance is enhance (improve)

→ Def of Dynamic binding :-

The process of binding appropriate versions (overridden methods) of DC's which are inherited from BC with BC object is called dynamic binding.

→ When we use dynamic binding principle at the time of using polymorphic appls then JVM will consider internally the following 2 points :

(a) What type of object?

(b) What type refference object contains?

In otherwords, when we creat an object indirectly (following dynamic binding principle) then JVM will consider the above 2 points.

NOTE :- To creat an object indirectly by using dynamic binding principle, there must exist super-sub/Base-Derived relationship (is-A relationship)

EX:- Parent p = new Child

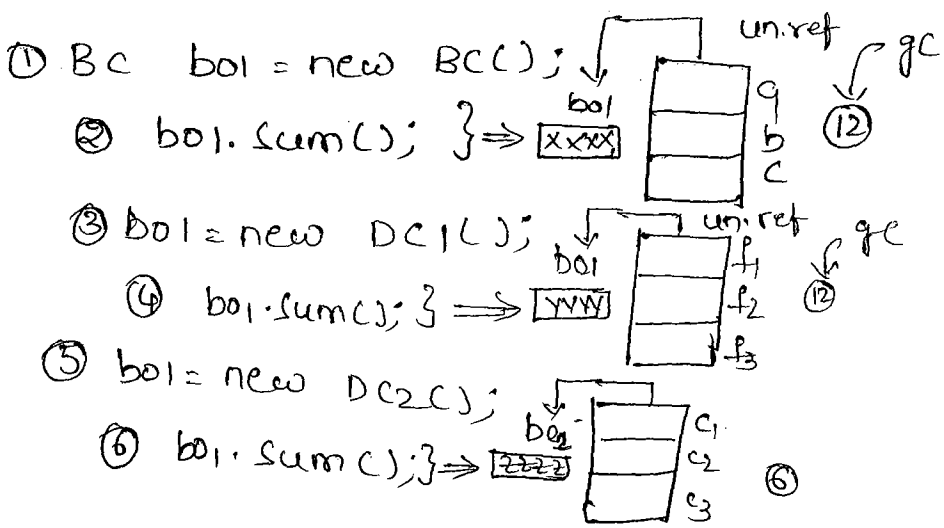
→ Here parent is a super class & child is the sub class

→ Here p is an object of type Parent & an object p contains refference of Child.

→ Type of object makes us to understand JVM will looks for existence of method

→ Type of object refference makes us to understand JVM will execute the method.

EX: Let us consider the following set of executable stmts (EL) which are used for executing the program represented in fig(1) by using dynamic binding principle.



In the above stmts

→ In line numbers (2,3,5) the object bo1 contains a refference ... of polymorphic object

→ The method sum is actually available in one form & it is implimented in multiple forms & hence sum () is called polymorphic method.

→ In line numbers ②, ④, ⑥ the stmt bol. sum() is actually one stmt but it gives int sum, float sum & char sum & hence the stmt is called polymorphic stmt.

In real world, we can develop any appll by using 3 cases.

Case:-1 Develop the Business logic in the form of independent classes & we develop the execution logic by direct object creation (plain old Execution logic)

case:-2 We develop the BL in the form of dependent classes & we develop by following polymorphism along with method over riding & we develop the EL by using direct object creation

The above 2 cases gives the following limitations

(a) Appll takes more memory space

(b) Appll takes more execution time.

(c) Appll give performance is degraded.

Hence case 1 & case 2 are not recomended.

case3:- We develop the BL in the form of dependent classes by following polymorphism along with method over riding & we develop the EL by using dynamic binding principle (Indirect object creation)

cases will give the following advantages (160)

(a) Appl takes less memory space

(b) Appl Execution time is less

(c) Appl performance is enhance

Hence, industry recomends to follow cases.

Whatever the ~~appls~~ OOPL's appls follows polymorphism & dynamic binding principles those appl will get the benifites of polymorphism & dynamic binding.

24/7

Q:- Define static Polymorphism & dynamic polymorphism?

Ans:- In static polymorphism the over loaded methods binds with an object at compile time.

The limitation static Polymorphism is that poor utilization of the resources (main memory)

In dynamic polymorphism the over loaded methods binds with an object at run time.

The advantage of dynamic polymorphism is that effective utilization of the resources.

Java programming follows only dynamic polymorphism but not static polymorphism due to its limitation.

Eventhough, C++ programming follows both static & dynamic polymorphisms, the real time C++ programmers will follow only dynamic polymorphism.

Sharp NOTE 1:- When we impliment polymorphism with method over loading concept then JVM will always consider signature of the over loaded methods.

Sharp NOTE 2:- When we impliment polymorphism concept with method over riding then JVM will always consider refference of derived classes (DC) present in the object of base class (BC).

Abstract classes in Java

WKT each & every Java program starts with a concept of class. Without classes concept we can't develop any purposeful appll.

In Java programming we've 2 types of classes. They are
1. concrete classes
2. Abstract classes

Concrete class:-

→ A concrete class contains fully defined methods

→ Defined methods of a class are also known as concrete/ implimented methods.

→ once the class is concrete we can instantiate (creation) its object directly

Ex: class Test

```
{
    void show()
    {
      sop("show");
    }
    void disp()
    {
      sop("disp");
    }
}
```

Here Test is one of the concrete class & whose object can be created directly.

```
Test t1=new Test();
    t1.show();
    t1.disp();
```

In real world appls concrete classes are always recomended to deal with specific requirements which are suitable for individual programmers. It is not recomended to use concrete classes to deal with common requirements which are suitable for all the programmers.

If we choose concrete classes to deal with common requirements the we get the following limitations.

1. Appl takes more memory space
2. Appl takes more execution time
3. [illegible] is degraded

To overcome these limitations to fullfil the common requirements of all the programmers. Industry is recomended to use the concept of abstract classes.

* We use always abstract classes to deal with common requirements. programatically, abstract classes contains common features which are suitable for all the programmers.

25/7 Def^n of abstract class :-

An abstract class is a collection of defined method and un defined methods.

undefined methods of a class are also known as abstract methods (or) un implimented methods

Def^n of abstract method :-

An abstract method contains only method heading/prototype/declaration but not containing method body/deffiniation.

To make un defined method as abstract method, the declaration of the un defined method must be preceeded by a keyword abstract otherwise we get compile time error.

SYNTAX for abstract method

```
abstract Ret-type methodName( list of formal parameter types if any );
```

Eg:- abstract void op(int,int);
abstract void sum( );
abstract void draw( );

→ IKIT every method of Java belongs to a class including abstract methods. If any class of Java contains any abstract methods then it is called abstract class. & whose defn must be made as abstract by using abstract keyword otherwise we get compile time error.

SYNTAX for abstract class

```
abstract class <clsName>
  {
    ------
    ------
  abstract Ret-type methodName( list of formal Parms types if only );
    ------
    ------
  }
```

Eg:- abstract class operations

```
  {
    ------
    ------
    abstract void op(int, int);
    abstract void sum( );
    abstract void draw( );
    ------
    ------
  }
```

→ Here, operations is one of the abstract class & whose object can't be instantiated directly becoz there is no use of calling abstract methods. But an object of operations class can be created indirectly w.r.t implimentation class. (This class will provide def^n for abstract methods of abstract class).

Eg:-

operations O_1 = new operations ( ); // Invalid

Hence, in general an object of abstract class can't be created directly but it can be created indirectly w.r.t implimentation/derived class.

* Write a Java program which will compute two integers & floats etc

```
// Abdemo1.Java
import Java.util.scanner;
abstract class operations
 {
  Scanner s=new scanner(system.in);
  abstract void sum();
 } // Operations
 class Isum extends operations
  {
   int a,b,c;
   void sum()
```

```
{
  sop("Enter First int val");
  a=Integer.parseInt(s.nextLine());
  sop("Enter second int val");
  b=Integer.parseInt(s.nextLine());
  c=a+b;
  sop("int sum="+c);
}
} // Isum.... Impl" class
class Fsum extends operations
{
  float a,b,c;
  void sum() // overriding abstract method
  {
    sop("Enter First float val");
    a=Float.parseFloat(s.nextLine());
    sop("Enter second float val");
    b=Float.parseFloat(s.nextLine());
    c=a+b;
    sop("float sum ="+c);
  }
} //Fsum-impl" class
class Abdemo1
{
  public static void main(String [] args)
```

```
{
 // Operations O1 = new operations();' invalid, operation is
   abstract
  sop("w.r.t. operations --- Isum --- Dynamic Binding");
     operations O1 = new Isum();
          O1.sum();
  sop("w.r.t operations --- Fsum --- Dynamic binding");
          O1 = new Fsum();
          O1.sum();
 }
}
```

* Write a Java program which will calculate area of different shapes. like circle, rect & square.

```
// Abdemo2.Java
import java.util.Scanner;
abstract class shape
 {
  Scanner s = new Scanner(System.in)
  abstract void area();
 } // shape
 class circle extends shape
  {
   float pi, r;
    int a;
   void area()
   {
    sop("enter the value of radius");
```

```
r = Float.parseFloat(S.nextLine());
a = 2πr2;
sop("area of circle = "+a);
}
} // circle
class rect extends shape
{
{ int a, b, l; void area()
sop(" Enter first int val ");
l = Integer.parseInt(S.nextline());
sop(" Enter second int val");
b = Integer.parseInt(S.nextline());
c = l x b;
sop("area of rect = "+c);
}
} // rect
class square extends shape
{
int a, d;
void area()
{
sop(" Enter the value ");
a = Integer.parseInt(S.nextline());
d = a2;
sop("area of square = "+d);
}
```

```
class Abdemo2
{
  Public static void main(String [] args)
  {
    sop("w.r.t shape ------ circle ---Dynamic binding");
        shape s1 = new circle();
            s1.area();
    sop("w.r.t shape ---- rect ---- Dynamic binding");
            s1=new rect();
            s1.area();
    sop("w.r.t shape --- square --- Dynamic binding");
            s1=new square();
            s1.area();
  }
}
```

* write a Java program which will calculate swapping of 2 integers, 2 floats, 2 chars etc.

* write a Java program which will calculate perimeter of different shapes like circle, rect, square etc

Points to be remembered :- (156)

→ All the abstract classes contains common reusable features & they always deals with common requirements.

→ All the abstract classes must parcipates in inheritance. becoz they contains common features

→ Abstract class defns should not be final becoz they always parcipates in inheritance.

→ An object of abstract class can't be created directly but it can be created indirectly w.r.t. implimentation class.

programmatically, an object of abstract class = an object of its implementation class

→ Every abstract class of Java must contain implementation of a derived class. So, that whose object can be created indirectly.

→ Abstract method should not be private & final.

→ Abstract classes contains only abstract instance methods but not abstract static methods becoz Instance methods can can do repeated operation. whereas static methods can do one time operation.

→ Abstract classes follows polymorphism principles for BL development & also follows dynamic binding for execution logic.

→ The advantages of abstract class are

a, Appl takes less memory space

→ Abstract methods of abstract class takes less method overhead cost whereas defined methods of concrete class takes more method overhead cost.

→ An object of abstract class can be declared but it can't be refferenced directly & we can refference it indirectly w.r.t implementation class.

EX:- operations O1; // valid

O1 = new operations(); // Invalid

O1 = new ISum(); // valid

→ An abstract class of Java can contain main method.

27/7 Abstract Base class & Abstract derived class:-

Def" of ABC :- An ABC always contains physical representation of abstract method which will always participates in inheritance process.

Def" of ADC :- An ADC contains logical representation of abstract methods which are inherited from ABC.

```
abstract C1
  abstract void f1();
  abstract void f2();
        |
        | extends
        ↓
abstract C2
  void f1()
  {
    sop("f1-over-C2");
  }
  abstract void f2();
```

→ ABC

→ physical representation of abstract method

← ADC

→ logical representation of abstract method

→ W.r.t ABC & ADC we can't instantiate their objects directly but we can creat their objects indirectly w.r.t concrete implimentation class.

→ To make the derived class as abstract, it has to satisfy the following properties.

a) If the DC is inherting multiple abstract methods from abstract base class & if the derived class is not defining atleast one abstract method. Then, that DC defn must be made as abstract otherwise we get compile time error.

b) If the DC contains completely defined methods & if the DC programmer add physical representation of abstract method. then the DC defn must be made as abstract otherwise we get compile time error.

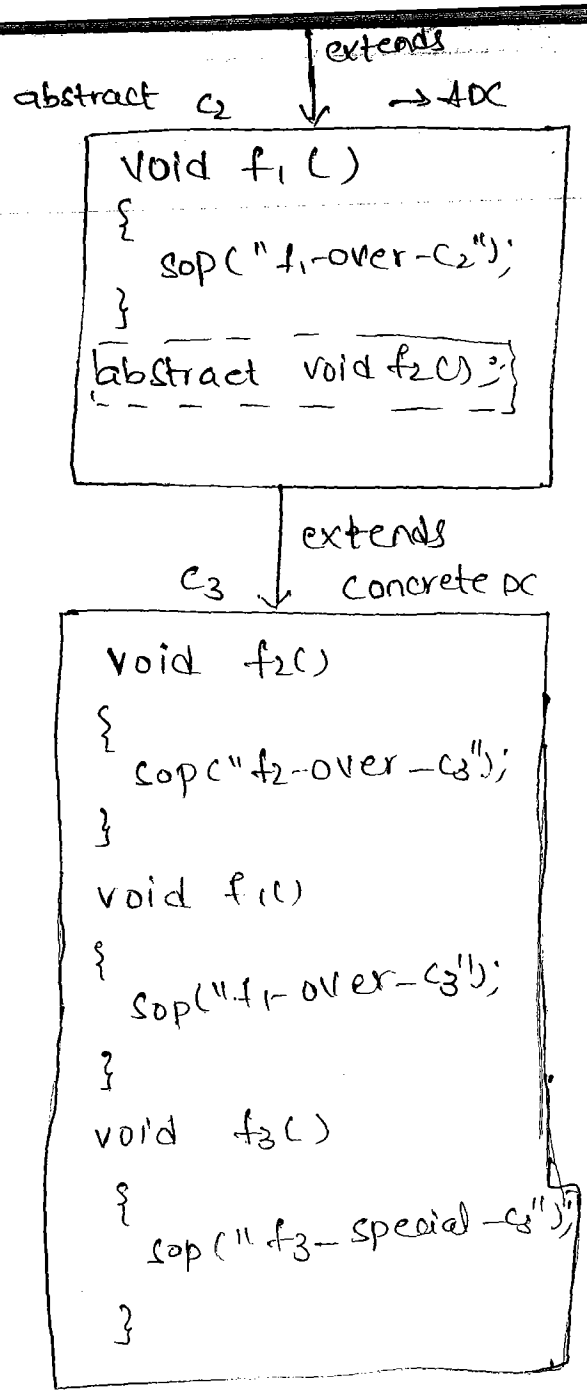
ABC's & ADC's are always reusable until concrete class implementation formed.

* write a Java Program which illustrate the concept of ABC & ADC.

abstract C, → ABC

```
abstract void f1();
abstract void f2();
```

↓ extends



*

```
//AbDemo4.Java
abstract class C1
{
    abstract void f1();
    abstract void f2();
} //C1--- ABC
```

```
abstract c2 extends c1
{
   void f1()
   {
      sop ("f1() --- overridden --- c2");
   }
} // c2 ---- ADC
class c3 extends c2
{
   void f2()
   {
      sop("f2() --- overridden ---- c3");
   }
   void f1()
   {
      sop("f1() ---- overridden ---- c3");
   }
   void f3()
   {
      sop("f3() ---- special --- c3");
   }
} // c3 --- CDC
class Abdemo4
{
   public static void main(string [] args)
```

```
{
  Sop("w.r.t C3 --- direct object creation");
    C3 o3 = new C3();
        o3.f1();
        o3.f2();
        o3.f3();
  Sop("w.r.t C2 --- Indirect object creation - Dynamic binding");
    // C2 o2 = new C2(); Invalid, becoz C2 is abstract
    C2 o2 = new C3();
        o2.f1();
        o2.f2();
    // o2.f3(); Invalid, becoz f3() does not exist in C2
  Sop("w.r.t. C1 --- Indirect object creation - Dynamic binding");
  // C1 o1 = new C1(); Invalid, becoz C1 is abstract
  // C1 o1 = new C2(); Invalid becoz C2 is abstract
    C1 o1 = new C3();
        o1.f1();
        o1.f2();
    // o1.f3(); Invalid, becoz f3() does not exist in C1
  }
}
```

Consequences in abstract classes:-

consequence:)

If any concrete class of Java contains purely null body methods then there is no use of calling null body methods w.r.t the corresponding concrete class object becoz NBM will not give any result. Hence, such corresponding

concrete class defns must be made as abstract & they are called abstract concrete class.

Def of Null Body Method (NBM):-

A NBM is one of the defined method & it never contains block of stmts.

SYNTAX:-

```
Ret.type MethodName (list of formal parameters if any)
{
    // Null Body
}
```

NBM's always exist in abstract concrete class & they must be overridden in implementation class of abstract concrete class.

* write a Java program which illustrate NBM's implementation

```
// Abdemo5.Java
abstract class parent
{
  void goe()// NBM
  {
  }
} // Parent --- ACC
class child extends parent
{
  void go()
  {
    sop (" going to college");
    sop(" reading Java notes");
  }
```

... class of Parent

```
class Abdemo5
{
  Public static void main(*String [] args)
  {
     sop("w.r.t. parent--- dynamic binding");
      Parent p = new Child();
        p.goc();
  }
}
```

28/4

Consequence:2

Industry is always recomended to override abstract methods of abstract classes but not defined methods of concrete class becoz method over head cost of overriding abstract methods is less than method overhead cost of overriding defined methods of concrete class.

Method overhead cost for overriding Defined method } = cost required for eliminating Existing body } + cost required for constructing new body

Method overhead cost for overriding abstract method } = cost required for constructing new body only

More method overhead cost reflects on more execution time & less method overhead cost reflects on less execution time

Q:- Can we make a concrete class as abstract? give reason?

Ans:- yes,

Reason :- See consequence ① [See previous page]

~~Interfaces~~

Limitations of Abstract classes :-

1. Abstract classes never participates in multiple inheri-tance
2. Abstract classes provides only common reusable features & unable to provide universal common reusable features.
3. WKT AC is a collection of defined methods & abstract methods. If we override defined methods of AC takes more execution time & overriding of abstract methods of interface takes less execution time. The overall execution time of abstract classes related appls is partially less & unable to provide completely less execution time.

Hence, to overcome the limitations of AC's in dealing with common requirements we use an another concept called interfaces.

Interfaces :-

Interfaces

The main intension of using interfaces concept is ~~that~~ to overcome the limitations of abstract classes.

→ Interfaces concept is used to develop universal user defined data type.

→ To develop universal user defined data type with interface concept we use a keyword called Interface

→ Each & every interface name is programatically treated as universal user defined data type.

Def^n of Interface:-

Def-1 An Interface is a collection of public static final xxx data members & public abstract methods

(or)

Def-2 An interface is a collection of universal common reusable data members & universal common reusable methods.

SYNTAX for defining an Interface :-

```
interface <intfName>
{
    variable declaration cum Initialization
    methods declaration
}
```

Explanation:-

→ Interface is a keyword used for developing universally user defined data type.

→ <IntfName> represents a java valid variable name treated

In general interfaces provides rules/specifications to the implem-entation classes.

→ Programatically, an object of interface can't be created directly but it can be created indirectly w.r.t implementation class

→ Variable declaration represents data members of the interface & they meant for data universally reusuable. So, that they must be initialized otherwise we get compile time error. Since, interface datamembers are universable, whose values can't be changed (final), whose memory space can be created only once (static) & they can be accessed everywhere (public). Hence, by default all the data members of interface belongs to public static final xxx data members.

Ex:- $\underbrace{\text{public static final}}_{\text{Appended by Java compilation Environment}}$ $\underbrace{\text{float PI = 3.1417f;}}_{\text{written by Java Programmer}}$

→ Methods declaration represents set of undefined methods for performing universal operations. To make these undefined methods as abstract explicitly we need not to write abstract (by default abstract) & to make them universally accessable we need not to write public hence by default every method of interface belongs to public abstract

Ex:- $\underbrace{\text{public abstract}}_{\text{Appended by Java compilation}}$ $\underbrace{\text{void draw();}}_{\text{written by Java programmer}}$

29/7

Eg: Define an interface sathya by taking suitable DM's & Methods.

```
//Sathya.java
interface Sathya
{
  int a=10;
  int b=20;
  void f1();
  void f2();
}
```

→ compile the above program

javac Sathya.java

Ensure that sathya.class file will be generated. In general for each & every interface def^n also a .class file will be generated.

→ view the profile of sathya interface

```
javap sathya
interface sathya {
  public static final int a;
  public static final int b;
  public abstract void f1();
  public abstract void f2();
}
```

In general all the features of interface are by default belongs to public & hence features of interfaces are universally accessable

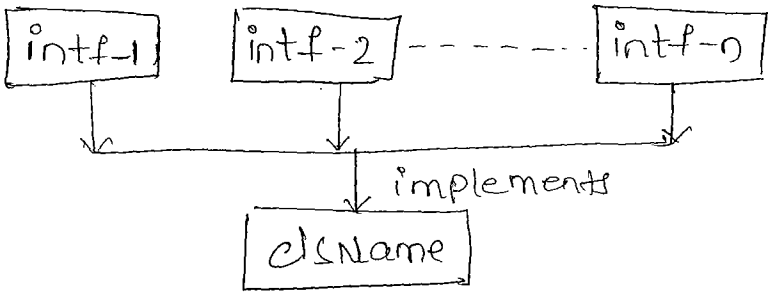
Inheriting the features of interfaces

In Java programming we've 3 approaches to inherite the features of interfaces. They are

Approach :- I

This approach makes us to understand how to inherite the features from base interfaces into derived class

Diagram :-



SYNTAX :-

```
[abstract]class <clsNames> impliments <intf-1>,<intf-2>---<intf-n>
    {
        Variable Declaration
        method Def/Declaration
    }
```

Explanation :-

In the above syntax

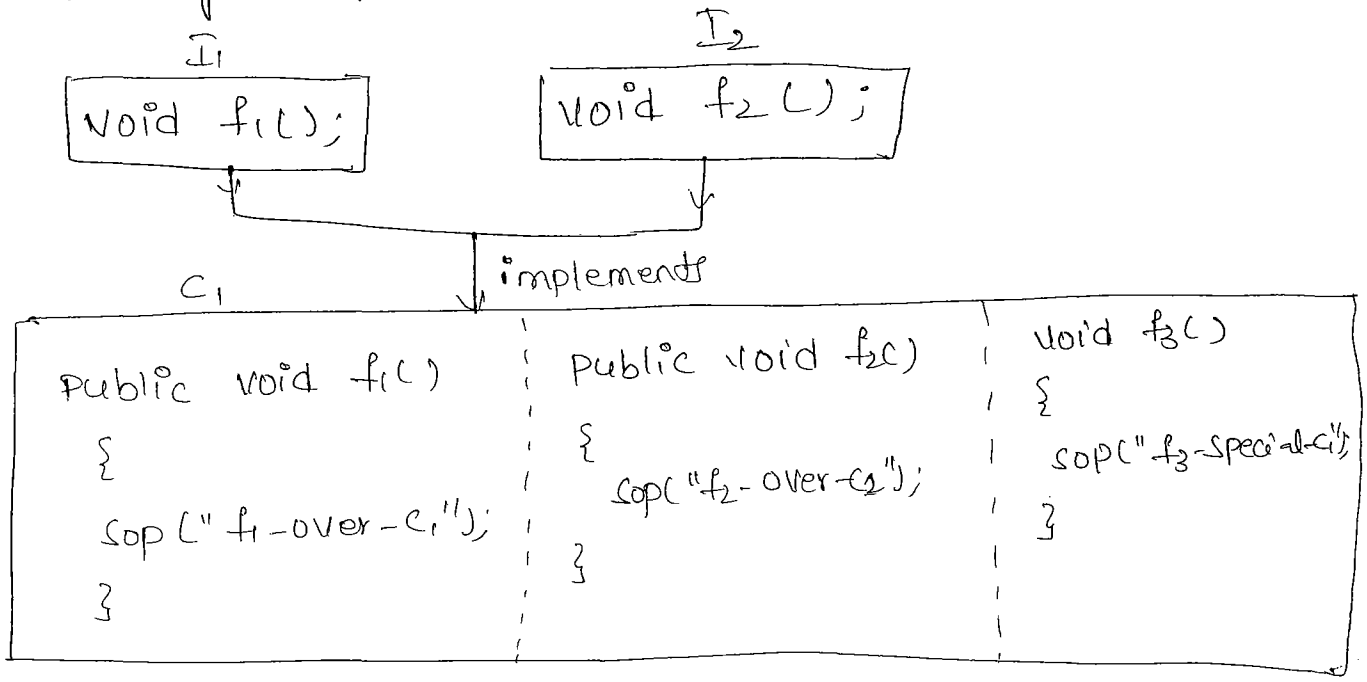
① < clsNames represents name of the DC.

② <intf-1><intf-2>--<intf-n> represents name of the BI'S.

③ impliments is a Keyword which is used to inherite the features of ~~BCS~~ Base interface(s) into DC + it provides

In Java programming one DC can extends only one BC whereas one DC can impliments either one or more than one interface becoz Java programming supports multiple inheritance through the concept of interfaces but not through classes concept.

* write a Java program which will impliment the following diagram.




```
// IntfProg1.Java
interface I1
{
  sopln(" ~~public~~ void f1(); // public abstract void f1()

} // I1
interface I2
{
  void f2(); // public abstract void f2()
} // I2
class C1 implements I1, I2
 {
  public void f1()
```

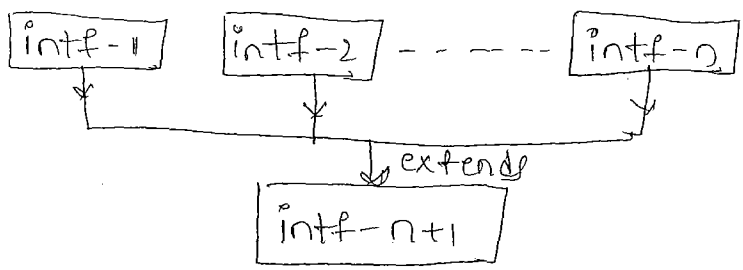
```
{
  sop("f1() ---- overridden --- C1");
}
public void f2()
{
  sop("f2() --- overridden --- C1");
}
void f3()
{
  sop("f3() ---- special ---C1");
}
} // class
class Intfprog1.
{
  public static void main(String [] args)
  {
sop("w.r.t C1 -- Direct Object creation");
    C1 o1 = new C1();
      o1.f1();
      o1.f2();
      o1.f3();
  sop("w.r.t I1 -- In-direct Object creation -- Dynamic binding");
    // I1 io1 = new I1(); Invalid, bcoz I1 is abstract
    I1 io1 = new C1();
      io1.f1();
    /* io1.f2(); invalid, bcoz f2() & f3() does not exist
       io1.f3(); */                in I1
```

```
sop("w.r.t I2 -- In-direct object creation -- Dynamic binding");
    // I2 io2 = new I2();  invalid, becoz I2 is abstract
    I2 io2 = new C1();
        io2.f2();
    /* io2.f1();  invalid, becoz f1() & f3() does not
       io2.f3(); */   exist in I2
  }
}
```

Approach :- 2 [interface inheritance]

This approach makes us to understand how to inheri-te the features from base interface(s) into derived interface

Diagram :-



Syntax :-

```
interface <intf-n+1> extends <intf-1>,<intf-2> ....<intf-n>
  {
    variable declaration cum initialization
    method declaration
  }
```

Explanation :-

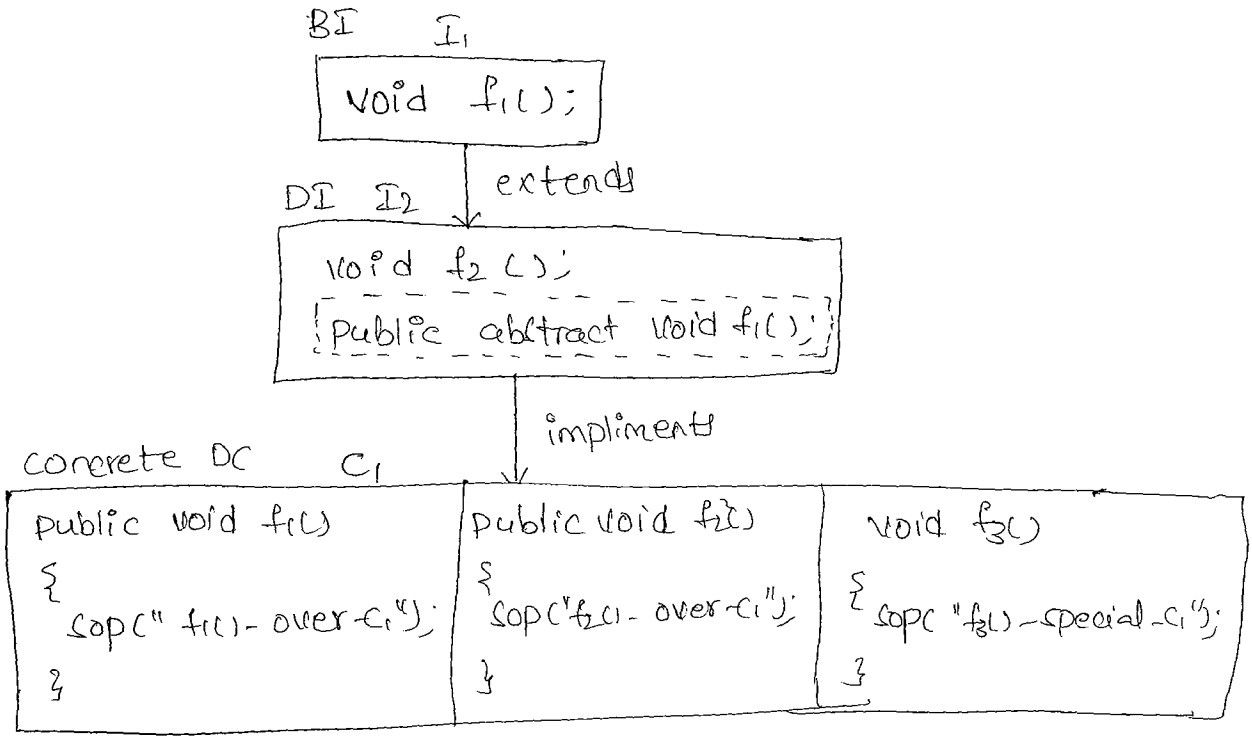
In the above syntax

1. <intf-n+1> represents name of the derived interface
2. <intf-1>, <intf-2> ... <intf-n> represents name of the base interfaces

3. Extends is a keyword which is used to inherite the features of base interfaces into derived interface & it provides richest functionality to derived interface

* In Java programming one derived class can extends only one base class whereas one derived interface can extends either one or more than one interface.

* write a Java program which will implement the following diagram




```
// Intfprog2.java
interface I1
{
    void f1();
} // I1
interface I2 extends I1
{
    void f2();
    public abstract void f1();
```

```
class c1 implimentes I2
{
    Public void f1()
    {
      sop("f1() --- overridden --- c1");
    }
    Public void f2()
    {
      sop("f2() --- overridden ---- c1");
    }
    void f3()
    {
    sop("f3() --- special ---- c1");
    }
} // class
class intfprog2
{
  public static void main(String [] args)
  {
    sop("w.r.t c1 Direct object creation");
        c1 o1 = new c1();
          o1.f1();
          o1.f2();
          o1.f3();
    sop("w.r.t I1 In-direct object creation - Dynamic binding");
        // I1 io1 = new I1(); invalid, beez I1 is abstract
        I1 io1 = new c1();
          io1.f1();
```

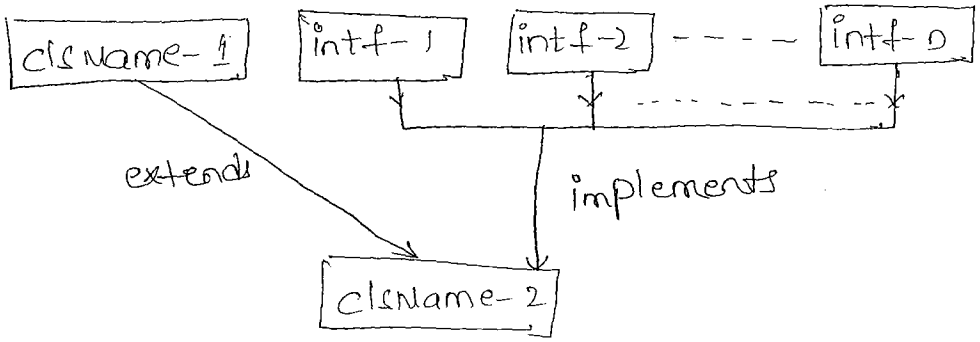
```
/* io1.f2();  invalid, becoz f2() & f3() does not exist in I1
   io2.f3(); */
sop("w.r.t I2 in-direct object creation - Dynamic binding");
// I2 io2 = new I2();  invalid, becoz I2 is abstract
   I2 io2 = new C1();
      io2.f2();
   /* io2.f3();  invalid, becoz f1() & f3() does not exist in I2
      io2.f1(); */
}
}
```

30/7

Approach:-3 [Parallel Inheritance]

In this approach we understand how to inherite the features of base class & base interfaces into derived class. [known as parallel inheritance]

Diagram :-



Syntax :-

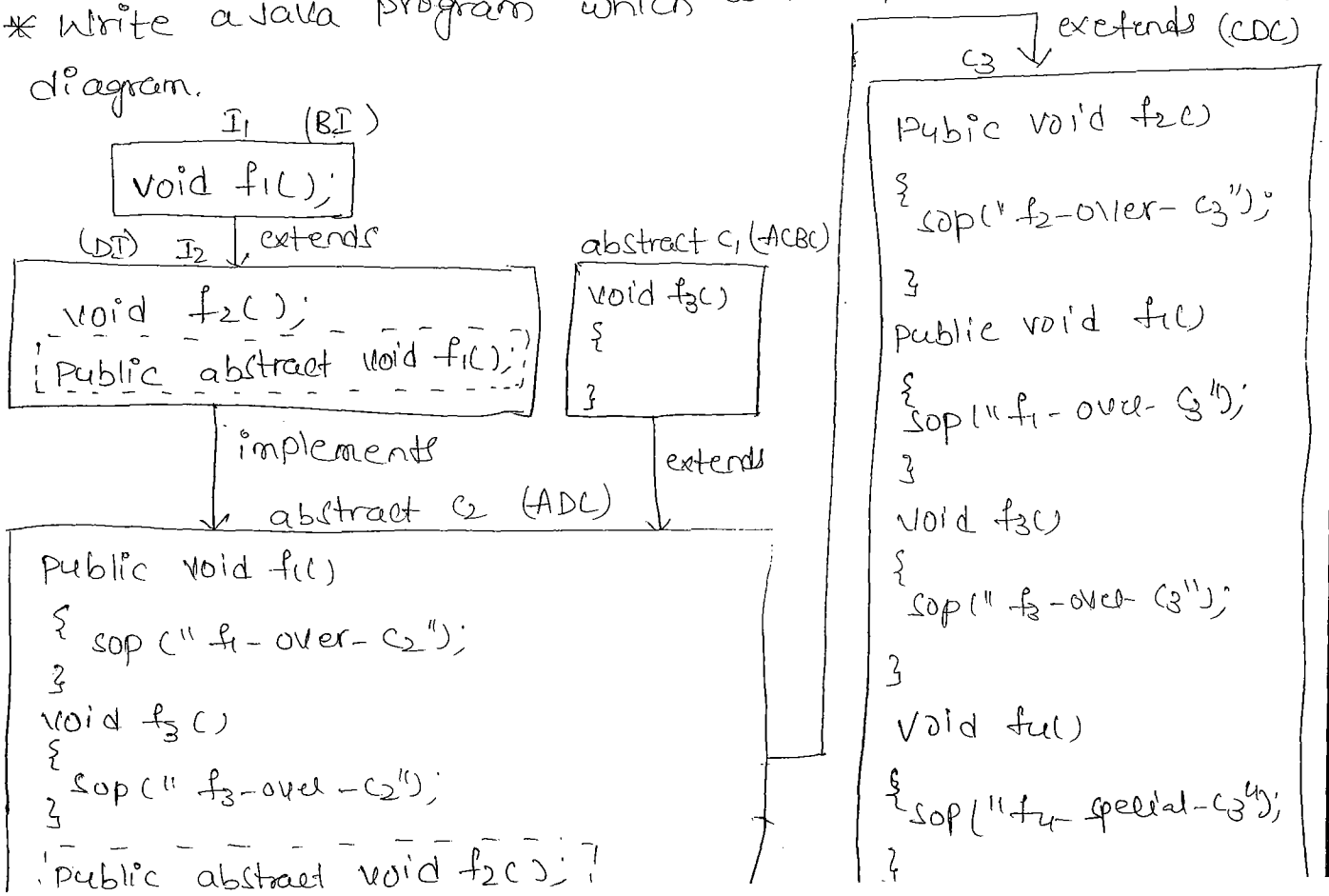
```
[abstract] class <clsName-2> extends clsName-1
                    implements <intf-1><intf-2> ... <intf-n>
{
    variable declaration;
    method defn/declaration;
```

Explanation :-

In the above syntax

1. <clsName-1> & <clsName-2> represents name of the base & derived classes
2. <intf-1> ....<intf-n> represents name of the base interfaces
3. Here extends & impliments are the 2 keywords used for inheriting the features of base class & base interfaces parallelly into derived class. + they provides richest functionality to the DC.
4. When we use both extends & implements keywords in a simple java program then it is mandatory to the java programmer to use extends keyword word first & later we use impliments keyword. otherwis we get compile time error.

* Write a java program which will implement the following diagram.



```
// intfprog3.java
interface I1
 {
  void f1();  // public abstract void f1()
 } // I1 - Base interface
interface I2 extends I1
{
  void f2();  // public abstract void f2()
} // I2 - Derived interface
abstract class C1
{
 void f3() // Null body method
  {
   }
} // C1 - Abstract concrete base class
abstract class C2 extends C1 implements I2
{
  public void f1()
  {
    sop("f1() -- overridden -- C2");
  }
 void f3()
 {
   sop("f3() --- overridden --- C2");
 }
} // C2 --- abstract derived class
```

```
class C3 extends C2
{
  public void f2()
  {
    sop(" f2() --- overridden -- C3");
  }
  public void f1()
  {
    sop(" f1() --- overridden --- C3");
  }
  void f3()
  {
    sop(" f3() --- overridden --- C3");
  }
  void f4()
  {
    sop("f4() --- special -- C3");
  }
} // C3 -- concrete derived class

class intfprog3
{
  public static void main(String [] args)
  {
    sop(" w.r.t C3 - Direct object creation");
        C3 o3 = new C3();
            o3.f1();
            o3.f2();
            o3.f3();
```

```
sop("w.r.t C2 - indirect object creation_dynamic binding");
    // C2 O2 = new C2(); invalid, becoz C2 is abstract
    C2 O2 = new C3();
        O2.f1();
        O2.f2();
        O2.f3();
    // O2.f4(); invalid, becoz f4 does not exist in C2
sop("w.r.t C1 - indirect object creation - dynamic binding");
    //C1 O1 = new C1(); invalid, becoz C1 is abstract
    // C1 O1 = new C2(); invalid, becoz C2 is abstract
    C1 O1 = new C3();
        O1.f3();
    /* O1.f1(); invalid, becoz f1(), f2() & f4()
       O1.f2();   does not exist in C1
       O1.f4(); */
sop("w.r.t I2 - indirect object creation - dynamic binding");
    // I2 io2 = new I2(); invalid, becoz I2 is abstract
    // I2 io2 = new C2(); invalid, becoz C2 is also abstract
    // I2 io2 = new C1(); invalid, becoz C1 is also abstract
    I2 io2 = new C3();
        io2.f1();
        io2.f2();
    /* io2.f3(); invalid, becoz f3() & f4() does not
       io2.f4(); */   exist in I2
Sop("w.r.t I1 - indirect object creation - dynamic binding");
    // I1 io1 = new I1(); invalid, becoz I1 is abstract
```

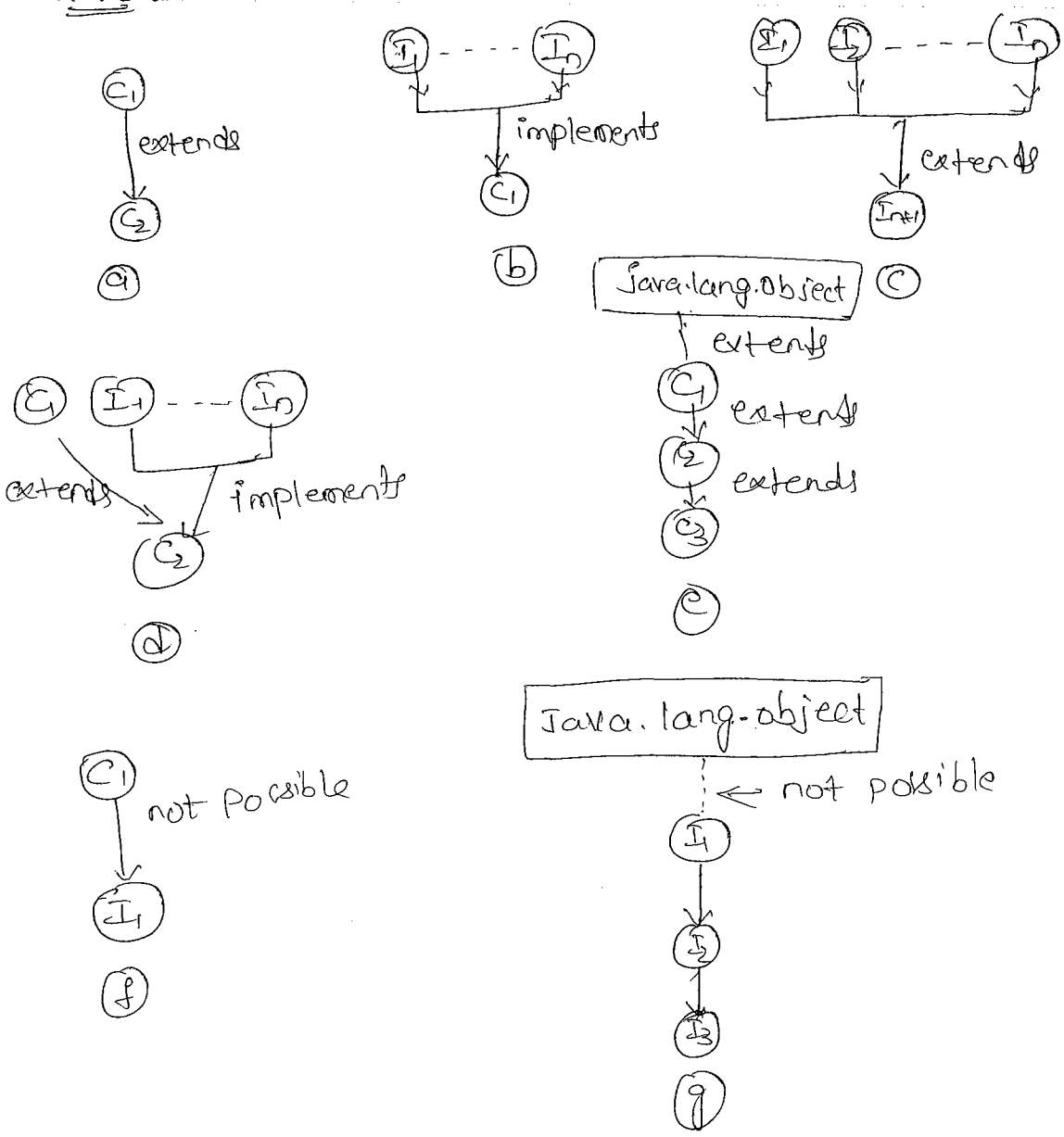
```
// I1 io1 = new I2();  invalid, becoz I2 is also abstract
// I1 io1 = new C2();  invalid, becoz C2 is also abstract
   I1 io1 = new C3();
        io1.f1();
/* io1.f2();  invalid, becoz f2(), f3() & f4() does
   io1.f3();            not exist in I1
   io1.f4(); */
    }
}
```

Points to be remembered in interfaces:-

1. Interfaces always contains universal common reusable features.
2. All the interfaces in Java must Participates in inheritance.
3. Interface defns should not be final.
4. Interfaces defns should not contain defined main methods.
5. While we are defining interface methods in the context of Derived class, interface methods must be preceeded by public otherwise we get compile time error.
6. Interfaces will not contain implicit pre defined super class whereas class still contain implicit pre defined super class
7. Every interface must contain an implementation class to Create its object indirectly.
8. Interfaces of Java never contains constructors, static blocks & instance blocks becoz data members of interface already initialized.
9. An object of an interface can be declared but it can't be refferenced directly.
10. Interfaces contains only static data members but not containing instance data members.

11. The modifier of the interface (96) is by default abstract

31/7 NOTE :-



class C1 implements I1

{

}

(h)

The super class of implementation of I1, class C1 is java.lang.Object

Inner/nested Interfaces:-

WKT normal interfaces of Java contains collection of abstract methods. If any class implements the normal interface which contains collection of abstract methods then the implementation class needs to define interested methods along with uninterested methods which is time consuming process. If we are not defining uninterested methods the current implementation class will become abstract & we can't call interested method. which is not a recomended procedure.

TO overcome these programatic difficulties of normal interfaces, we've ano another called nested/inner interfaces

Def'n of inner/nested interface:-

If we define an interface "y" inside of an another interface "x" then the interface def'n "x" is called outer interface & def'n of interface "y" is called inner/nested interface.

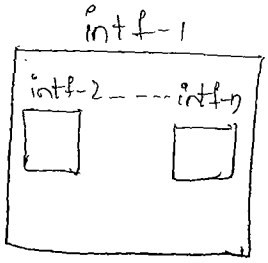
Def of containership:-

The process of defining one interface def'n inside of another interface def'n is called "containership".

syntax:-1

```
interface <intf-1>
{
  -----
  -----
  interface <intf-2>
  {
    -----
  }
}
```

```
interface <intf-n>
{
  -----
  -----
}
}
```



To access inner interfaces in syntax-1 we write <intf-1>.<intf-2> and <intf-1>.<intf-n>.

NOTE:-

When we compile the above inner interfaces program we get the .class file as follows.

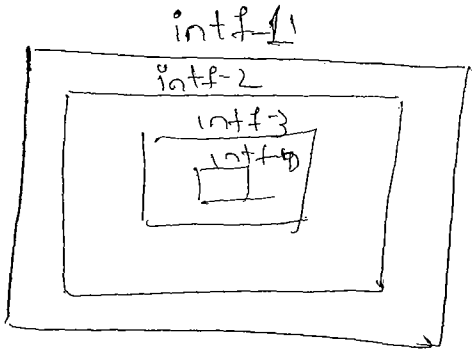
<intf-1>.class

<intf-1>$<intf-2>.class

<intf-1>$<intf-n>.class

Syntax-2:-

```
interface <intf-1>
{
    ----
    ----
    interface <intf-2>
    {
        ----
        ----
        interface <intf-3>
        {
            ----
            ----
            interface <intf-n>
            {
                ----
                ----
            }
        }
    }
}
```



To access inner most interface we write <intf-1><intf-2><intf-3>----<intf-n>

Advantages of inner interface :-

when we develop any java app with the concept of inner interfaces we get the following advantages

1. They allowses to define interested abstract methods by leaving un interested abstract methods

2. Inner interfaces related appls gives less appl develop-ment time & whose overall performance is enhance.

* write a java program which illustrate the concept of inner interfaces.

```
// InnerIntfDemo.Java
interface SathyaTech
  {
    void PayFees();
    interface Java
     {
       void learnJava();
     } // Java--- inner intf
    interface DotNet
     {
       void learnDotNet();
     } // DotNet-inner intf
  } // SathyaTech _ outer intf
```

```
class JavaStudent implements SathyaTech, SathyaTech.Java
{
  Public void learnJava()
  {
    sop("learning Java");
  }
  Public void PayFees()
  {
    sop("paid fees");
  }
} // JavaStudent -- Impl class

class InnerIntfDemo
{
  public static void main(String []args)
  {
    sop("w.r.t SathyaTech, SathyaTech.Java -Indirect
        object creation- dynamic binding");
      SathyaTech st= new JavaStudent();
          st.PayFees();
      SathyaTech.Java sj=new JavaStudent();
          sj.learnJava();
    Sop("w.r.t JavaStudent--- Direct object creation");
      JavaStudent js=new JavaStudent();
          js.learnJava();
          js.PayFees();
  }
```

* Develop a java appl for performing arthemetic operations on integers & floats with inner interfaces concept.

```
//InnerIntfDemo.java
interface AOP
{
  interface Sum
  {
    void int();
    void float();
  }
  interface Sub
  {
    void int();
    void float();
  }
  interface Mul
  {
    void int();
    void float();
  }
  interface Div
  {
    void int();
    void float();
  }
} //AOP
```

```
class C1 implements AOp.Sum
{
    public void int()
    {
      sop("operation on integers");
    }
    public void float()
    {
      sop("operation on floats");
    }
}
Class InnerIntfDemo
{
  public static void main(String [] args)
  {
    sop("w.r.t C1 - Direct object creation");
        C1 S1 = new C1();
        S1.int();
        S1.float();
    Sop("w.r.t AOp - indirect object creation - dynamic binding");
        AOp S2 = new C1();
            S2.float();
        AOp.Sum S3 = new C1();
            S3.int();
  }
}
```

Functional Interface :- (201)

It is one of new concept/feature available in JDK 1.8 version onwards.

→ Def:- An interface is said to be a functional interface if & only if it contains only one abstract method.

→ With an ordinary programming of Java every interface method must be defined in the implementation class.

→ In Java programming one can also implement/define abstract methods of interfaces without implementation classes. i.e, by using the concept of anonymous inner classes.

→ Most of our real time appls the abstract methods of functional interfaces overridden in anonymous inner classes rather than explicit implementation classes.

* Write a Java program which illustrate the concept of anonymous inner class for implementing functional interface methods.

1/8/15

```
// Satkya.java
interface India
{
  void makeIndia();
} // India - functional Interface
class Satya
{
  public static void main (String [] args)
  {
    India io = new India()
    {
      public void makeIndia()
```

```
    {
      sop("makeIndia() --- overridden in AINC");
    }
  };
  io.makeIndia();
  India io2=new India()
  {
    Public Void makeIndia()
    {
      public Void makeIndia()
      {
        sop ("makeIndia() --overridden in another AINC");
      }
    };
    io2.makeIndia();
  }
}
```

When we compile the above program internally Java environment created 2 anonymous inner classes namely Satya$1 & Satya$2 Here 1 & 2 are called anonymous inner classes namely which are implementing an interface India & providing different implementations for makeIndia() method.

* Write a Java program for performing swaping of integers & String data by using functional interfaces with anonymous inner classes by accepting the values dynamically.

```
//SathyaTech1.java
import java.util.Scanner
interface Swap
{
```

```
void swapValues();
} // swap - functional interface
class Sathyatech
{
  public static void main( String [] args)
  {
    new swap()
    {
      public void swapValues()
      {
        Scanner s = new Scanner (System.in)
        Sop(" Enter first int Val");
        int a = Integer.parseInt (s.nextLine());
        Sop(" Enter second int Val");
        int b = Integer. parseInt (s.next Line());
        Sop("Original value of a="+a);
        Sop ("original value of b="+b);
        int t=a;
          a=b;
          b=t;
        Sop("Interchanged value of a="+a);
        Sop( "Interchanged Value of b="+b);
      }
    }.swapValues(); // calling the method with nameless object
    new swap()
    {
      public void swapValues()
      {
        Scanner s=new Scanner (System.in);
        Sop("Enter first string Val");
        String a = s.nextLine();
```

```
Sop("Enter second String val");
String b = S.nextLine();
Sop("original value of a="+a);
Sop("original value of b="+b);
String t=a;
        a=b;
        b=t;
Sop("Interchanged value of a="+a);
Sop("Interchanged value of b="+b);
}
}.swapvalues();
}
}
```

Q:- Can we implement the abstract methods of interface without implementation class?

Ans:- yes, wrt anonymous inner class.

Marker/Tagged interface

MI doesn't contain any abstract methods explicitly (Implicitly there exists some abstract methods which are JVM dependable overridden methods) & it provides run time behaviour to the implementation class of marker interface

Ex:- java.lang.Cloneable
java.io.Serializable
java.rmi.Remote etc

: write a Java program which will display fonts of the System.

```
// FontsPrgrm.Java
import java.awt.GraphicsEnvironment;
```

```
class Fontsprgrm
{
  public static void main(String [] args)
  {
    GraphicsEnvironment ge = GraphicsEnvironment.getLocal
          GraphicsEnvironment();
    String F[] = ge.getAvailableFontFamilyNames();
    Sop("no.of fonts = "+f.length);
    // printing the fonts
    for (int i=0; i<f.length ; i++)
    {
      sop(f[i]);
    }
  }
}
```

→ In the above program GraphicsEnvironment is one of the predefined abstract class present in java.awt.* package.

→ GraphicsEnvironment .class contains the following factory method

Public Static GraphicsEnvironment getLocalGraphicsEnvironment()

Eg:- GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();

→ To get the fonts from the current working machine we use the following method which is present in GraphicsEnvironment class

Public String [] getAvailableFontFamilyNames()

Eg:- String f[] = ge.getAvailableFontFamilyNames();

3/8

:- Packages :-

In any programming language there exist a library where it contains common features for most of the programmers.

For example, in C-programming whose library contains collection of header files & it is a collection of pre-defined/common funns. In C s/w development if any function is common for most of the programmers then such type of funns will be placed in header files. In otherwords, header files always contains common funns.

Similarly, in java programming, it contains rich set of API (Appl programming Interface)

Def :- An API is a collection of packages. A package is a collection of classes, interfaces & sub packages. A subpackage interns contains collection of classes, interfaces & sub sub packages. etc.

* The purpose of package concept is that when the class & interface is common for most of the java programmers then those common classes & interfaces must be placed in the package. In otherwords, packages are always containing common classes & interfaces.

Advantages of Packages :-

If we develop any java appl with the concept of packages then we get the following advantages.

1. Appl development time is less
2. Appl memory space is less

3. App execution time is less (2CT)

4. App performance is enhanced.

5. App redundancy of code is minimized.

6. Able to get the slogan of Java (WORA)

7. we are able to differentiate/provide uniqueness among duplicate classes.

Q:- What is the difference btw inheritance & Package

Ans:- Inheritance concept always makes us to understand how to reuse the features within the program btw class to class, interfaces to interface & interfaces to class but not across the programs.

Packages concept allowses to reuse the features both within & across the programs. btw class → class, interfaces → interface & interfaces → class.

Types of Packages in Java:-

We've 2 types of packages they are

1. Pre-defined/Builtin Packages
2. user/programmer/custom defined packages.

Pre defined packages :-

These are developed by SUN Micro system developers available as a part of Java s/w & whose role is to deal with universal requirements.

user defined packages are developed by java programmers available as a part of java project & they always deals with common requirements.

Types of Pre defined Packages:-

We've 3 types of pre defined Packages. They are

(i) J2SE(Core) Package :-

The purpose of these packages is to develop client side appls.

(ii) J2EE (advanced) Packages:-

These packages are used for developing server side appls.

(iii) J2ME (Mobile/micro) Packages:-

These packages are used for developing mobile/wireless appls.

List of pre defined J2SE Packages :-

J2SE contains 8 essential packages. They are

1. Java.lang.* :-

It is one of the pre defined package & it is always used for obtaining language facilities/services.

Some of the language facilities are.

a. Accepting cmd line arguements
b. Data conversions
c. Obtaining the garbage collection facility
d. Developing the thread based appls
e. Developing the exception based appls. etc.

By default this package is imported to each & every java program & hence this package is called default package.

2. Java.awt.* :- (Abstract windowing Tool Kit) :-

The purpose of this package is to design GUI appy (looks&feel)

To design the GUI appl^n we need GUI components which are available in the form of pre defined classes.

Ex :- Label, Button, checkbox, TextField etc

Creating a GUI component is nothing but creating an object of an appropriate pre defined class.

Ex:- Create a Button with a name Save

```
Button b1 = new Button ("Save");
```

Save

3. Java.awt.event.* :-

Here event is the sub package of awt package. Event package contains collection of classes & interfaces & they provides functionality / behaviour to the GUI appl^n.

To develop the complete GUI appl^n we must import Java.awt.* & Java.awt.event.*

4. Java.io.* :- (File programming)

The purpose of this package is to achieve the data persistency by using the concept of files.

Def^n of Stream :-

The flow of data btw main memory & secondary memory is known as stream.

5. Java.applet.* :- (Applet programming)

The purpose of this package is to develope distributed appl^ns where they can run in the context of browser & whose results are sharable across the universe.

To develop the distributed appl^ns, in the initial versions of Java Sun ms developers developed a concept called applets

To fulfill the concept of applets, we have predefined class called Applet present in Java.applet.*

Defn of applet :-

An applet is one of the Java Program which are running in the context of browser & whose results are sharable accross the universe.

6. Java.net.* :- (Network programming)

The purpose of this package is to develop n/w ing appls (or) client server appls for sharing the data btw multiple machines which are located either in same n/w (or) in different n/w.

For developing client & server side appls there exist seperate classes & interfaces which are present in Java.net.* package.

7. Java.util.* :- (Collection Frame work)

The purpose of this package is to develop high performance Java based appls.

Def of collection Frame work:-

It is one of the standardized mechanism which allowes to group multiple values either of same type (or) different types (or) both the types in a single variable with a dynamic size. This single variable is called collection Frame work variable.

8. Java.text.* :- (Text processing)

This package is used for 3 puposes. They are

(i) Formatting of dates

(ii) Formatting of times

(iii) Numeri cal manipulations like truncating, rounding, ceiling, floating etc.

In otherwords, this package we use most of the times in report generation modules.

Development of user/programmer defined packages:-

The purpose of creating user defined package is that to place common classes & interfaces. In otherwords, if a class (or) interface is common for most of the programmers then such type of classes & interfaces must be placed in a package.

What are all the guiddines are followed by SUN MS developers for developing pre defined packages, same guidelines will be followed by Java programmer for developing user defined packages.

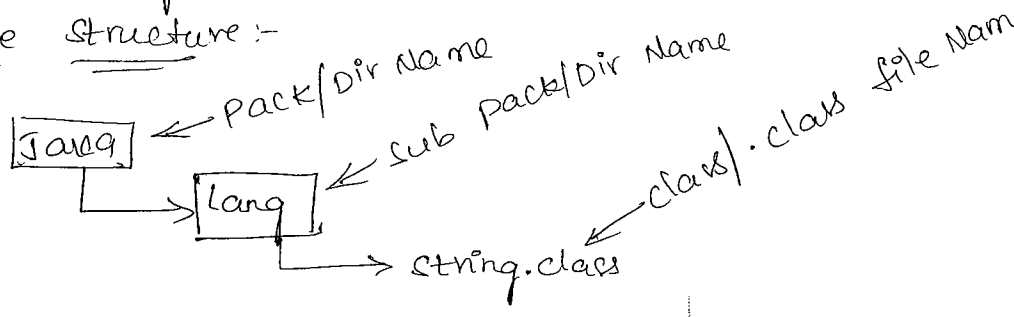
5R Creating a user defined package is nthg but creating it as a directory/folder in current working machine.

consider the following statement

Java.lang.String

Java point of view	OS Point of View
Java – upper package	Java – Root Directory
lang – sub package	lang – sub Directory
String – ClassName	String – .class File

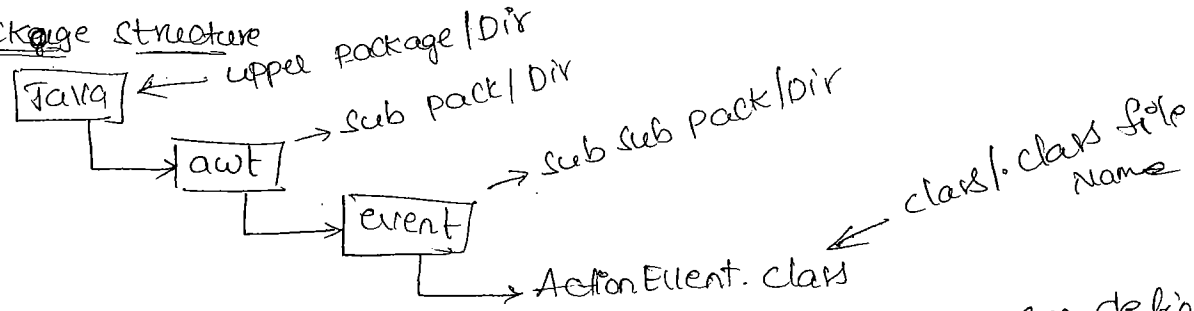
Package Structure:-



Consider the following statement.

Java.awt.event.ActionEvent

Package structure



No. of phases required for Developing & using user defined packages:-

To develop & use the user defined packages we've 4 phases. They are.

1. create a package
2. place the classes & interfaces defns in the package
3. compile the classes & interfaces of a package.
4. use the classes & interfaces of a package.

Phase:-1

creating the package.

To create a package we use the following syntax.

```
Package  pack1 [pack2 [.... [packn]]];
```

Explanation:-

1. Here package is one of the keyword used for creating user defined packages.
2. pack1 pack2 .... packn represents Java valid variable names treated as UDP.
3. pack2, pack3 ... packn represents name of the sub packages
. . . compilation is optional

4. Pack1 represents name of the upper package & whose specification is mandatory.

Ex:- Package P1;
(or)
Package P1.P2;
} Package Statement

Rule :-

When we are placing class def" (or) interface def" in the package, the packages statement must be 1st executable stmt otherwise we get compile time error.

Phase:2

Steps for placing class/interface def" in the package :-

To place class & interface def"s in the package we follow the following steps.

1. Choose an appropriate package name for placing common classes & interfaces & ensure that the package stmt must be 1st executable stmt.
2. Choose an appropriate class/interface name & ensure that whose modifier must be public
3. The modifier of the constructor of the class which is present in the package must be public (this rule is not applicable in the case of interfaces)
4. The modifier of the method of the class / interface which is present in the package must be public (this rule is optional in the case of interfaces)
5. Whatever the class/interface name we are placing in the package must be given as a file name with an extension .Java.
6. At any point of time in one package we must define either a class def" (or) interface def" otherwise we get file recognization problem at compilation time (Becoz multiple public classes/interfaces never permitted

EX:- create a package tp & place a class Test.

```
// Test.Java ——— 5
  package tp; ——> 1
   Public class Test ——> 2
   {
     Public Test() ——> 3
     {
       sop("Test--DC");
     }
     public void disp() ——> 4
     {
       sop("Test--disp()");
     }
   } // Test --- common BLC
```

EX:- create a package tp & place an interface ITest

```
// Itest.Java // ——> 5
  package tp; // ——> 1
   Public interface ITest // ——> 2
   {
                    // ——> 3 not applicable
     Public void show () // ——> 4 optional
     {

     }

   }
```

Phase :-3

compiling classes & interfaces of a package

To compile the package classes & interfaces we use the following syntax.

```
Javac -d . FileName.Java
```

Explanation :-

1. Here -d is one of the directory option/switch, which gives an indication to the Java environment saying that go to file name.Java, take the package name & create it as a directory automatically in the current working machine provided the following conditions must be satisfied.
   i, FileName.Java should not contain any errors
   ii, Earlier the package name should not be created as a directory (if it is already created as a directory then it will be carry forwarded without re creating it) & will it be considered as a current directory.
2. Javac is a tool used to compile .Java programs & generates .class files.
3. dot(.) refferes current directory. Due to dot(.), currently generated .class files will be automatically copied into currently created directory (package name in the java program)

EX:- compile Test.Java

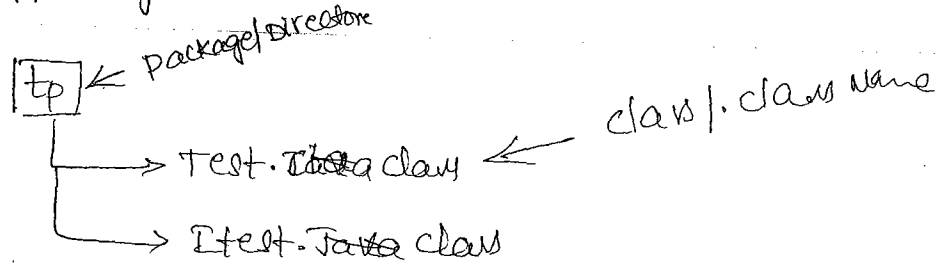
E:\\ 11-AmJava\packages> Javac -d . Test.Java

tp ← pack/Dir

→ Test.class ← class/.class Name

Ext compile Itest.Java (216)

E:\\IAmJava\packages > Javac -d . Itest.Java



The fully qualified names of Test class & Itest interface are tp.Test & tp.Itest.

* Write a Java program which makes use of Test class of tp package.

```
// PackDemo1.Java
import tp.Test;
class PackDemo1
{
  public static void main(String [] args)
  {
    Test t1 = new Test();
        t1.disp();
  }
}
```

* write a Java program which makes use of Itest interface of tp package.

```
// PackDemo2.Java
class PackDemo2
{
  public static void main(String [] args)
  {
    tp.Itest io = new tp.Itest();
  }
```

```
        public void show()
        {
            sop("show() --overridden-- AINC");

        }
    };
    io.show();
  }
}
```

Phase :- 4

NO. of Phase to use the classes & interfaces of a package

In Java programming we've 2 ways to use the classes & interfaces of a package.

1. By using import statement
2. By using fully qualified name/canonical form approach

By using import statement :-

The purpose of import stmt is that either reffering of all the classes & interfaces of a package (or) a particular class (or) interface of a specific package in the current Java program.

The import stmt can be used in various approaches. They are.

Approach :- 1

This approach makes us to understand how to refer all the classes & interfaces of a particular package in the currently working Java Program.

SYNTAX:

```
import pack1[.pack2[...[pack n]].*;
```

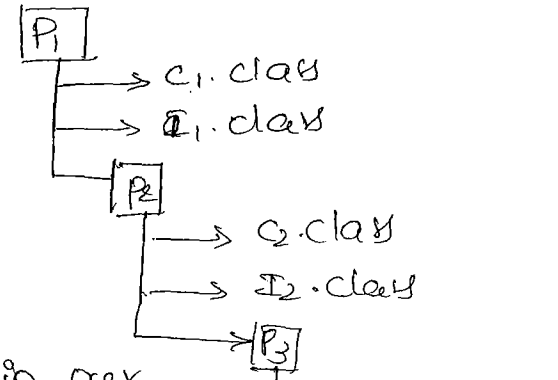
Here "*" is called a wild character which gives an indication to Java environment saying that provide all the classes & interfaces present in the specified package at both compile & execution time.

Ex:-

import P1.*; → ①

import P1.P2.*; → ②

import P1.P2.P3.*; → ③



→ If we use stmt ① in our Java program we can refer all the classes & interfaces of P1 package but not the classes & interfaces of P2 & P3 packages.

→ When use stmt ② in our Java program we can refer all the classes & interfaces of P2 package but not the classes & interfaces of P1 & P3 packages.

18 Approach :-2

This approach makes us to understand how to use a class (or) interface of a particular package.

Syntax :-

```
import pack1[.pack2[....[.pack n]]].class/intf name;
```

Ex:- import P1.C1; → ①

import P1.P2.I2; → ②

import P1.P2.P3.C3; → ③

→ When we use stmt ① in our Java program we can use/refer only the class C1 but not other classes & interfaces of same package P1.

Industry is always recomended to use only approach-2 becoz [illegible] classes & interfaces & not recomended to use approach 1

results in more memory space.

*** Static import statement :- (SIS)

It is one of the new facility present in JDK 1.5 version onwards.

The purpose of SIS is that eliminating the redendant reffering of class names/intf name before static data members & eliminating the redandant reffering of class name before static method names

Approach :- 3

This approach applicable to static data members which are present in classes & interfaces. This approach makes us to refer directly in the SDM's without preceeded by class/intf name

Syntax :-

```
import static pack1[.pack2[....[.packn]]].cls|intf name.SDM Name;
```

Approach :- 4

This approach applicable to static methods present in class. This approach makes us to refer static methods directly without preceeded by cls name

Syntax :-

```
import static pack1[.pack2[....[.packn]]].clsName.SMName;
```

Approach :- 5

This approach makes us to understand refer the static features (SDM & SM) directly without preceeded by cls/intf Name.

Syntax :-

```
import static pack1[.pack2[....[.packn]]].cls|intf name.*;
```

* Write a Java program which illustrate the concept of static import stmt

```
// SIMDemo.Java
/* import static ip.Inst.name;
   import static ip.Inst.loc;
   import static ip.Inst.crs;  (OR)*/
   import static ip.Inst.*;
/* import static java.lang.Math.PI;
   import static java.lang.Math.E;
   import static java.lang.Math.pow;
   import static java.lang.Math.sqrt; (OR) */
   import static java.lang.Math.*;
   import static java.lang.system.out;
Class SIMDemo
{
  public static void main(String [] args)
  {
    out.println("Inst Details");
    out.println("=========");
    out.println("Inst Name="+name);
    out.println("Inst Loc="+loc);
    out.println("Inst course="+crs);
    out.println("==========");
    out.println("Math Details");
    out.println("=========");
    out.println("Val of Pi="+PI);
    out.println("Val of e="+E);
    out.println("pow(7,4)="+pow(7,4));
    out.println("pow(4,2)="+pow(4,2));
    out.println("sqrt(625)="+sqrt(625));
```

Q:- What is the role of static keyword in java?

We apply the static keyword in the following levels

1. At data member level called static data member (SDM)
2. At method level called static Methods (SM)
3. At block level called static block (SB)
4. At import stmt level called static import (SIM)

Q:- What are the differences btw import & package keywords?

import	package
→ import is a keyword used for reffering classes & interfaces	→ package is a keyword used to create user defined packages of a package.
→ We can write any no. of import stmt in a java program after package stmt.	→ package stmt should appear only once in a java program
→ We can apply static keyword during import stmt	→ we can't apply static keyword during package creation.

By using fully qualified name approach:-

→ A class name (or) interface name is qualified by its complete package hierarchy then it is called fully qualified name of a package

→ It is an alternative approach for reffering the a class & intf instead of using import stmt.

→ If we refer a class / intf name multiple times with this approach in all the times they must be preceeded by package name otherwise we get compile time error.

Syntax:-

Pack1[.Pack2[. . . [.Packn]].class/intf name

Ex1:-

```
class Sathya extends P1.C1
{
  =
}
```

FQNA (Fully qualified name approach)

Ex:-2

```
class Hyd implements P1.P2.P3.I3
{
  ==
}
```

FQNA (Fully qualified name approach)

Ex:-3

```
P1.P2.C2 O2 = new P1.P2.C2();
```

FQNA [fully qualified name approach]

2018

Access Specifiers (or) Modifiers :-

AM are applied at 2 levels

① At class/intf level:- Public, abstract, final, Public abstract, public final } derived AM's

Access modifiers are those which are applied before the class defn & interface defn whereas access specifiers are those which are applied before the data members & methods.

② At features of class level :- Before the features of class level we can apply these AM's

In Java programming we have 4 types of Access specifiers. They are,

1. private.
2. Default (not a keyword)
3. protected
4. public

If we are not writing using either private (or) protected (or) Public keywords then JVM is by default treated as "default" access ~~specifier~~ modifier.

The purpose of Access ~~specifiers~~ modifiers is that how to access the features where to access the features & where not to access the features within & across the packages. Between class to class, interfaces to interface

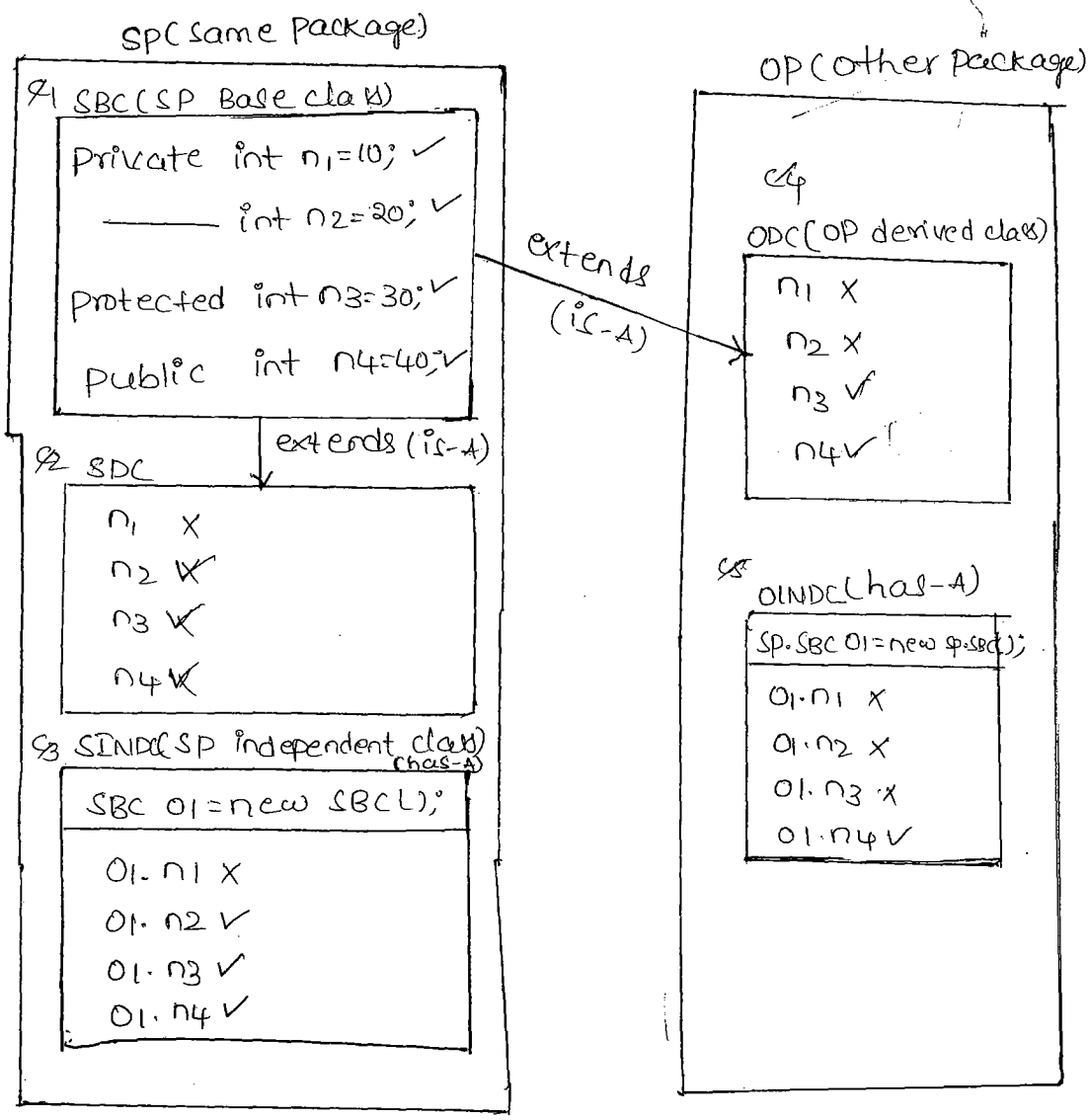
Features controlling mechanism

Access specifiers always makes us to understand where to access the features & where not to access the features (OO)

Access specifiers always provides features controlling accessing mechanism.

Rules for Access ~~specifiers~~ modifiers :-

Following diagram gives rules for access ~~specifiers~~ modifiers.



NOTE:- * Private AM is also known as native AM
* Default AM is also known as package AM
* protected AM is also known as inherited AM
* public AM is also known as universal AM.

Access ~~specifier~~ modifier protection matrix:-

Category of classes / Category of access modifier ~~specifier~~ protection	Same package BC	Same package DC	Same package INDC	Different (or) other package DC	Different (or) other package INDC
private	✓	✗	✗	✗	✗
default	✓	✓	✓	✗	✗
protected	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓

✓: Accessible/ visible , ✗ : Not accessible/ Invisible

Write a Java program which illustrate the rules ~~for~~ of access ~~specifiers~~ modifiers.

```
// sbc.java
Package sp;
Public class sbc
{
    private int n1=10;
            int n2=20;
    protected int n3=30;
    public int n4=40;
    Public sbc()
    {
        sop("val of n1="+n1);
        sop("val of n2="+n2);
        sop("val of n3="+n3);
        sop("val of n4="+n4);
    }
```

sp
→ sbc.class

cmd>javac -d . sbc.java

```
package sp;
public class sdc extends sbc
{
    public sdc
    {
      sop("val of n1="+n1);
      sop("val of n2="+n2);
      sop("val of n3="+n3);
      sop("val of n4="+n4);
    }
}
```

10|11|13 — 10,000 - 2/-

13|2|14 — 10,000

200
16
3200 — 13,200

100 - 2
1000 - 20
10,000 - 200

200
19
3800 — 13,800

13,800
13,200
27,000

20,000
7,000
27,000

3,800

27
27
54

27
19
6

control Structure.

① Branching stmts / conditional stmt / selection stmts

Ex:- if, if else, Switch

② Looping stmt / Repetative stmt / Iterative stmts

Ex:- while, do while

③ Misloneous stmts

Ex:- Break, continue, go to

10 - X - w
9 - IX - if
5 - V - if
4 - IV - if
1 - I - w

100 - C - w
90 - XC - if
50 - L - if
40 - XL - if

1000 - M - w
500 - D - if
900 - CM - if
400 - CD - if

10/8

Exception Handling

→ The aim of exception handling is to build strong appls.

→ To develop any project (or) any appll as the part of real world we use some programming language. If we write any program with certain type of language then we get 3 types of errors. They are 1. compile time errors 2. logical errors, and 3. Run time errors.

Compile time Errors :-

→ These errors occurs during compilation phase due to syntaxes are not followed by the programmer.

→ These errors solved by the programmers during development level.

Logical Errors :-

→ These errors occurs during Execution/Run time becoz of mis interpretation/wrong representation of logic.

→ logical errors always gives inconsistant/wrong results

→ logical errors solved by the programmer at development level.

Run time Errors :-

→ These errors occuring during execution/run time

→ These errors are occuring due to invalid/wrong input entered by appll users.

→ Run time errors solved by the programmers at development level by having forecasting knowledge (or) at implementation level.

→ The languages like C, C++, COBOL etc & whose appls are treated as weak becoz these languages are not containing a facility called exception handling.

The languages like Java & .Net & whose appl'ns are treated as Robust becoz these languages contains an effective facility called exception handling.

In general to any project/appl' which is developed with any language/technology & if the appl' user enters invalid input they are by default generates system/technical error messages. which are not understandable by appl' users. & understandable by only programmers. which is not a recomended process in s/w development.

Hence, industry is highly recomended to build the strong appl'ns by making use of exception handling & convert system error message into user friendly error messages.

Points to be remembered :—

→ When the appl' user enters invalid input, we get run time errors.

→ Run time errors always generates system error messages.

→ Run time errors of Java program are known as exception.

→ Exception always generates system error messages.

** → When the exception occurs in the Java program, program execution is abnormally terminated, cpu control comes out of the program flow & JVM generates system error message

→ Def of Exception Handling :-

The process of converting system error messages into user friendly error messages is known as exception handling.

* → Each & every task/operation/action must be performed w.r.t object in Exception handling

WKT whenever an exception occurs in the Java program, program execution is abnormally terminated, CPU control comes out of program flow & JVM generates system error message (SEM).

Generating SEM is considered as an operation & it must be performed wrt an object & every object need a class & this class is called exception class.

The purpose of exception class is to terminate the program execution abnormally, JVM will create an object of exception class & generates SEM. As a Java programmer we use the same exception class & we generate user friendly error msg. which is one of recomended process in S/w development.

In general for every common problem (Invalid i/p), there exist exception class.

Types of Exceptions

In Java programming we've 2 types of exceptions. They are

1. Pre defined/Built in exceptions
2. User/Programmer/custom defined exceptions

Predefined exceptions developed by SUN MS developers, supplied as a part of Java S/w & whose role is to deal with universal problems.

Some of the universal problems are

a. Division by zero problem (ArithmeticException)

b. Invalid number format problem (NumberFormatException)

c. Invalid bounds of the array (ArrayIndexOutofBoundsException) etc

User defined exceptions developed by Java programmer, supplied as a part of their Java project & whose role is to deal with common problems.

Some of the common problems are

a. Attempting to enter invalid PIN number in ATM based appl.
b. Attempting to enter negative salary for an employee.
c. Attempting to enter wrong user name & password in authentication based appl.
d. Attempting to deposit invalid amount
e. Attempting to withdraw more amount than existing.

Types of pre defined exceptions :-

pre defined exceptions are classified into 2 types. They are

1. Asynchronous exceptions
2. Synchronous exceptions

Asynchronous exceptions :-

These exceptions deals with H/w problems & external Problems.

Ex :- mouse/ keyboard/ mother failures
Memory problems } H/w problems

power failures — External problem.

At present SUN MS/ oracle corp developers did not develop complete API for develop dealing with Asynchronous exceptions (partially developed).

For dealing with all asynchronous exceptions in Java for near feature we've a pre defined class called

Java.lang.Error

- → StackOverFlowError
- → ⋮
- → etc

Synchronous exceptions :-

These exceptions deals with programatic run time errors synchronous exceptions are classified into 2 types. They are

1. Checked exception
2. Un Checked exception

Checked exception :-

Def :- 1

Checked exceptions are those which are the direct sub classes of Java.lang.Exception (or)

Def :- 2

The exceptions which are occuring at run time will be showing them as errors at compile time.

Ex :-

Java.lang.Exception

Checked

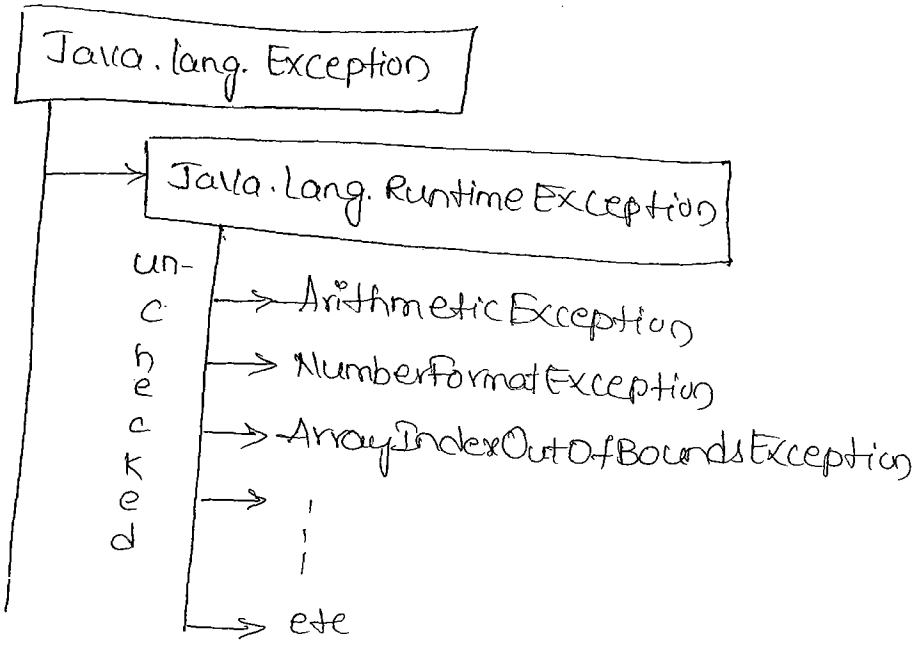
- → FileNotFound Exception
- → Interrupted Exception
- → IOException
- → ClassNotFoundException
- ⋮
- → etc

12/8

Unchecked exception :-

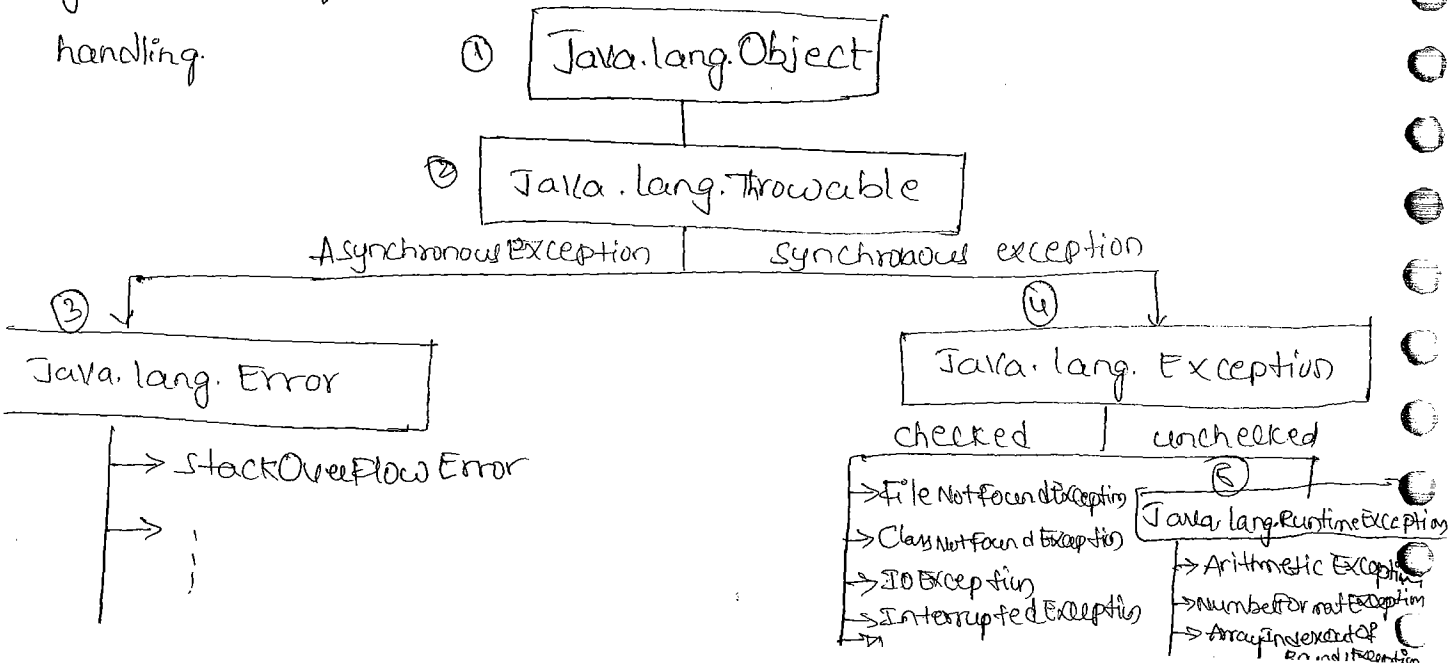
Def :- 1 unchecked exceptions are those which are the direct sub classes of Java.lang.RuntimeException (or)

Def:- The exceptions which are occuring at run time will be showing them as errors at run time only but not at compile time.



Exception Handling Hierarchy Chart :-

Hierarchy Charts in Java programming makes us to understand how the features are inheriting from top most base class (TBC) to intermediate base class (IBC) & from IBC to Bottom most derived class (BDC). The following diagram gives list of exception classes available in hierarchy of exception handling.



① Java.lang.object is the super class for all the classes in Java & it provides garbage collection facility for collecting unused memory space to improve the performance of Java based appl.

② Java.lang.Throwable is one of the pre defined super class for all the exceptions in Java. & it decides what type exception occurs in the Java program (either asynchronous (or) synchronous)

③ Java.lang.Error is the super class for all Asynchronous exceptions.

④ Java.lang.Exception is the super class for all synchronous Checked exceptions

⑤ Java.Lang.RuntimeException is the super class for all synchronous unchecked exceptions.

Programatically, eventhough there exist multiple exceptions in the Java program, at any point of time either JVM (or) Java programmer can generate & process only one exception but not multiple exceptions.

If an exception occurs in Java program JVM will create an object of exception sub class & generates system error message by default.

Technical Def^n of exception :-

An exception is an object occurs at run time which will describe nature of the message. The nature of the message can be either system error message (or) user friendly error message

Ex:-

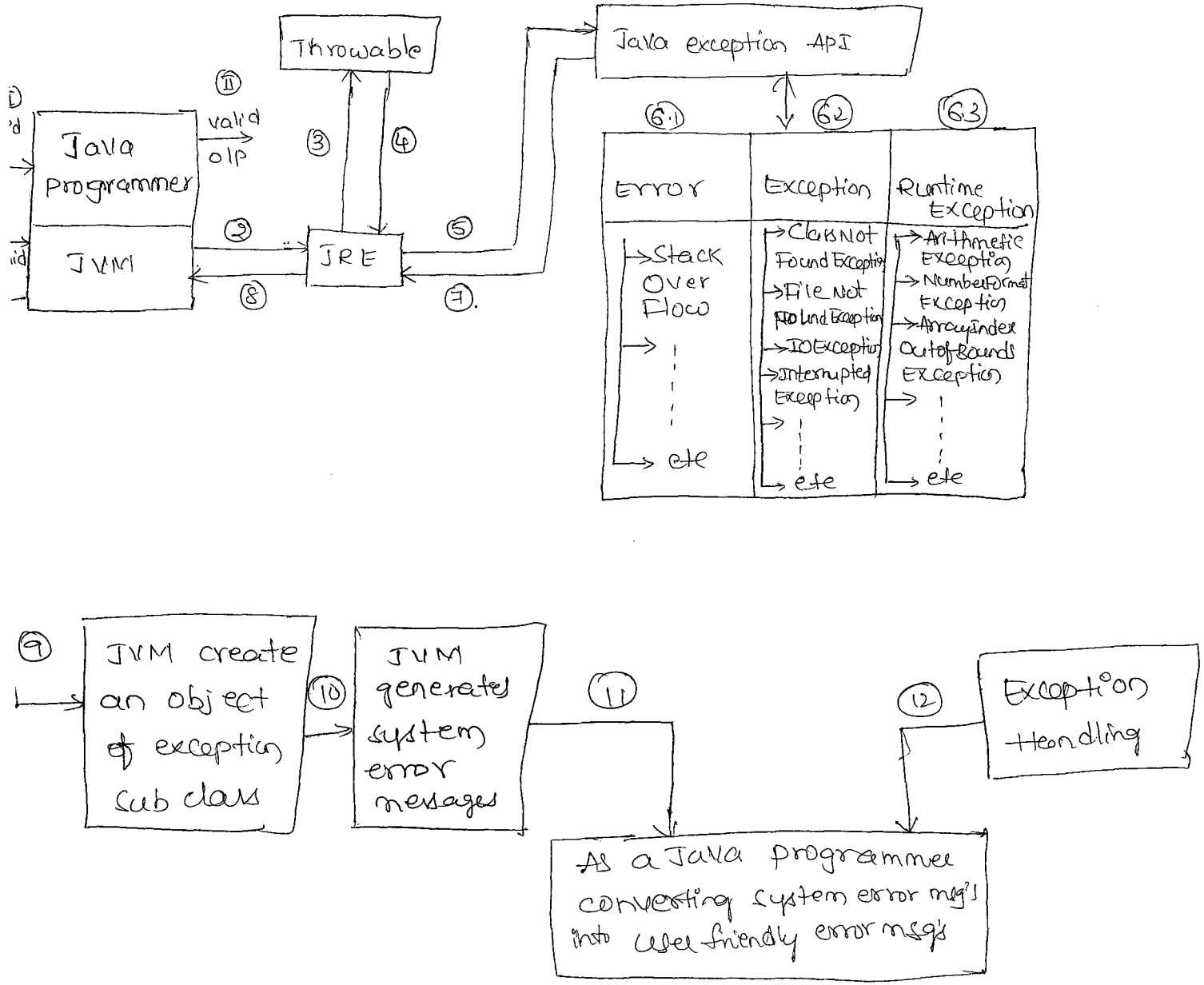
```
ArithmeticException ae = new ArithmeticException(" /by zero");
```

ArithmeticException ae = new ArithmeticException(" /by zero");

- ArithmeticException → Exception sub class
- ae → Object
- new → Run time
- ArithmeticException(" /by zero") → Single parameterized constructor by taking string as a parameter

Internal flow of exception occurance:-

Whenever an exception occurs in the Java program it is mandatory to the Java program to know internal flow of exception occurance and it is given in the following diagram.



18/5

I. Appl user enters valid i/p to the Java programmer (Sol)

II. JVM can process the Java program with valid i/p & gives valid o/p.

1. Appl user enters invalid i/p to the Java programmer.

2. JVM can't process the Java program with invalid i/p & contacts to JRE (JRE = JVM + Java-API)

3. JRE communicates with Java.lang.Throwable for finding what type of exception occurs in the Java program.

4. Java.lang.Throwable is one of pre-defined super class for all the exceptions in Java. & it decides what type of exception occurs in the Java program (either synchronous (or) Asynchronous) & gives same msg back to JRE.

5. JRE communicates with Java exception-API for obtaining exception sub class.

6.1, 6.2, 6.3. Java exception API executes either (6.1) (Java.lang.Error for Asynchronous exceptions) (or) (6.2) (Java.lang.Exception for synchronous checked exception) (or) (6.3) (Java.lang.RuntimeException for synchronous unchecked exception)

7. Java exception-API gives the exception sub class to JRE

8. JRE gives exception sub class to JVM.

*9. JVM will create an object of exception sub class

Ex:- For division of 2 numbers program if we enter 10 & 0 as an i/p then JVM will create an object of ArithmeticException. Programatically, ArithmenticException ae = new ArithmeticException("\by zero");

10. JVM will generate system error message after creating an object of exception sub class.

Ex:- The system error msg for the exception which occured in the Ex of

S.E.M

```
exception in thread main ----------
Java.lang.ArithmeticException : /by zero ----
----------
```

It is not recomended to generate system error msg's in real world application & highly recomended to generate user friendly error msg's.

Q12. As a Java programmer we will convert the system error msg's into user friendly error msg's by making use of Exception handling.

Ex:- The user friendly error msg for the system error msg in the Ex of step NO:-10 is shown below

U.F.E.M

```
Don't enter zero for denominator ----
```

Handling the exceptions

→ Handling the exceptions are nothing but converting system error messages into user friendly error messages.

→ For handling the exception we've 5 Keywords. They are

1. try
2. catch
3. finally
4. throws
5. throw

Syntax for handling the exception :-

Syntax:-

```
try
{
  Block of stmt(s) - which are
  causing problems at runtime
} catch (Type of exception1 obj1)
  {
    Block of stmt(s) - which
    provides user friendly error msg's
  }
catch(Type of exception2 obj2)
  {
    Block of stmt(s) - which provides
    user friendly error msg's
  }
    :
catch(Type of exception-n obj-n)
  {
    Block of stmt(s) - which provides
    user friendly error msg's
  }
finally
  {
    Block of stmt(s) - which will
    execute compulsarily
}
```

try Block:-

→ It is the block in which we write the block of statements which are causing problems at run time. In otherwords the problematic statements which are generating exceptions at runtime must be written in try block & it is called "exception monitoring block."

→ If any exception occurs in try block then JVM control comes out of try block & executes appropriate catch block.

→ After executing appropriate catch block, JVM control never goes to try block even we use return statement in the catch block.

→ Each & every try block must be immediately followed by catch block. i.e., we should not write any intermediate statements btw try & catch blocks

→ Each & every try block requires atleast one catch block otherwise we get compile time error (It is required to write more no. of catch blocks for generating multiple user friendly error messages to make the appl. strong)

→ In some of the circumstances one try block can contain another try block i.e., nested/Inner try blocks can be permitted.

catch block :-

→ It is the block in which we write the block of statements which will provide user friendly error messages. In otherwords this block supresses system error messages & generates user friendly error messages & this block is called 'exception handler block".

→ catch block will execute when an exception occurs in try block

→ Eventhough there exist multiple catch blocks, JVM can execute an appropriate catch block depends on type of exception occured in the block.

*→ In the catch block as a Java Program we declare an object of exception sub class & internally, exception sub class object refferenced by JVM.

Ex:-

```
try
{
  int c=10/0;
} catch (ArithmeticException ae)
  {
  }
```

Declaration of exception sub class written by Java programmer

Internally,

ArithmeticException ae = new ArithmeticException("/ by zero");

Refference of exception sub class done by JVM.

→ In some of the circumstances in the catch block one can also write try & catch blocks.

finally block:-

→ It is the block in which we write the block of statements which will relinquish (release/ terminated / close) the resources (files & Databases) which are obtained in try block.

→ finally block will execute compulsarily.

→ Writing finally block is optional.

→ If at all we write finally block, it should be written after catch block (if catch block present) (or) after try block

→ In some of the circumstances in the finally block one can write try & catch blocks.

→ It is highly recomended to the Java programmer to write one finally block per Java program to relinquish the resources.

* Write a Java program which illustrate try block catch block & finally block.

```
//Div2.Java

class Div2
{
  Public Static void main(String [ ] args)
  {
    try
    {
      String s1= args[0];
      String s2= args [1];
      int x=Integer. ParseInt (s1);
      int y= Integer ParseInt(s2);
      int z=x / y;
      Sop (" division in Div2="+z);
    } catch(ArithmeticException ae)
    {
      sop(" system. err. println("don't enter zero for den--");
    }
    catch (NumberFormat Exception nfe)
    {
      system. err. println ("don't enter alpha- numeric values");
    }
    Catch (ArrayIndexOutOf BoundsException ab)
    {
      system. err. Println (" enter two values");
```

```
finally
{
  sop("i am from finally block");
}
}// main()
} // Div2
```

<u>Case :- 1</u> if (exception occurs) then

* A part of try block will execute
* An appropriate catch block will execute
* A finally block will execute (If we write)

<u>case :- 2</u> if (exception doesn't occurs) then

* A complete try block will execute
* A finally block will execute (If we write)

<u>Q :- 1</u> Determine the o/p of the following program

```
//Exam1.Java
class Exam1
{
 public static void main(String [] args)
  {
   try
    {
      sop("Java");
      int x = 100/0; // exception stmt
      sop(" val of x = "+x);
    }
   finally
   {
     sop("i am from finally");
   }
```

Output:-

```
Java
i am from finally
Java.lang.ArithmeticException : / by zero
```

Explanation :-

If any Java program contains try block & finally block only & if an exception occurs in try block then JVM will execute the following.

a, The stmts appears before exception stmt

b, Stmts written in finally block

c, exception related messages.

Q:- Determine the o/p of the following program.

```
//Exam2.Java
class Exam2
{
  public static void main(String [] args)
  {
    try
    {
      sop("Java");
      int x = 100/0;
      sop("val of x="+x);
    }
    finally
    {
      sop("iam from finally");
    }
    sop("sathya");
    sop("hyd");
}
```

output:-

Java
i am from finally
Java.Lang.Arithmetic Exception:\by zero

Explanation

If any Java program contains try, & finally blocks only & stmts followed after finally block & if an exception occurs in try block then JVM will execute the following

a. Stmts appears before exception stmt
b. Stmts written in finally block
c. Exception related messages but JVM will never execute the stmts appears after finally block.

17/8
Q:-3 Determine the o/p of the following program.

```
//Exam3.Java
class Exam3
{
 Public Static void main(String [] args)
 {
  try
  {
   sop("Java");
   int x = 100/0; // exception statement
   sop("val of x =" + x);
  } Catch (Arithmetic Exception ae)
  {
   System.err.Println("don't enter zero for den...");
  }
  sop("hello1");
  sop("hello2");
 } // main()
} // Exam3
```

output :-

Java

Don't enter zero for den. —

hello1

hello2

Explanation :-

If any Java program contains a try block, catch block & the block of stmts after the catch block ~~then~~ and defined if an exception occurs in the Java program then JVM will execute the following (a) The stmts appears before the exception stmt.

(b) stmts appears in catch block

(c) stmts appears after catch block.

NOTE :- If any Java program contains try block, catch block & block of statements after catch block and if an exception occurs in Java program & not matching with catch block then JVM will execute the following.

(a) statements appears before exception statement

(b) exception related messages

(c) But not executing the statements appears after catch block.

* Decide the output of following program.

```
//Exam5.Java
class Exam5
{
  public static void main(String [] args)
  {
    try
```

```
    sop(" Java");
    int x=10/0;
    sop("val of x="+x);
    } catch(ArithmeticException ae)
     {
       System.err.println("don't enter zero for den...");
     }
    finally
     {
       sop("i am from finally");
     }
    sop("hello1");
    sop("hello2")
  }
}
```

output:-

```
Java
don't enter zero for denominator
i am from finally
hello1
hello2
```

Explanation :-

If any Java program contains try block, catch block, finally block & stmts appears after finally block & if an exception occurs in the Java program then JVM will execute the following.

(a) The statements executes before exception statement

(b) Statements appears in appropriate catch block

(c) The statements written in finally block

(d) The statements appears after finally block

NOTE:- We should not write consequetive multiple finally (2qs) blocks. & also we should not write consequetive try blocks.

THROWS

In the context of exception handling we've 2 types of methods. They are

1. Specific method
2. Common method

→ With specific methods we always fulfill specific requirements.

→ With common methods we always fulfill common requirements

→ If any exception occurs in specific method then it is highly recomended to the programmer to write try & catch blocks for handling the exceptions. But one is not recomended to write "throws" keyword.

→ If any exception occurs in common method then such exceptions should not be handled but ~~they~~ it must be always send always the exceptions to specific method by using "throws" keyword.

Def"/ Purpose -1 of throws :-

"throws" is a keyword which will describe/ express the exceptions which are occuring in common method.

Def"/Purpose-2 :-

"throws" is a keyword which gives an indication to the specific method to place common exception method under try & catch block for generating user friendly error messages

→ The place of using "throws" keyword is always as a part of common method heading

→ If a common method heading contains "throws" keyword then it is called "common exception method". otherwise it is called as "common normal method".

Rules of throws keyword :-

Rule :-1

If "throws" keyword is throwing checked exception then it is mandatory to the programmer to place common exception under try & catch block. otherwise we get compile time error.

Rule :-2

If "throws" keyword is throwing unchecked exception then it is optional to the programmer to place the common exception under try & catch block.

Rule :-3

Irrespective of ~~throws~~ type of exception "throws" keyword is throwing then industry is highly recomended to the programmer to place the common exception method under try & catch block. for generating user friendly error messages.

18/8

* Write a Java program which illustrate "throws" keyword which is used as a part of pre defined method

Consider the following statement

Java.lang.Integer
↓
Public static int ParseInt (String) throws NumberFormatException

Here parseInt method is one of the common exception method & it must be placed always with in try & catch block for generating user friendly error messages

```
import Java.util.Scanner;
class Sathyaprog1
 {
   Public Static void main(String [] args)
    {
      Scanner s=new Scanner(System.in);
      Sop("Enter the marks");
       String marks=s.nextline();
      try
      {
      int im=Integer.ParseInt(marks); // common exception method
         Sop("Marks="+im);
      }Catch (NumberFormat Exception nfe)
      {
       System.err.println("don't enter Alpha-numeric values for marks")
      }
      finally
      {
       Sop("I am from finally block");
      }
    }
 }
```

* Write a Java program which illustrate the concept of "throws" which is used as a part of user defined method.

```
// Sathya.Java
  Package sp;
  Public class sathya
  {
    Public void division (string s1, string s2) throws
    ArithmeticException, NumberFormatException
    {
      int a = Integer.ParseInt(s1);
      int b = Integer.ParseInt(s2);
      int c = a/b;
      sop(" div in sathya = "+c);
    }
  } //Sathya
```

Compile the above program

```
javac -d . Sathya.java
```

```
[sp]
 └→ sathya.class
```

In sp.sathya class the division method is one of the user defined common instance exception method & it must be always written with in try & catch block for generating user friendly error messages.

* Write a java main program which makes use of division method of sp.sathya class

```
//Sathyaprog2.java
import java.util.scanner;
import sp.sathya;
class SathyProg2
{
  Public static void main(String [] args)
  {
    scanner s=new scanner(system.in);
    sop("Enter First value");
    String s1 = s.nextLine();
    sop("Enter Second value");
    String s2=s.nextLine();
    Sathya so=new Sathya();
    try
    {
      so.Sathya(s1,s2); //user defined common instance
                          exception method.
    }catch(ArithmeticException ae)
    {
      System.err.println("don't enter zero for denominator");
    }
    catch(NumberFormatException nfe)
    {
      system.err.println("don't enter alphanumeric values for division");
    }
    finally
    {
      sop(" i am from finally block");
    }
}
```

re-throwing / deep throwing the exception:-

Def:- First method is throwing the exception to 2nd method. 2nd method is throwing 1st method related exception to 3rd method. w.r.t 2nd method 1st method related exception is called re-throwing the exception.

Ex:- In sathyaprog2.java, sathya.java (see previous pages), the ParseInt (-) (1st method) of integer class is throwing NumberFormatException to division (-,-) (2nd method). division (-,-) of sathya class is again throwing NumberFormatException of ParseInt (-) to main() (3rd method). with respect division-, NumberFormatException of ParseInt(-,-) is called re-throwing the exception

NOTE:-

→ out & err are the two objects of Printstream class and they are created as static data member in another predefined class called system. The Purpose of out Object is to display soft outputs of java program. whereas the purpose of err object is that to display abnormal terminated messages which are occuring as a part of exceptions.

→ in is an object of InputStream created as a static data member in another pre defined class called system. The purpose of in object is that to provide an environment to read the data from keyword

Finding the details about unknown exceptions :-

When we write a java program there is a possibility of many no. of exceptions & java programmer may not be knowing details about all the exceptions & at that point of time we are unable to generate user friendly error messages.

In java programming to find the details about unknown exceptions, we have 3 approaches they are

1. By using an object of java.lang.Throwable/java.lang.Exception
2. By using PrintStackTrace ( )
3. By using getMessage ( )

By using an object of java.lang.Throwable/java.lang.Exception :-

By using the concept of dynamic binding, an object of java.lang.Throwable/java.lang.Exception we can get two details related to unknown exceptions

(a) Name of the unknown exception

(b) Nature of the message.

Ex :-

```
try
{
  =
  int c = 10/0;
  =
} catch (Throwable t) = new ArithmeticException ("/ by zero")
        Exception
{
  System.err.println(t); // Java.lang.ArithmeticException : / by zero
```

Reffered by JVM

Nature of

By using PrintStackTrace() :-

It is one of the pre-defined method present in java.lang.Throwable and further inherited into java.lang.Exception & java.lang.Error.

By using PrintStackTrace() we can get three details

a. Name of the unknown exception
b. Nature of the message
c. Line number where the exception occurs

The prototype of the printStackTrace() is shown below

Java.lang.Throwable

Public void printStackTrace();

Since the return type of printStackTrace() is void we can't call this method as a part of either sop(e) sep.

```
sep(e.printStackTrace());  // Invalid
e.printStackTrace();       // valid
```

EX:-

```
// X.Java
try
{
1.
2.
3. int e=10/0;
4.
} catch(ArithmeticException e)
 {
 e.printStackTrace();  // Java.lang.ArithmeticException : /by zero at X.Java : 3
}
```

e = new ArithmeticException("/by zero");

Name of the unknown exception — Nature of message — Line number

By using getMessage():-

It is one of the pre-defined method present in Java.lang.Throwable and inherited into Java.lang.Exception and Java.lang.Error.

This method gives only nature of the message but not other details.

The prototype of getMessage() is shown below

Java.lang.Throwable

```
public String getMessage();
```

EX:-

```
try
{
  int x=Integer.ParseInt("10");
}catch(Exception e)
{
  sep(e.getMessage());// For InputString : 10x
}
```

Nature of message

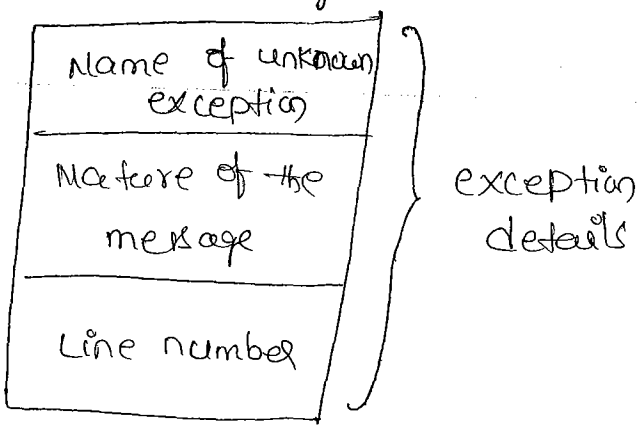
NOTE:-1

In Java programming if any method is returning a value then holding that value in a variable of that return type is optional.

NOTE:-2

If any exception occurs in the Java Program, then exception details will be pushed into the stack memory. ~~Name of exception, nature of msg & line no.~~

Stack memory



20/8

throw

"throw" is a keyword which is used for carrying the created exception from method body & handover to "throws" keyword

(or)

"throw" is a keyword used for hitting/generating/raising the exception which is occured as a part of method body.

The place of using "throw" keyword is always as a part of method body.

Rules for using "throw" keyword :-

Rule :- 1 When "throw" keyword is hitting checked exception then it is mandatory to write "throws" keyword as a part of method heading. Otherwise we get compile time error

Rule :- 2 When "throw" keyword is hitting unchecked exception then it is optional to write "throws" keyword as a part of method heading.

Rule :- 3 Irrespective of type of exception we hit with "throw" keyword then it is highly recomended to write "throws" keyword as a part of method heading.

SYNTAX:-1

```
<clsname> <obj> = new <clsname> (msg);
        throw (obj);
            (or)
        throw(new <clsname> (msg));
```

Here, clsname represents name of exception sub class.

SYNTAX:-2 ("throw" Keyword along with "throws" Keyword)

```
Rettype methodName (List of formal Params if any) throws <clsname-1> ----
                                                        <clsname-n>
{
  if (Test cond 1)
  {
    <clsname-1> <obj-1> = new <clsname-1> (msg);
          throw (obj-1);
  }
    :
    :
  if (Test cond-n)
  {
    throw(new <clsname-n> (msg));
  }
}
```

In the above syntax <cls name-1> ---- <clsname-n> represents name of the exception subclasses.

* Write a java program which will recreate an object of ArithmeticException & generate out own message.

```
// SathyaTech.java
package st;
Public class SathyaTech
 {
  public void div(int a, int b) throws ArithmeticException
   {
    if (b==0)
     {
     ArithmeticException ae = new ArithmeticException("don't
        enter zero for denominator ");
     throw ae;
     }
     else
      {
       int c=a/b;
       SOP("div in sathyatech="+c);
      } // else
   } // div
 } // Sathya Tech
```

Compile above program

javac -d . SathyaTech.java

st
└→ SathyaTech.class

In st.SathyaTech class the div() is one of the user defined common exception instance method. & it must be placed with in try & catch block.

*Write a java main program which makes use of div() of St.SathyaTech class (257)

```
//ExceDemo.java
import java.util.scanner;
import St.SathyaTech;
class ExceDemo
 {
   Public static void main( String [] args)
    {
      Scanner S=new scanner(system.in);
       S.OP("Enter First value");
       int x= S.nextInt();
       Sop("Enter second value");
       int y= S.nextInt();
       SathyaTech St=new SathyaTech();
       try
        {
          St.div(x,y);
        } catch(Exception e)
        {
          system.err.println(e); //java.lang.ArithmeticException : don't
                                    enter zero for denominator
        }
        finally
         {
          System.err.println(" finally block");
         }
    } // main
 } // SathyaTech
```

Differences btw throw & throws

throw	throws
→ "throw" is a keyword used for hitting or generating the exception which is occuring as a part of method body.	→ "throws" is a keyword which is used for giving an indication to the specific method to place the common exception method within the try & catch block for generating user friendly error messages.
→ The place of using "throw" keyword is always as a part of method body	→ The place of using "throws" keyword is always as a part of method heading
→ When we use "throw" keyword as a part of method body for hitting any type of exception it is highly recomended to write "throws" keyword as a part of method heading	→ When we write "throws" keyword as a part of method heading it is optional to write "throw" keyword as a part of method body (In the case of re-throwing the exceptions)

Development of user/programmer defined exceptions:-

user defined exceptions developed by Java programmers available as a part of Java project & whose role is to deal with common problems.

Some of the common problems are

1. Attempting to enter invalid salary to an employee.
2. Attempting to enter wrong PIN number in ATM based apps.
3. Attempting to enter invalid user name & password etc

What are all the guidelines are followed by SUN MS Developers at the time of developing pre-defined exception classes, same guidelines will be followed by Java programmers for developing user defined exception classes

Steps/ guidelines for developing user defined exceptions:- (259)

1. Choose an appropriate package for placing user defined exception sub classes for common access & ensure that the package stmt must be first executable stmt.
2. Choose an appropriate user defined class & ensure that whose modifier must be public.
3. Whatever the class is selected in step(2), it must extends either java.lang.Exception or java.lang.RuntimeException for inheriting the capabilities of exception handling to terminate the program execution abnormally when the appl user enters invalid input. Hence, Step(2) class is called user defined exception sub class.
4. Each & every user defined exception sub class must contain a Parameterized constructor by taking String as a parameter. Here String parameter represents nature of the message.
5. Each & every user defined exception sub class parameterized constructor must call the parameterized constructor of either java.lang.Exception or java.lang.RuntimeException by making use of Super(msg)

   Here msg is the string parameter describes nature of message (The reason of writing msg is that if any specific exception occurs in the java program & it not handled by the java program then specific exceptions will be posted to either java.lang.Exception or java.lang.RuntimeException)
6. Whatever the exception sub class we are placing in the package, it must be given as a file name with an extension .java.

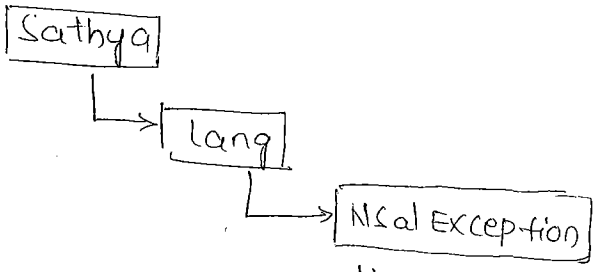
8/18

* Develop a user defined sub class for dealing with -ve salary of an employee

```
//NSalException.java ----- (6)
Package Sathya.lang; //------(1)
Public class NSalException //(2) extends Exception //(3)
{
    Public NSalException(String s) //----(4)
    {
        Super(s); //---- (5)
    }
} //NSalException - development of user defined exception sub class
```

compile the above program

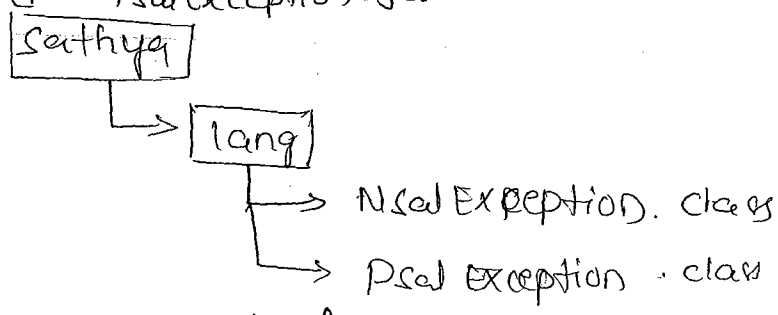
```
javac -d . NSalException.java
```



* Develop a user-defined exception sub class to deal with the salary of an employee

```
//PSalException.java
Package Sathya.lang;
Public class PSalException extends RuntimeException
{
    Public PSalException(String s)
    {
        Super(s);
    }
} //PSalException
```

Compile the above program
javac -d . PsalException.java



No. of Phases required for developement of user defined exceptions:

Phase-1 :- Development of user-defined exception sub classes

Phase-2 :- Development of user-defined common/general classes

Phase-3 :- Development of user-defined specific classes

Development of user-defined exception sub class :- (P-1)

To develop phase-1 we need to follow 6 guiddines (See previous pages)

Ex:- NsalException.java, PsalException.java } are comes under user defined exception sub classes (see previous pages)

Development of user-defined common/general classes :- (P-2)

The purpose of P-2 classes development is that to deal with common requirements of all the programmers. By making use of pre-defined exception sub classes & user-defined exception sub classes for generating appropriate exceptions.

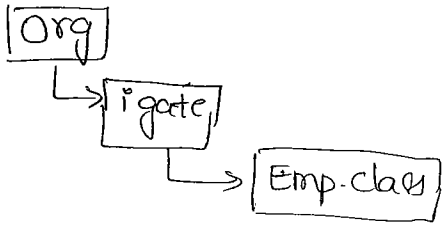
* Write a java program which will check the salary of any employee & generate appropriate exceptions.

```
// Emp.java
package org.igate;
import Sathya.lang.NsalException;
import Sathya.lang.PsalException;
```

```
Public class Emp
{
 Public void checksal(String s) throws NsalException,
  PsalException, NumberFormatException
  {
    int sal=Integer.ParseInt(s);
     if(sal<=0)
     {
      NsalException no=new NsalException("Invalid sal");
      throw no;
     }
     else
     {
      PsalException po=new PsalException("valid sal");
      throw po;
     } // else
  } // checksal()
} // Emp--- dev of user defined common class.
```

compile the above program

javac -d . Emp.java



In org.igate.Emp class, checksal is one of the user defined common exception Instance method & it must be placed always with in try & catch block.

Development of user defined specific class (P-3)

This phase classes deals with specific requirements of individual programmers. This phase classes handles common classes (P-2) & handles the exceptions.

*Write a java program which will check the salary of sathya employee.

```
// SathyaEmp.java
import org.igate.Emp;
import Sathya.lang.NsalException;
import Sathya.lang.PsalException;
import static java.lang.System.*;
import java.util.Scanner;
class SathyaEmp
{
  public static void main(String [] args)
  {
    Scanner s = new Scanner(in);
    out.println("Enter UR Sal");
    String sal = s.nextLine();
    Emp eo = new Emp();
    try
    {
      eo.checksal (sal);
    } catch(NsalException n)
    {
      err.println("don't enter -ve/zero sal for emp");
    }
```

```
catch (PsalException)
 {
   err.println("Good sal, try for best");
 }
Catch (NumberFormatException)
 {
   err.println("don't enter Alphanumeric sal for emp");
 }
finally                    catch (Exception e)
 {                          {
   sop("i am from finally");  err.println(e)
 }                          }
} //main()
} // SathyaEmp --- dev of User-defined specific class
```

NOTE:- When we write multiple catch blocks in the java program then it is mandatory to the java programmer to write specific catch blocks first & later general catch block (java.lang.Exception) otherwise we get compile time error.

2/18

* Write a user-defined exception based app// for user authentication process (verify the user name & password)

```
//InvalidLoginException.java
Package ivp.lang;
Public class InvalidLoginException extends Exception
 {
   Public InvalidLoginException (String s)
    {
      Super(s);
    }
```

```
// ValidLoginException.java
Package vp.lang;
Public class ValidLoginException extends RuntimeException
 {
   Public ValidLoginException (String s)
    {
      super(s);
    }
 }
```

Cmp:- javac -d . ValidLoginException.java

```
// Login.java
Package org.wipro;
import ~~InvalidLoginException~~ ivp.lang.InvalidLoginException;
import ~~ValidLoginException~~ vp.lang.ValidLoginException;
public class Login
 {
   public void Auth(String un, String Pwd) throws
   InvalidLoginException, ValidLoginException
    {
      String s=
     if((un.equalsIgnoreCase(Sathya)) && (Pwd.equalIgnoreCase
          (Java)))
      {
        ~~Sop~~ ValidLoginException VO=new ValidLoginException(valid
                                                un, pwd)
        throw VO;
      }
     else
     {
       ~~Sop~~
          ..... InvalidLoginException (Invalid un, pwd);
```

```
// Sathyauser.java
import org.wipro.Login;
import ~~util~~ java.util.Scanner;
import ivp.lang.InvalidLoginException;
import vp.lang.ValidLoginException;
class Sathyauser
{
  Public static Void main(String [] args)
  {
    Scanner s = new Scanner(System.in);
    Sop(" Enter user name:");
    String un = s.nextLine();
    Sop("Enter Password:");
    String pwd = s.nextline();
    Login lo = new Login();
    try
    {
      lo.Auth(un, pwd);
    } catch(ValidLoginException)
    {
      System.err.println(" un&pwd is correct, Proceed");
    }
    catch(InvalidLoginException)
    {
      System.err.println(" un&pwd is wrong, try again");
    }
    Catch (Exception e)
    {
      System.err.ptrinthn(e);
    }
    finally { Sop(" am from finally") }
```

Develop an exception based appln in such a way that names of the people / Product names should not contain digits. (267)

Hint:- Sathya4Tech — invalid - becoz it contain digits

SathyTech - valid - becoz no digits

P-1 :- NameFormatException

P-2 :- org.tcs.ValidateName

Public void CheckName(~~Sathya~~ String s)

```
// NameFormatException.java
package np;
Public class NameFormatException extends Exception
{
  Public NameFormatException(String s)
  {
    super(s);
  }
} // NameFormatException — UDESC
```

cmd:- javac -d NameFormatException.java

```
// ValidateName.java
Package org.tcs;
import np.NameFormatException;
Public class ValidateName
{
  Public void CheckName(String s) throws NameFormatException
  {
```

```
char ch[] = s.toCharArray();
for(int i=0; i< ch.length; i++)
 {
   if(Character.isDigit(ch[i]))
   {
     found = true;
     break;
   } //if
 } // for
 if (found)
 {
  throw(new NameFormatException("Invalid Name:"));
 }
 else
 {
  sop("UR Name is Valid");
 }
} // CheckName()
} // ValidateName - dev of UDCC
```

cmd: javac -d . ValidateName.java

```
// CheckNameDemo.java
import java.util.Scanner;
import np.NameFormatException;
import org.tcs.ValidateName;
class CheckNameDemo
{
  Scanner s = new Scanner(System.in);
  sop(" Enter UR Name: ");
  [illegible] s.nextLine();
```

```
ValidateName vo = new ValidateName();
try
{
   vo.checkName(name);
} catch (NameFormatException
{
  System.err.println("OR Name should not contain digits");
}
catch (Exception e)
{
   System.err.println(e);
}
finally
{
  sop("i am from finally block");
}
} //main()
} // CheckNameDemo
```

* Write a exception based class which will works like Sathya ArithmeticException.

* Develop a exception based class which works like Sathya NumberFormatException

24/8/15

Multi Threading

The purpose/aim of multi threading is that to get/provide Concurrent execution [simultaneously/parallelly executing multiple operations].

Defn of thread:- A flow of control is known as 'thread.'

→ The purpose of creating a thread is that "to execute logic of the java program which is written in the form of user defined methods concurrently".

→ If a java program contains multiple threads then it is called "Multi threaded".

→ The languages like C, C++ are called single threaded modeling languages becoz their execution environment contains single flow of control provides sequential execution, takes more execution time & doesn't contains any library for development of multi threading appls.

→ A language like Java is treated as multi threaded modeling language becoz their whose execution environment provides multiple threads, provides concurrent execution, take less execution time & contains an effective API for development of multi threaded applications.

→ Whenever we write a java program we get 2 types of threads

(a) Foreground/ child thread

(b) Background/ Parent thread

A Foreground thread always executes logic of the java program which is written in the form of methods.

A Background thread always monitors the execution status of

By default there exist single FGT & single BGT. Programmatically, we can have multiple FGT's & recomended to have one BGT.

→ The real time implementation of multi threading is that to develop real world server softwares (compeect, Tomcat-Apache, weblogic-BEA system, websphere-IBM etc) In otherwords most of the server s/w's developed by Server s/w vendors in java language with multi threading concept.

→ To develop multi threading appls in java we use the following API

a. Java.lang.thread (class)

b. Java.lang.Runnable (Interface)

Hence, multi threading concept of java is one of the specilized form of multi tasking of OS.

Q:- How do you justify "Each & every java program is multi threaded"?

Ans:- When we execute the java program, logic of the java program is executed by one of the thread known as "Foreground thread".

To monitor the execution of status of FGT internally one more thread is created & it is known as "Background thread".

Hence, java program contains 2 threads by default & it is multi threaded.

Q:- What are the differences btw program & process.

Ans:-

Program	Process
→ Set of optimized instructions is known as program.	→ Process is under execution of program.
→ Programs resides in secondary memory for a long time until we delete	→ Process resides in main memory for a limited span of time.
→ A program is considered as a static object	→ A process is considered as a dynamic object

2018

Process Based Vs Thread Based applications (or) Non-Multi Tasking Vs Multi Tasking applications :-

→ In real world we develop 2 types of application. They are

1. Process based application (PBA)
2. Thread based application (TBA)

PBA	TBA
→ PBA provides/contains single flow of control	→ TBA contains/provides multiple flow of controls.
→ Single threaded modeling languages related appls comes under PBA Ex:- C, C++, COBAL, PASCAL etc	→ Multi threaded modeling languages comes under TBA. Ex:- JAVA & .Net.
→ In PBA, context switch is more,	→ In TBA, context switch is less.
→ In PBA, for each & every sub-program there exist a seperate address space.	→ In TBA, irrespective of no. of sub programs, there exist single address space.
→ PBA takes more execution time.	→ TBA takes less execution time.
→ PBA are treated as heavy weight components	→ TBA are treated as light weight components.
→ PBA provides only sequential execution but not concurrent execution	→ TBA provides both sequential & concurrent executions.

Definitions:-

Address space:- It is an amount of temporary memory space created by OS on the stack memory for the temporary execution of the method.

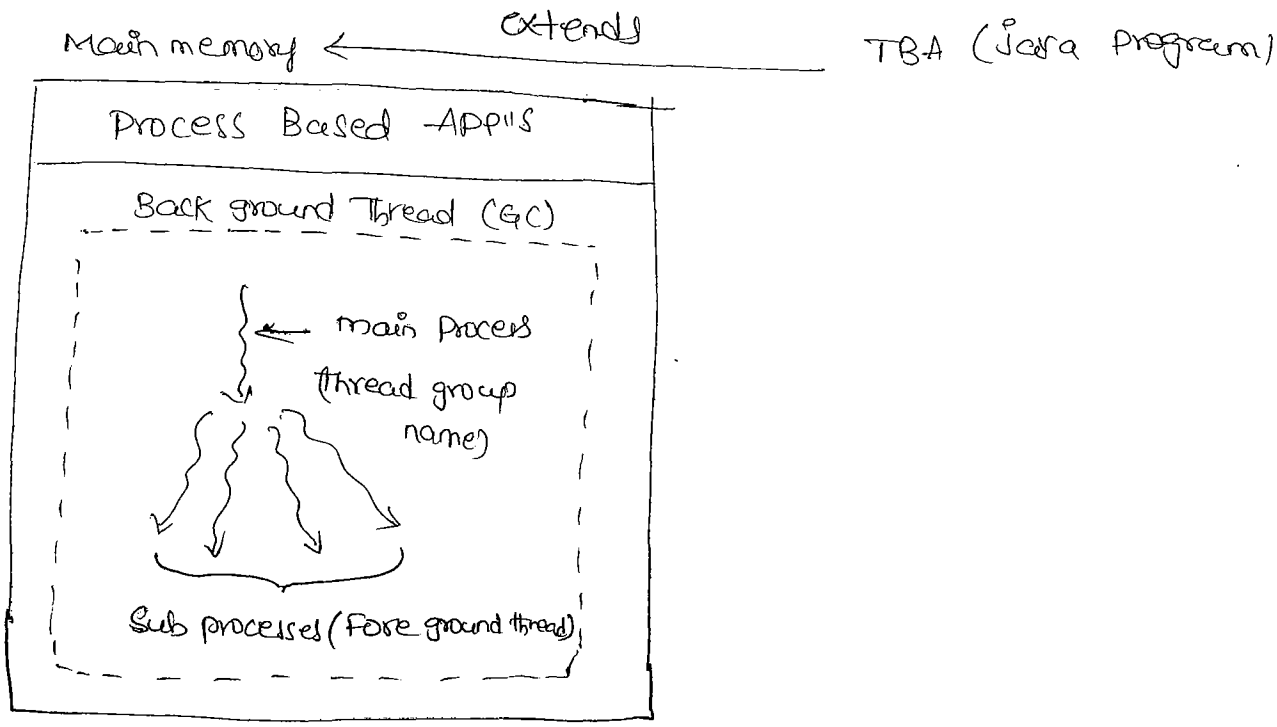
Context switch:- The mechanism of switching the control of CPU from one address space to another address space is called context switch.

→ For the best appl development context switch must be less.

components takes less execution time.

Sharp NOTE1:-

Whenever we execute any TBA then it will be considered as PBA & in which one main process will be created. The main process internal creates sub processes. In Java programing point of view the main process is called Thread group name & the sub processes are called Foreground Threads & they meant for executing the methods which contains logic of Java program concurrently. The following diagram gives view about the threads layers.



Whereas a PBA will be considered as PBA only but not treated as TBA becoz there exist only one process which is considered as one thread (green thread) only & there is no possibility of creating multiple threads.

States of a thread (or) life cycle of a thread :- (274)

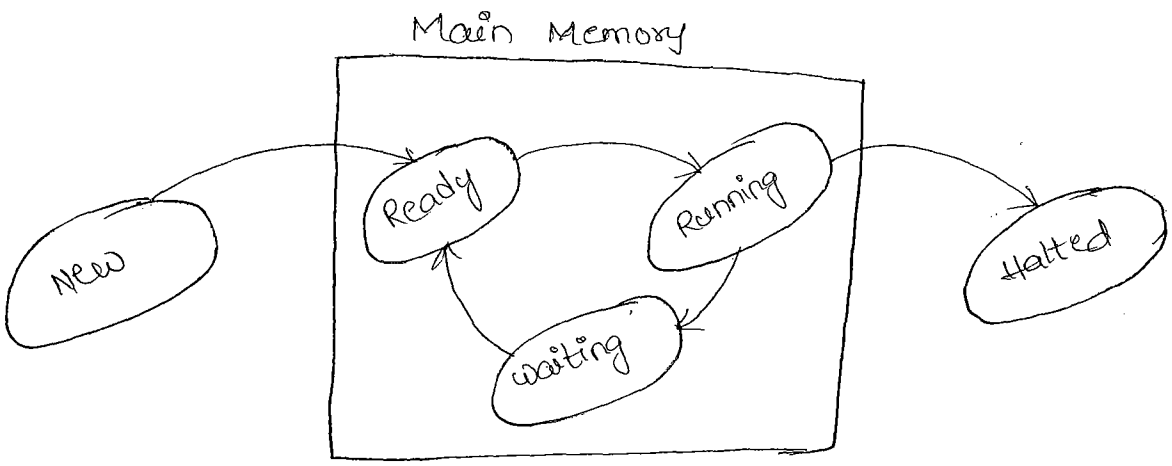
The different breaking points / stages occuring in the execution of logic of the java program by the thread are known as States/ life cycle of a thread.

States of thread are classified into 5 types. They are

1. New State
2. Ready State
3. Running State
4. Waiting State
5. Halted State

26/8

State chart diagram:-



New State:-

In this state, thread is created & about to enter into ready state.

Ready State:-

In this state, thread is entered into main memory, memory space is created for the thread & thread is waiting 1st time for the CPU.

Running State:-

In this state, thread is under the control of CPU. In otherwords,

is written in the form of user defined program then the state of the thread is Running state.

Def" of CPU burst time :-

It is an amount of time taken by the thread from the device CPU for complete execution of the logic of the java Program is called CPU burst time.

CPU burst time is decided by OS. but not by Programer

Halted State :-

In this state, thread completed its execution

Waiting State :-

A thread is said to be in waiting state if & only if it satisfies any of the following factors.

a. Thread is coming to waiting state for the remaing CPU burst time

b. Suspending currently executing thread.

c. Making the currently executing thread to sleep for the Period of time interms of msec. [1sec = 1000msec]

d. Making the currently executing thread to wait for a period of time interms of msec

e. Making the currently executing thread to wait without time

f. Joining the threads.

Hence, as long as thread is in New & Halted States (Out memory States) whose execution status is False. whereas as long as thread is in Ready, Running & Waiting states (In-memory states) whose execution Status is True.

Q:- How do you justify threads of java executes concurrently?

Ans: Let us assume if threads of java executes with the difference of hours, minutes, seconds then it will be considered as slowest execution/ sequential execution. Becoz, we can differentiate difference of hours, minutes & seconds. If threads of java executes with the difference of milliseconds of time then it will be considered as fastest execution. ~~becoz, we can~~ concurrent execution

Hence in the java execution environment threads are executing concurrently with the difference of msec of time by following Round Robbin Algorithm

NOTE:- Difference of msec of time is not able to be differentiate by human being calculations & we feel that all at once execution

27/8

Creating a thread

WKT the purpose of creating a thread is to execute logic of the java program which is ~~re~~written in the form of a method

In Java programming we've 2 approaches to create a thread. They are

1. By using Java.lang.Thread class
2. By using Java.lang.Runnable Interface

By using Java.lang.Thread class:-

By using Java.Lang.Thread class we create its object in 3 ways. They are

a. Directly with new operator:-

Thread t1 = new Thread();

b. By using Factory method :-

Thread t1 = Thread.CurrentThread()

c. By using an object of sub class of Java.lang.Thread :-

```
class Th1 extends Thread
{
  ≡
  ≡
}
Th1 t1 = new Th1();
```

Here t1 is an object of Th1 & Th1 is a sub class of Java.lang.Thread. Indirectly, t1 is an object of Java.lang.Thread class

*Imp Profile of Java.lang.Thread :-

Data Members :-

Public static final int MAX_PRIORITY (=10)

Public Static final int NORM_PRIORITY (=5)

Public static final int MIN_PRIORITY (=1)

The above data members are called thread priority modifiers. The Purpose of thread priority modifiers is that makes the CPU to understand which threads to execute first, second & last.

The default priority of the thread is NORM_PRIORITY (=5)

Constructors :-

1. Thread () :- (Default Constructor)

This constructor is used for creating an object of thread class without giving user friendly name to the thread.

Ex:- Thread t1 = new Thread();

If we creat 'N' no. of Thread class objects with this default constructor then whose default names starts with Thread-0, Thread-1, ..., Thread-N-1 & they are not recomended to give & always recomended to give user friendly names to the thread.

2. Thread (String) :- (Parameterized constructor)

This constructor is used for creating an object of thread by giving user friendly name to the thread.

Ex:- Thread t1 = new Thread(Sathya);

ref variable/obj name user friendly name

3. Thread (Runnable) :- (Object parameterized constructor)

This constructor is used for converting Runnable intf object into Thread class object for making use of all the methods of Thread class especially start() to enter into the overridden run() of Sub class of Runable intf

Ex:- Th1 t11 = new Th1();

(or)

Runnable t11 = new Thr();

//t1.start(); // invalid

Thread t1 = new thread(t11); // Thread-0

reference

t1.start(); //valid

4. Thread (Runnable, String) :-

The Purpose of this constructor is similar to the above constructor & with this constructor additionally we are giving user friendly name to the thread.

```
Ex:- Runnable r = new Th1();
     // r.setName("Sathya"); //Invalid
     Thread t1 = new thread (r,"Sathya");
     Sop(t1.getName()); // Valid - Sathya
```

Instance Methods :-

1. Public final void setName(String)
2. Public final String getName()

The above methods are used for setting the user friendly name to the thread & getting the user friendly name of the thread.

```
Ex:- Thread t1 = new Thread();
     String tname = t1.getName();
     Sop(tname); //Thread-0
     t1.setName("Sathya");
     tname = t1.getName();
     Sop(tname); // Sathya
```

Q:- Define mutators & inspectors ?

Ans:- A mutator is a type of method which will change the values of the object & whose generalized notation is setXxxxx(—)

Inspector is also one type of method which will obtain the value from the object & they starts with getXxxxx()

3. Public final void setPriority (int)

4. Public final int getPriority ( )

The above methods are user for setting the priority to the thread & getting the priority of the thread.

```
Thread t1 = new Thread(Sathya);
int pv t1.getPriority();
    Sop(pv); // 5 => NORM_PRIORITY
        t1.setPriority(Thread.MAX_PRIORITY);
            PV = t1.getPriority();
                Sop(PV); // 10 => MAX_PRIORITY
```

5. Public boolean isAlive();

This method is used for checking the execcating states of the ~~method~~ thread. This method returns TRUE provided the thread is in in-memory states (Ready, Running & waiting). & this method returns FALSE when thread is in Out-memory states (new & Halted)

6. public void run()

This method is used for providing the defination of logic of the Java program. In otherwords the logic of the java program written by to be executed by the thread to be written in run(). This method originally contains null body & it is always to be overridden by the programmer ~~by~~ for providing logic of the thread.

7. Public final void start():-

When we call start() upon the thread class object, internally start() will perform the following operations.

1. It moves the thread from New state to Ready state
2. Provides multi-threading services (concurrency, synchronization & inter Thread com.n (ITC))
3. It automatically calls run(). which contains logic of the thread.

Hence, inorder to develop any thread based appl.n we choose a class, it must extends Java.lang.Thread class & we must override run() with the logic of the thread.

Ex:-

```
class Th1 extends thread
{
    public void run() // overridden
    {
        ≡ // logic of a thread
    }
}

Th1 t1 = new Th1(); // New State
boolean state = t1.isAlive();
    Sop("state"); // False
        t1.start(); // Ready state.
    state = t1.isAlive();
    Sop("state"); // True
```

The best approach for development of thread based appls is to create an object of Java.lang sub class of Java.lang.Thread class

8. Public final void suspend():-

This method is used for suspending the currently executing thread. When the thread is suspended it stops its execution, temporary execution details are placed in PCB & transfered to waiting state

Ex:- t1.suspend();

NOTE:-

A PCB is one of the buffer it contains temporary execution details of the thread. PCB created by OS when the thread is entered first time in Ready state

9. Public final void resume():-

This method is used for transfering suspended thread from waiting state to Ready state. When the thread is resumed (restarted) it starts continueing its execution where it left off. By retrieving the temporary execution details from PCB.

Ex: t1.resume();

10. Public final void stop():-

This method is used for terminating/stoping the current thread execution & stopped threads will be entered into Halted state. When we restart the stoped threads they start continueing their execution from the begining but not from the place where they stoped.

11. Public final getThreadGroup getThreadGroup():-

This method is used for obtaining thread group name of foreground thread.

```
Th1 t1 = new Th1();
ThreadGroup tg = t1.getThreadGroup();
    sop(tg); // [name=main, MAXPRIORITY=10]
```

12.* Public final Thread.State getState():-

This method is used for finding name of the thread state. This method returns an object of State which is the inner/nested class of Thread class.

Ex:-

```
Th1 t1 = new Th1(); // New state
boolean eState = t1.isAlive();
    sop("Exec. Status of t1 = "+ eState); // False
Thread.State ts = t1.getState();
    sop("State Name = "+ts); // NEW
```

13.* Public final void join() throws Java.lang.InterruptedException:-

This method is used for joining the threads which are completed after their execution as a single unit. This method strengthen the performance of multithreading appls by joining completed threads as a single unit & handover to garbage collector.

The default environment of multithreading says collects the individual threads after their execution & handover to garbage collector individually. This is one of the time consuming process.

This method throws a predefined exception called "Interrupted Exception".

When InterruptedException occurs:-

It is one of the checked exception, this exception never occurs in stand alone machines, but it rarely occurs in busy environment like client server environment due to mis memory management of server side os.

Situation:-

Let us assume 'n' threads are created, started their execution, n-1 threads are completed their execution & joined any n^{th} thread is still under execution. Due to mis memory management of server side os, the ThreadGroupName is attempting to collect all the threads including n^{th} thread. At this point of time w.r.t n^{th} thread JVM generates a predefined exception called "Java.lang.InterruptedException."

Ex:-

```
try
{
  t1.join();
  t2.join();
  ==
  tn.join();
} catch(InterruptedException ie)
{
  sep("problem in thread execution");
}
```

Static Methods :-

1. Public Static final Thread currentThread();-

It is one of the factory method. The purpose of this method is to find the threads which are by default running in Java environment.

Ex:- Thread t = Thread.currentThread();

Sop(t); // Thread [main, 5, main]

FGT priority TGN

t.setName("Sathya");

Sop(t); // Thread [Sathya, 5, ~~Sathya~~ main]

FGT priority TGN- priority is always 10 [MAX_PRIORITY]

2. Public Static final void sleep(long) throws Java.lang.InterruptedException :-

This method is used for making the currently executing thread to sleep interms of msec's. Once the sleeping time is overed auto-matically the thread will goes to ready & running states.

Once the thread is sleeping, it is in waiting state & no further operations will be performed until sleep time completed. This method throws a predefined exception called Java.lang.InterruptedException.

Let us assume first program threads are sleeping in some location of server side OS. Due to mis memory management of server side OS, second program threads are attempting to sleep in the same location where first program threads are sleeping. At this point of time w.r.t first program thread JVM generates a predefined exception called InterruptedException.

Eg:-
```
try
{
  Thread.sleep(1000);
} catch(InterruptedException ie)
{
  sop(" problem in thread execution");
}
```

* Write a Java program which will display preliminary info about the thread such as threads which are by default running & whose execution status, whose FGT, TGN, Default name of programmer defined FGT, whose execution status, state name, thread priority modifier values, parent of TGN etc

```
//Thdemo1.java
import static java.lang.Thread.*;
class Thdemo1
{
  public static void main(String [] args)
  {
    //default thread info
      Thread t = currentThread();
      sop("default threads =" + t); //Thread[main, 5, main]
      t.setName("Sathya");
      sop("default threads after mdfunction = " + t); //Thread[
                                                        Sathya, 5, main]
      String dfgt = t.getName();
      sop("default FGT = " + dfgt); //Sathya
      Thread.State ts1 = t.getState();
      sop("State name of t = " + ts1); //RUNNABLE
      sop("exe// Status of t = " + t.isAlive()); //True
    // thread group info
      ThreadGroup tg = t.getThreadGroup();
      sop("default tg = " + tg); //Thread[name = main, maxPri = 10]
      ThreadGroup ptg = tg.getparent();
      sop("parent of default tg = " + ptg); //Thread[name = SLS, Pri = 10]
    // prog-def thread info
      Thread t1 = new Thread();
      sop("name of prog-def FGT = " + t1.getName()); // Thread-0
```

```
        SOP("exe status of t1="+ t1.isAlive());  //False
        Thread.State ts = t1.getState();
        SOP("state name of t1="+ts);  //NEW
    // Print thread priority modifier info
        SOP("val of max pri="+ MAX_PRIORITY);  //10
        SOP("val of norm pri="+ NORM_PRIORITY);  //5
        SOP("val of min pri="+ MIN_PRIORITY);  //1
    }
}
```

3/18

Internal flow of threads:-

Whenever we write any Java program it contains multiple threads & as a Java program we must understand the internal flow of the threads.

The following sequence of steps gives the internal flow of threads

1. Java program starts its execution
2. JVM creates TGN[TGN always is resides in main()]
3. TGN creates FGT(s) [FGT(s) resides & execute run()]
4. TGN dispatches(starts) the FGT(s) to execute respective run().
5. FGT(s) execute respective run() & gives result back to TGN either one by one (or) all at once.
6. TGN receives the result from FGT(s) either one by one (or) all at once
7. TGN gives the result to JAVA Programmer/Application user either one by one (or) all at once
8. TGN collects the FGT(s) & hand over to garbage collector
9. JVM collects the TGN & hand over to garbage collector
10. Java program stops its execution.

* Write a Thread based appl which will print 1-10 after each & every second.

```
//Thdemo2.Java
Class Th1 extends Thread
{
  Public void run()
  {
    try
    {
      for(int i=1; i<=10; i++)
      {
        sop("val of i=+i);
        Thread.sleep(1000);
      } //for
    }catch(InterruptedException ie)
    {
      sep("prob in thread exec---");
    }
  } //run
} //Th1 -- user defined class
Class Thdemo2
{
  Public static void main(String [] args)
  {
    Th1 t1 = new Th1(); // new state
    sop("exec states of t1 before start="+t1.isAlive());//false
```

```
t1.start();
sop("exec status of t1 after start="+t1.isAlive());//true
try
{
  Thread.sleep(5000);
} catch(InterruptedException ie)
{
  sep("prob in thread exec...");
}
sop("exec status of t1 during execution--line-35
  = "+t1.isAlive());//true
try
{
  Thread.sleep(5001);
}catch(InterruptedException)
{
  sep("prob in thread exec---");
}
sop("exec status of t1 after execution--- line-43
  = "+t1.isAlive());//false
}

}
```

The some other best solutions for printing the execution status of the FGT after its completion are

1. substitute t1.join() in place of thread.sleep() [use join()]
2. subs use while (t1.isAlive())
   {
   }
   in place of thread.sleep stmt.

1/9/15

By using Java.lang.Runnable :-

→ It is one of the pre-defined functional interface present in Java.lang.* package

→ This interface contains only one method & whose prototype is given below

```
Public abstract void run()
```

→ The run() of Thread class contains null body & run() of Runnable interface is abstract.

→ Industry is highly recomended to override abstract run() of Runnable interface instead of overriding null body run() of Thread class.

→ Java.lang.Runnable interface is an alternative solution for development of multi-threading appls instead of using Java.lang.Thread class

Consider the following invalid code segment which is not possible to develop multi threading appls

```
Class Th1 extends Sathya, Thread // Invalid code
{
  ≡
  ≡
}
```

To overcome the above multiple inheritance problem we rewrite the above code as follows with the help of Runnable interface.

```
Class Th1 extends Sathya implements Runnable // valid code
{
  ≡
  ≡
```

Hence, to develop multithreading applications with Runnable interface, we choose ~~their~~ a user defined class, it must implements Runnable interface & override run()

```
class Th1 implements Runnable
{
  Public void run() //overridden
  {
    ≡ //Logic of a thread.
  }
}
```

* Write a thread based appl^n which will print 1 to 10 no. by using Runnable interface

```
//Thdemo6.java
Class Th1 implements Runnable
{
  Public void run()
  {
   try
    {
     for(int i=1; i<=10; i++)
      {
       sop("val of i="+i);
       Thread.sleep(1000);
      } //for
    } Catch(InterruptedException ie)
    {
     sep(" Prob in thread exec---");
    }
  } //run
```

```
class Thdemo6
{
  Public Static void main(String [] args)
  {
    // Th1 t1=new Th1(); // new State (OR)
      Runnable t11 = new Th1();
      Thread t1 = new Thread(t11, Sathya); //Converting Runnable inst object into thread class
      sop("name of FGT="+ t1.getName()); // Sathya object
       sop("exec Status of t1 before start =
        "+ t1. isAlive()); // false
        t1.start();
       sop("exec Status of t1 after start=
        "+ t1.isAlive()); //True
  }
}
```

* Develop a thread based appl for computing sum & sub of 2 nos concurrently

```
import java.util.Scanner;
Class Sum extends Thread
{
    int a,b,c;
    void set(int a, int b)
    {
      this.a=a;
      this.b=b;
    }
    Public void run()
    {
      c=a+b;
```

```
    } sop("sum="+c);
    } //sum

Class Sub implements Runnable
{
    int a,b,c;
    void set(int a, int b)
    {
        this.a=a;
        this.b=b;
    }
    Public void run()
    {
        c=a-b;
        sop("sub="+c);
    }
} //Sub

Class Thdemo7
{
    Public Static void main(String [] args)
    { //accept the values from KBD
        Scanner s= new Scanner(System.in);
        sop("Enter first value");
        int x1=Integer.parseInt(s.nextLine());
        sop("Enter Second value");
        int x2=Integer.parsInt(s.nextLine());
    // create the threads
        Sum s1= new Sum();
        Sub s1= new Sub();
```

```
// Set the values to threads for computing sum & sub operations
    S1.set(x1, x2);
    S2.set(x1, x2);
// dispatch the threads
    S1.start();
    // S2.start(); Invalid
    Thread t22 = new Thread(S2); // converting obj of sub class
        t22.start();            of Runnable into obj of Thread class
// join the FGTs
    try
    {
        S1.join();
        t22.join();
    } Catch (InterruptedException ie)
    {
        Sep("prob in thread exec...");
    }
// Check the execution status after joined
    Sop("exec status of S1 = "+S1.isAlive()+" and
        State Name = "+S1.getState()); // TERMINATED (halted)
    Sop("exec status of t22 = "+t22.isAlive() + " and
        State Name = "+t22.getState()); // TERMINATED (halted)
    }
}
```

∴ NOTE :- In general if we want to execute multiple independent operations concurrently then we must choose multiple independent class & they must implement Java.lang.Runnable & we (

NOTE:

Java.lang.Thread is internally implementing Java.lang.Runnable Intf & Provides null body for run()

219 * Write a java program which will implement banner animation OR WATP for scrolling the given message

```
// ani.java
import java.awt.*;
import java.applet.*;
/* <applet code="ani" height=300 width=300 >
 </applet> */
Public class ani extends Applet implements Runnable
 {
   String msg="JESUS SAVES US";
   Public void init()
   {
     setBackground(color.PINK);
     setForeground(color.GREEN);
   }
   Public void start()
   {
    Thread t=new Thread(this);// 'this' refers to current
    class obj. which implements 'Runnable' interface & converts
    into Thread class obj
    t.start();
   }
   Public void run()
   {
     try
     {
```

```
{
    Char ch=msg.charAt(0);
     msg = msg.substring(1,msg.length());
     msg = msg+ch;
     Thread.sleep(1000);
     repaint();
  } // while
} catch(InterruptedException ie)
  {
   Sep("prob in thread exec...");
  }
} //run()
 Public void paint(Graphics g)
  {
    Font f=new Font("arial", Font.BOLD, 60);
    g.setFont(f);
    g.drawString(msg, 100,100);
  }
} // ani-class
```

* Write a Java Thread based appln which will automatically increment the numbers

```
//AutoInc3.java
import java.awt.*;
import java.applet.*;
/*<applet code="AutoInc3" height=300 width 300> </applet>*/
Public Class AutoInc3 extends Applet
  {
    int i=0;
```

```
{
  SetBackground(color.Blue);
  SetForeground(color.white);
}
Public void Start()
{
  Sathya s=new Sathya(this); // capturing the current class
  obj & entering into Sathya class
  Thread t=new thread(s); // 's' refers to obj of InDp class
  which implements 'Runnable' intf & converts into thread class obj
  t.Start();
}
Public void paint(Graphics g)
{
  Font f=new Font("arial", Font.BOLD, 60);
  g.SetFont(f)
  g.drawString(String.ValueOf(i), 100, 100);
}
} //AutoInc3_outer class
class Sathya implements Runnable // INDp class
{
  AutoInc3 a;
  Sathya(AutoInc3 a) // object parameterized constructor
  {
    this.a=a;
  }
  Public void run()
  {
    try
    {
```

```
while (true)
  {
    a.i++;
    a.repaint();
    Thread.sleep(1000);
  } //while
} Catch(InterruptedException ie)
{
  sop("prob in thread exec");
}
} //run()
} //Seathga --- INDP Class
```

Synchronization

1/9/15

Synchronization is the concept of OS & it is available in Java as a predefined facility with the aim of building thread safety applications

If multiple threads executes (or) performs the operations on the same resource then by default we get inconsistant results/ dead lock results which is a not a recomended process. To overcome this problem we apply the concept of synchronization.

The advantage of synchronization concept is that preventing the dead lock results & generating consistant/thread safety results

Defn of synchronization:-

The process of allowing one thread among multiple threads into the shareable area for perform write & read operations for getting consistant result by eliminating inconsistent results is

→ Synchronization concept is always recomended to apply on shareable operations & not recomended to be apply on independent operations.

Need of Synchronization:-

Let us assume there is a sharable variable balance & whose initial value is zero. Let us assume there exist two threads decided to deposit Rs. 10 & 20 respectively in the balance of account concurrently. When both the threads started their execution for depositing & after completion of their execution the result of the balance variable is either 10/- or 20/- but not 30/- which is one of the inconsistant result. To overcome this inconsistant result we must apply the concept of synchronization

Synchronization applied (on the above problem)

Let as assume both the threads started their execution at the same time (t_1 starts first & later t_2 with the difference of milli seconds of time) then JVM gives value of balance (Rs. 0) to thread t_1 & balance variable locked by JVM & thread t_2 mayed wait until thread t_1 completes its execution. Thread t_1 completed its execution, JVM unlock the balance variable, JVM gives value of balance (Rs-10) to thread t_2 & once again balance variable will be locked. Thread t_2 also completed its execution & balance variable will be unlocked. The final result of the balance variable is 30/- which is one of the consistant result.

Hence, during the synchronization process JVM continues the process of locking & unlocking until all the threads completed their execution.

Thread synchronization Techniques :-

We've 2 types of synchronization techniques

1. Synchronized methods
   a. synchronized instance methods
   b. synchronized static methods
2. Synchronized blocks

Synchronized Methods :-

Synchronized methods are classified into 2 types. They are

1. Synchronized instance method
2. Synchronized static method

1. Synchronized instance method :-

If an ordinary instance method accessed & executed by multiple threads concurrently then by default we get inconsistent result. To overcome this inconsistent results of ordinary instance methods, whose defns must be made as synchronized by using "synchronized" keyword.

SYNTAX :-

```
synchronized Rettype methodname(List of formal Params if any)
{
    Block of Stmt(s) -- Business logic - Thread Safety results
}
```

As long as the thread is executing synchronized instance method JVM will lock object of corresponding class.

Ex:- class Account

```
{
  Private int balance=0;
  Synchronized void deposit(int amt)// writable method
  {
      bal=bal+amt;
      sop("curr.bal=" +bal);
  }
}
```

As long as thread executing the above synchronized instance deposit() then JVM will lock an object of account class

419

2 Synchronized static method:-

When an ordinary static method accessed by multiple threads concurrently then we get inconsistant result.

To overcome these inconsistant results the defn of ordinary static method must be made as synchronized by using synchronized keyword.

Syntax:-

Synchronized ~~public~~ static Rettype methodname (List of formal Params if any)

```
{
    Block of stmts - Business logic - Thread safety result
}
```

Ex:- class Account

```
{
  private int bal=0;
  synchronized static void deposit(int amt)
  {
      bal=bal+amt;
      sop("curr bal=" +bal);
  }
```

As long as the thread is executing the above synchronized static deposit() JVM will lock the account class.

In general as long as the thread is executing synchronized static method JVM will lock corresponding class

During the synchronization process JVM is locking either an object (or) class depends on type of synchronized method executed by JVM.

Synchronized Blocks :-

Synchronized block is an alternative mechanism for obtaining thread safety results instead of using synchronized methods.

→ Synchronized blocks to be written inside the non synchronized instance method

→ Syll synchronized blocks always makes the JVM to lock an object of corresponding class

Syntax:-

```
synchronized( obj of current class)
{
  Block of stmts() - Business logic - Thread safety results
}
```

Need of Synchronized block :-

Whenever we inherite non-synchronized instance methods either from base class/ int-f to derived class & if the inherited non-synchronized instance method accessed by multiple threads ... To get consistent result as

a derived class programmer attempting to write "synchronized" keyword before inherited non-synchronized instance method which is not a recomended process & to overcome this problem we use synchronized blocks.

Ex:-

```
interface Account          //abstract class Account
{                          (or)  {
  void deposit(int);       //      void deposit(int amt)
}                                  {}
```

```
Class savingaccount implements / extends Account
{
  int bal=0;
  public void deposit (int amt)
  {
    synchronized (this)
    {
      bal= bal+ amt;
      Sop("curr bal =" + bal);
    }
  }
}
```

→ synchronized block

* Write a java program which illustrate the concept of synch

Problem Stmt:- Let us assume there exist an account with the balance & whose initial value is zero. On Account perform the operations like depositing & balance enquiry.

Let us assume there exist n-no. of customers who decided to deposit rupees 10/- by each concurrently in the balance of Account. Ensure that during the concurrency we must get consistant result in the balance of account.

5/9/15

30(4)

```
//sync.java
class Account
{
  private int bal=0;
  synchronized void deposit(int amt)
  {
    bal=bal+amt;
    sop("curr bal="+bal);
  }
  int getBal()
  {
    return bal;
  }
} //Account
class cust extends Thread
{
  Account ac;
  cust(Account ac)
  {
    this.ac=ac;
  }
  public void run()
  {
    ac.deposit();
  }
} //cust
```

```
Class sync
{
  Public static void main( string [] args)
  {
  // create the single object of Account
      Account ac=new Account();
  // create the array of obj of cust(threads)
      cust cu[]=new cust[5];
  // give a single obj of Account to cust class objects
    for (int i=0; i<cu.length; i++)
      {
        cu[i] = new cust(ac);
      }
  // dispatch the array of cust object to deposit Rs. 10/- by
    each in the bal of Account
    sop(" ------------ ");
      for(int i=0; i<5; i++)
      {
          cu[i].start();
  //join the array of cust objects
    try
    {
      for(int i=0; i<cu.length; i++)
      {
          cu[i].join();
      }
    } catch (InterruptedException ie)
    {
      sep("prob in thread exec...");
    }
```

```
//print total bal.
    sop("----------");
    sop("\n\n TOTAL BAL = " + ac.getBal());
    sop("----------");
  } //main()
} // sync.
```

NOTE:1 :- When we execute synchronization based appls (sync.java) on mono/single user OS's (DOS, WINDOWS 3.1 & WINDOWS 95) then it is mandatory to the java programmer to write Synchronized keyword. Otherwise we get inconsistent result beoz eventhough there exist synchronization concept in OS, which is not implemented in mono OS's. If we used synchronized kwd explicitely as a part of our java program then we are using java based synchronization beoz & it gives consistent result during the order of updation & in final result.

NOTE:2 When we execute synchronization based appls on multi threaded / multi user OS's (WIN98, WINXP, UNIX, LINEX etc) then writing synchronized keyword is optional. beoz synchronization concept of OS implemented in multi threaded OS's. If we don't use synchronized kwd then the multi-threaded OS's will provide OS based synchronization. Due to this we get consistent final result but order of updation is inconsistent which is not a recomen-ded process.

NOTE:3 Irrespective of type of OS being used, industry is highly recomended to use java based synchronization for getting consistent result during order of updation & final result.

Inter thread communication (ITC)

→ ITC concept of java is the specialized form of inter process com.n of OS.

→ ITC based appls are fastest appls compared to IPC appls becoz ITC appls are light weight components (thred based technology) whereas IPC appls are heavy weight components (process based technology)

→ The real time implementations of ITC concept are

(a) Development of mobile based application

(b) Development of real world server s/w

(c) Development of universal protocols

Def^n of ITC :-

The process of exchanging the data/information btw multiple threads with the consistant comm.n is known as Inter thread communication.

(OR)

If the o/p of 1st thread is given an i/p to 2nd thread & o/p of 2nd thread is given an i/p to 3rd thread etc then the consistant com.n btw 1st, 2nd & 3rd threads is known as Inter thread com.n

To develop ITC appls we use the pre-defined methods present in java.lang.object & they are called ITC methods.

ITC methods in Java.lang.object :-

1. Public final void wait(long)
2. Public final void wait()
3. Public final void notify();
4. Public final void notifyAll();

Method 1 is used for making the thread to wait for a period of time. Once the waiting time is completed automatically the thread will be entered into ready state. Dnamically,

the next thread to wait untill the CPU burst time of current thread is completed which may not be knowing to the java programmer becoz CPU burst time is not decided by programmer but it is decided by OS. This method is not advisable to use.

method:-2 is used for making the thread to wait without any time. this method is advisable to use.

The above 1&2 methods throws the pre-defined exception called java.lang.InterruptedException [for more explanation see sleep() Previous Pages].

Method:3 is used for transfering one thread at a time from waiting state to ready state.

Method:4 is used for transferring all the threads from waiting state to ready state.

Classical examples of IPC (From IPC):-

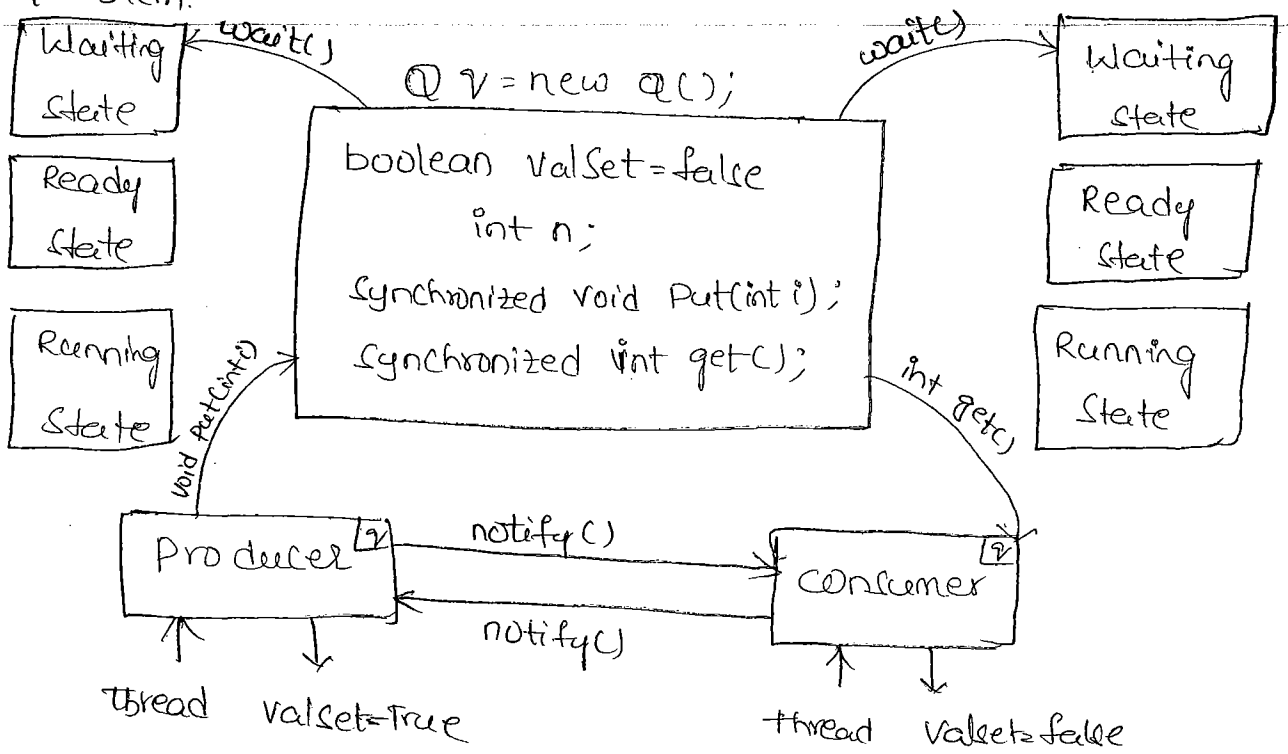
1. Producer-consumer problem
2. Dining-Philosphers problem
3. Readers-writers problem
4. Barber shop problem

Implementation of Producer-consumer problem:-

Producer: It is one of the thread which is always producing the items/values which are consumed by consumer. At any point of time the producer thread will produce only one item at a time but not multiple items.

Consumer:- It is the thread used for consuming the items/values which are produced by producer thread. At any point of time consumer ~~produce~~ will consume ~~produce~~ only one item at a time but not multiple items

The following diagram gives implementation of Producer-Consumer Problem.



* Write a java program which will implement producer consumer problem.

```
//PCPC.java
 Class Q
 {
    boolean valset=false;
    int n;
    Synchronized void put (int i)
    {
      try
      {
        if(valset)
        {
          wait(); //pt is waiting
        }
      } catch (InterruptedException ie)
```

```
{
  sep("prob in thread exec...");
}
  n=i;
  sop("put= "+n);
  valset= true;
  notify();
} // put(-)
synchronized int get()
{
  try
   {
     if(!valset)
      {
        wait(); //ct is waiting
      }
   } catch(InterruptedException ie)
   {
    sep("prob in thred exec---");
   }
    sop("got= "+n);
    valset=false;
    notify();
    return n;
  } // get()
} // Q --- BLC
```

```
class prod implements Runnable
 {
    Q q;
    Prod(Q q)
    {
      this.q=q;
      Thread ct=new Thread (this); //'this' refers current
                                    class obj which impl
      Ct.Start();                   Runnable intf & converting
    }                               into Thread class obj
                                    for calling start()
    Public void run()
    {
      int i=0;
      While(true)
      {
        q.Put(i++);
      } //while
    } // run
 } //prod --- BLC --- Thread

Class cons implements Runnable
 {
   Q q;
   cons(Q q)
   {
     this.q=q;
     Thread pt=new Thread (this);
     Pt.Start();
   }
   Public void run()
   {
     while(true)
     {
```

```
    } // while
  } // run()
} // Cons ---BLC--- Thread
class Facpc
{
  Public Static void main(String [] args)
  {
      Q q = new Q();
      Prod po = new Prod(q);
      Cons co = new Cons(q);
  } // main()
} // Facpc
```

compile the above program

```
javac Facpc.java
```

Run the program

```
java Facpc
```

Output

```
Put=1
got=1
Put=2
got=2
Put=3
got=3
-----
-----
```

In the above o/p Put msg's are generated by Prod thread & got msg's are generated by consumer thread consistently. Hence the consistant com^n btw prod & cons thread is known as "ITC".

* Write a java program which will print name of the programmer defined FGT during the execution of run(), where the user-defined class either extends Thread or implements Runnable.

```
// Exam1.Java
Class Th1  implements Runnable
 {
    Public void run()
    {
      SOP("I am from run();
      Thread t = Thread.currentThread();
        SOP("thread in run() = " + t.getName());  // Thread[Sathya, 5,main]
    }
 }

class Exam1
 {
 Public static void main(String [] args)
   {
     Th1 t1=new Th1();
```

Sol:1 extending thread cls

In the run method write the following code

```
sop("Name of FGT="+getName());
sop("Name of FGT="+Thread.currentThread.getName());
```

Sol:-2 implementing Runnable intf

```
sop("FGT in ~~main~~ run()=Thread.currentThread.getName());
```

```
//Exam2.java
Class Th1 implements Runnable
{
  Th1()
  {
    sop("Th1() --- default");
    Thread t1=new thread(this,"Sathya");
      t1.start();
  }
  public void run()
  {
    sop("i am from run()");
    sop("FGT in run()="+Thread.currentTheard.getName());
  }
}//Th1
Class Exam2
{
  psvm(String[]args)
  {
    sop("main()");
    new Th1();
```

8/9/15

Collection Frame Work (Java.util.*)

→ CFW is one of the additional service/add on service developed by SUN MS developers with the aim of providing high performance to the java based appls

→ To deal with CFW programming we must import a pre-defined package called Java.util.*

Defn of CFW:-

It is one of the standard mechanism which allowses to group multiple values either of same type (or) different type (or) both the types in a single variable with dynamic size. This single variable is called "collection frame work variable".

What are the differences btw Arrays & CFW:-

Array	CFW
1. An array is a collective name given to a group of consequetive memory locations which are all refered by similar type of values	1. CFW is one of the standard mech which allowses to group multiple values either of same type (or) different type (or) both the types in a single vari-able with dynamic size
2. Concept of arrays allowses to organize only homogeneous values but not hetrogeneous values.	2. Concept of CFW allowses to organize both homogeneous & hetrogeneous values.
3. Arrays concept contains fixed size in nature.	3. CFW provides dynamic size in nature.
4. Arrays concept belongs to non-serializable	4. CFW concept belongs to serializable.

Q:- What is the need of going for CFW when an array of objects of Java.lang.object is organizing all types of values

Ans:- Eventhough an array of objects of java.lang.object is organizing all types of values, it has the following problems. (due to arrays nature).

a. Fixed size (additional size leads to ArrayIndexOutOfBoundsException)

b. Non-Serializable.

To overcome these problems industry is always recomended use the concept of CFW.

Advantages of CFW:-

If we use CFW's as a part of our java appl then we get the following advantages.

1. CFW provides high performance
2. CFW provides Expandability (depends on the incoming flow of the data, the size of the CFW variable will be ↑d.)
3. CFW provides Adaptability (the content of one CFW variable is adding to another CFW variable either in the begining (or) in the middle (or) in the ending is called Adaptability.)
4. CFW is one of the algorithmic oriented
   - 4.1. CFW provides built-in sorting techniques.
   - 4.2. CFW provides built-in searching techniques.
   - 4.3. CFW provides preliminary services of data structures.

9/9/15

Types of CFW

In the initial days of SUN MS the concept of CFW was knows as Data structures. The data structures concept was able to fulfill few requirements of s/w industry & unable to fulfill few more requirements of industry. SUN MS Developers revised the existing data structures concept & released to the real world on the name of New CFW & the existing data structures is carry forwarded on name of Legacy CFW.

In other words we've 2 types of CFW's. They are

1. New CFW
2. Legacy CFW

New CFW:-

→ Revised form of existing data structures of java is called New CFW.

→ New CFW is classified into 2 types. They are

1. 1-D / Single CFW
2. 2-D / Double / maps CFW

finding the value by keeping the key is mapping.

1-D CFW in New CFW:-

→ In 1-D CFW we organize the data in the form of either row/column but not in the form of rows & columns (or) not in the form of (K,V) pair

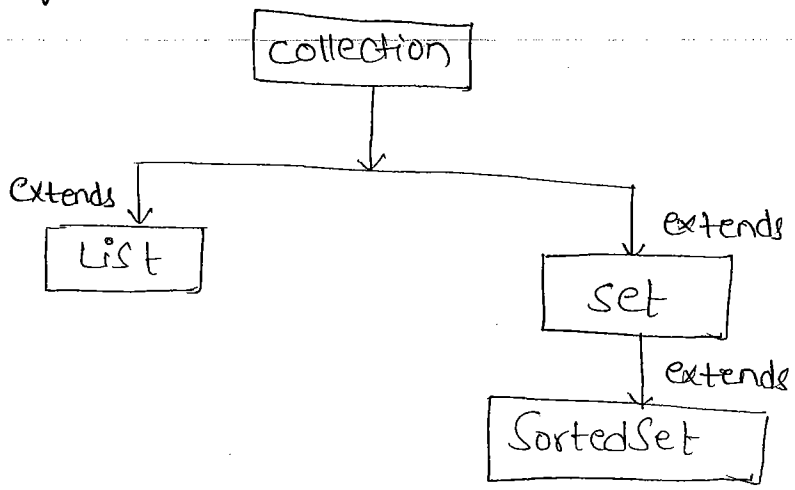
→ To deal with NCFW programming we need to learn corresponding interfaces & classes present in java.util. *

1-D CFW Interfaces:-

We've 4 1-D CFW interfaces. They are

1. Java.util.collection
3. Java.util.Set

Hierarchy chart:-



Here, the objects of Collection, List, Set & SortedSet are called CFW variables.

Differences btw collection, List, Set & SortedSet :-

Objects of Collection & Set are called unit directional objects. becoz they allowses to retrieve the data only in forward direction.

The Objects of List & SortedSet are called multi directional objects becoz they allowses to retrieve the data in forward, backward & Random directions.

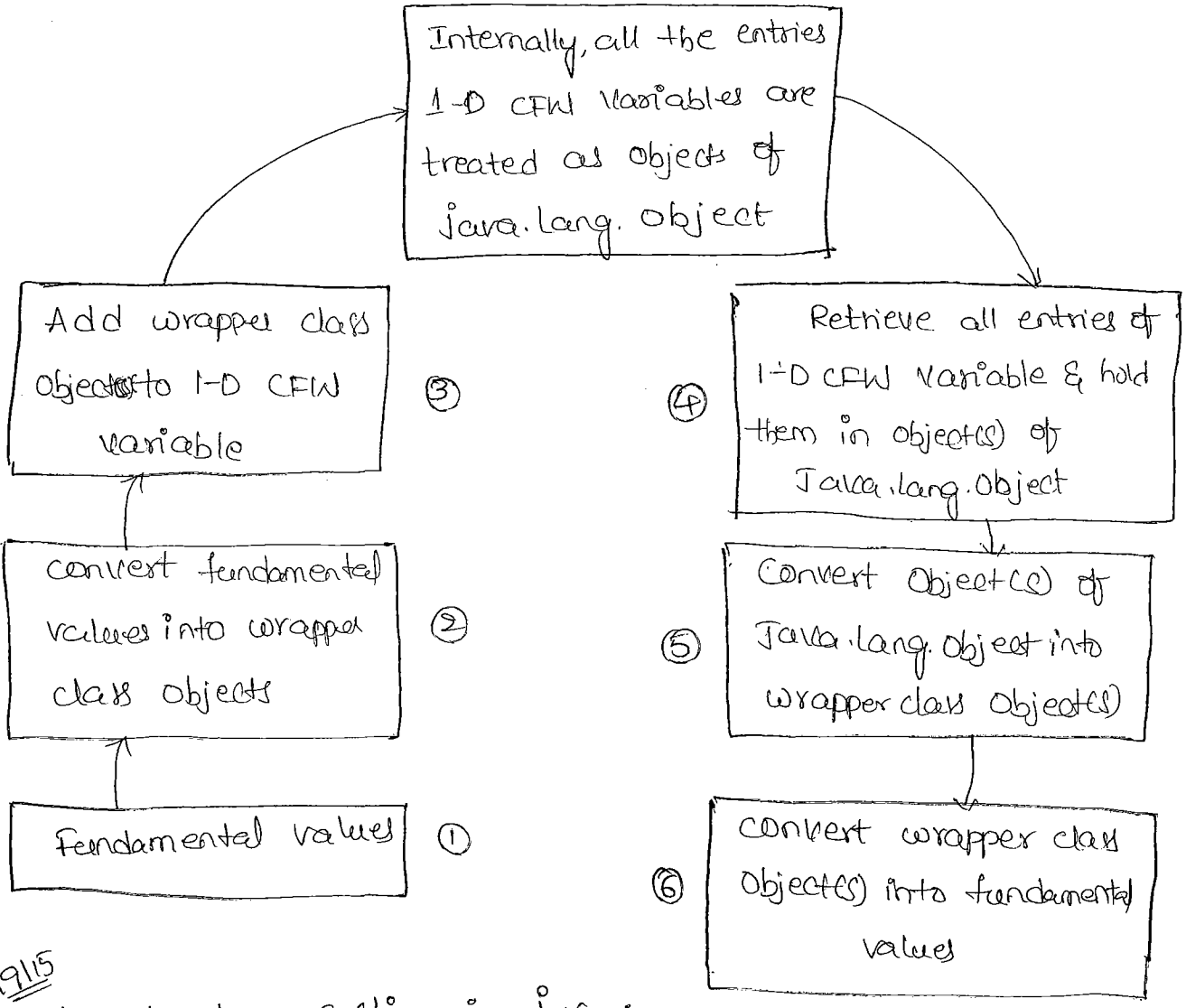
Differentiation factor	Collection	List	Set	Sorted Set
Hierarchy	It is one of the Pre-defined interface available on the Top of hierarchy of I-D CFW interfaces	List is the sub interface of collection	Set is the sub interface of collection	Sortedset is the Sub interface of Set
Functionality	An object of Collection allows duplicate elements	An object of List allows duplicate elements	An object of Set allows unique elements	An object of Sortedset allows unique elements
Display difference	Collection interface object displays the data in Random order	List interface object displays the data in Sorted order	Set interface object displays data in Random order	Sortedset interface object displays the data in Sorted order
Data organization	An object of Collection interface allows to add the data only at end	An object of List interface allowses to organize the data either in the begining or in the middle or in the ending	An object of Set interface allowses to add the data only at end	An object of Sortedset interface allowses to organize the data either in the begin or in the middle or in the ending
Retrieval	collection interface object allows to retrieve the data only in forward direction but in other directions	List intf object allowses to retrieve the data in forward, backward & Random directions	Set intf Object allowses to retrieve the data only in forward direction but not in other directions	Sortedset intf object allowses to retrieve the data in forward, backward & Random directions

1-D CFW Process:-

This process contains 2 phases. They are

1. Grouping/ Assembling Phase
2. Un-grouping/ Deassembling Phase

The following diagram gives sequence of steps for grouping & un-grouping phases



10/9/15

Object type casting in Java:-

Def:- The process of converting reference of base class object into the reference of sub class object is called "Object Type casting."

Syntax:-

Sub class Name Sub obj = (Sub class Name) Base class Obj

Need of object Typecasting:-

1. The result of the Java Program is available in the object of Java.lang.Object.

   EX:- Object obj = "Sathya";

2. To process the result of Object of Java.lang.Object we are not having any methods present in Java.lang.Object.

   EX:- int noc = obj.length(); // Invalid. becoz Length() doesn't exists in Java.lang.Object

3. To process the result of Object of Java.lang.Object, we have the special methods present in sub class of Java.lang.Object.

   EX:- String

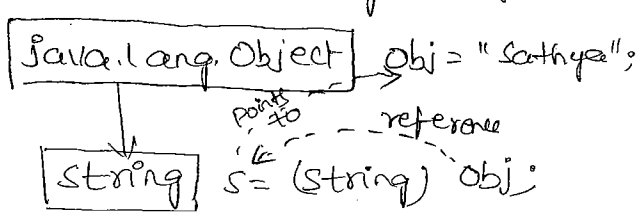
   Public int Length()

4. To process the result of Object of Java.lang.Object, w.r.t the object of Java.lang.Object, we can't access special methods of sub class of Java.lang.object (according to the scope of base class object) see in Inheritance topic.

5. To process the result of object of Java.lang.Object, w.r.t the special methods of its sub class we apply the concept of Object Typecasting

   String s = (String) obj;

   Consider the following diagram.



* Write a Java Program which illustrate the concept of "Fund" type casting (implicit & explicit) & Object type casting

```
//TypeCast.java
Class TypeCast
{
  Public static void main(String [] args)
  {
    Sop("Fund. Type Casting");
    Sop("----------");
    int a=10;
    float b;
     b=a; //fund. implicit type casting
    Sop("Val of a(int)= "+a); //10
    Sop("val of b(float)= "+b); //10.0
    float x=5.5f;
    int y=(int) x; //fund. explicit type Casting
    Sop("val of x(float)= "+x); //5.5
    Sop("val of y (int)= "+y); //5
    int k=65;
    char l=(char)k; //fund. Explicit type casting
    Sop("val of k(int)= "+k); //65
    Sop("val of l(char)= "+l); //A
    Char p='a';
     int q;
     q=p; //fund. implicit type casting
    Sop("val of p(char) = "+p); //a
    Sop("val of q(int)= "+q); //97
```

```
sop("----------");
sop("object type casting");
sop("----------");
Object obj="SATHYA";
sop("val of obj="+obj); //SATHYA
// int noc=obj.length(); invalid
String s=(String)obj; //object type casting
int noc=s.length();
sop("no. of chars "+obj+"="+noc);
Object obj1=100;
// obj1=obj1+1; invalid
Integer io=(Integer)obj1; //object type casting
io=io+1; //auto un-boxing
sop("val of obj1="+obj1); //100
sop("val of io="+io); //101
int x1=io.intValue(); //programatic un-boxing
x1=x1+1;
sop("val of x1="+x1); //102
sop("----------");
```

11/9/15

Methods in Java.util.collection:-

1. Public int size():-

This method is used for minding the no. of elements present in a CFW variable

2. Public boolean isEmpty():-

This method returns true provided CFW variable is empty & it returns false when CFW variable is non-empty

3. public void clear():-

This method is used for removing / clearing the elements of CFW variable.

4. Public boolean contains(Object):-

This method returns true provided the specified values present in CFW variable otherwise it returns false.

EX:-

```
sop(c); // [10, 10.5, A, true]
boolean b = c.contains(10.5);
        sop(b); // true
        b=c.contains("Sathya");
        sop(b); // false
```

C → 1-D CFW variable

10
10.5
A
true

5. Public boolean add(Object):-

This method is used for adding an element to any CFW variable & internally, those elements are treated as objects of Java.lang.Object.

This method returns true as long as we call w.r.t objects of Collection & List irrespective of type of elements we add this also returns true provided we call this method w.r.t Set & SortedSet & adding unique elements. This method returns false provided we add duplicate elements to Set & SortedSet interface objects.

EX:- In JDK1.4

```
int a=10;          → ①
Integer io = new Integer(a);  → ②
c.add(io);         → ③
```

In JDK 1.4

```
C.add(new Integer(10));
```

FROM JDK 1.5 Version onwards

```
C.add(10);
C.add(10.5);
C.add('A');
sop(c); // [10, 10.5, A]
```

C ← 1-D CFW variable

10
10.5
A

6. Public Iterator iterator()

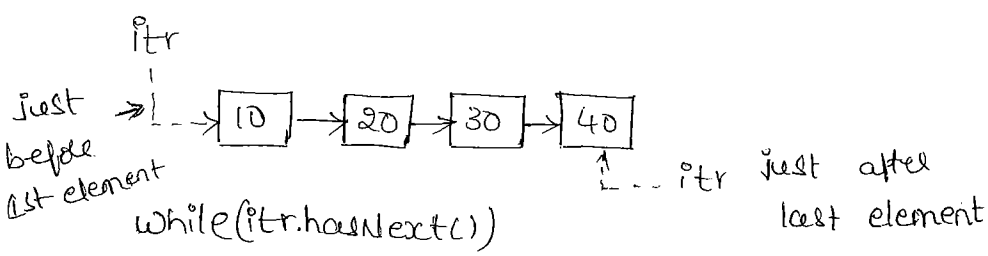
This method is used for retrieving / extracting all the elements from CFW variable & forms a forward directional chain. This method returns an object of iterator interface & it is by default pointing just before the first element of forward directional chain.

Ex:-

```
sop(c); [10, 20, 30, 40]
int s=0;
Iterator itr = c.iterator();
```

C → 1-D CFW variable

10
20
30
40




```
while(itr.hasNext())
{
    Object obj = itr.next(); // 4
    Integer io = (Integer) obj; // 5
        int x = io.intValue(); // 6
        sop(x); // 10 20 30 40
        s=s+x;
}
sop("sum = "+s); // 100
```

7. Public Object[] toArray():-

This method is used for extracting/retrieving all the elements of 1-D CFW variable & holding the array of objects of Java.lang.Object

Ex:-

```
Sop(c); // [10,20,30,40]
  int s=0;
Object obj[]=c.toArray(); //(4)
  for(int i=0; i<obj.length; i++)
  {
    Integer io=(Integer) obj[i]; //(5)
      int x= io.intValue(); //(6)
        Sop(x); //10 20 30 40
        s=s+x;
  }
```

c - 1D CFW variable

10
20
30
40

obj

10	0
20	1
30	2
40	3

Iterator:-

→ It is one of pre-defined interface present in Java.util.*

→ The purpose of Iterator interface object is to extract the data from any CFW variable in forward direction only.

→ By default an object of Iterator is pointing just before the first element of any CFW variable.

Methods:-

1. Public boolean hasNext()
2. public Object next()

Method:-1 is used for checking whether we've next element or not. It returns true provided it contains next element.

Method-2 is used for retrieving next element provided

Methods in Java.util.List :-

WKT List is the sub interface of collection so that all the methods of Collection are inherited into List & List interface contains the following special methods.

1. Public void add(int, Object) :-

This method is used for adding an element to 1-D CFW variable (List) either in the begining (or) in the middle (or) in the last.

Ex :-

0th location - "Sathya"

l.add(0, "Sathya");

2nd location - "Sathya"

l.add(2, "Sathya");

last location - "Sathya"

l.add(4, "Sathya");

l

10	0
10.5	1
A	2
true	3

Q :- How do you add an element "Sathya" at the end of existing elements of List object by using add(int, Object)

Ans :- l.add(l.size(), "Sathya");

2. Public Object get(int) :-

This method is used for obtaining / extracting the value from any 1-D CFW variable by passing valid position. Otherwise we get a predefined exception called Java.lang.IndexOutOfBoundsException

Ex :- Object obj = l.get(1); ← valid position

Sop(obj1); //1.5

Object obj2 = l.get(6); ← Invalid position

Null → throws → Java.lang.IndexOutOfBoundsException

l

10	0
1.5	1
sathya	2

3. Public void remove(Object)

Public Object removeElementAt(int) :-

The above methods are used for removing the elements of the 1-D CFW variable either on the basis of content (or) on the basis of valid position.

Public void removeAll():-

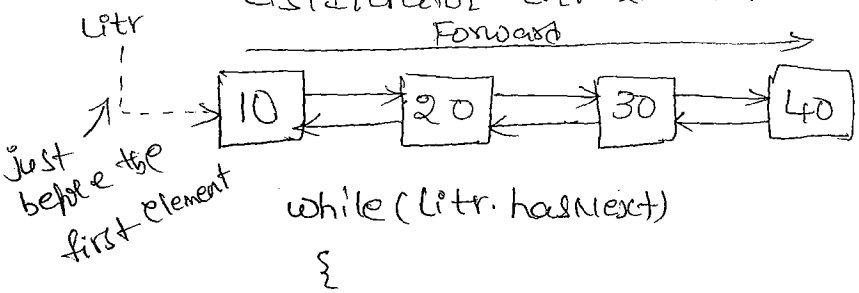
This method is used for removing all the elements of 1-D CFW variable.

4. Public ListIterator listIterator():-

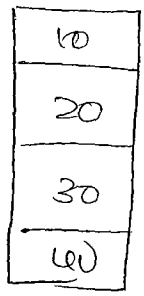
This method is used for extracting/retrieving the elements from 1-D CFW variable & forms bi-directional chain. This method returns an object of ListIterator & it is by default pointing just before the first element.

Ex:-

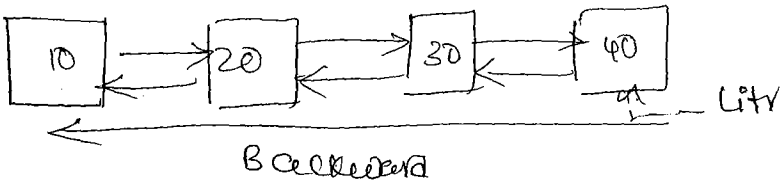
```
sop(l); // [10, 20, 30, 40]
ListIterator Litr=l.listIterator();
```



l - 1-D CFW variable




```
while(Litr.hasNext)
{
  Object obj= Litr.next();
  sop(obj); // 10 20 30 40
}
```



```
int s=0;
while(Litr.hasPrevious())
{
    Object obj=Litr.Previous(); //④
      Integer io=(Integer) obj; //⑤
        int x= io.intValue(); //⑥
          sop(x); // 40 30 20 10
            s=s+x;
}
    sop("sum in BD="+s); //100
```

14/9

Q:- What are the different methods used for retrieving the data from 1-D CFW variables?

Ans:- We've 4 methods to retrieve the data from 1-D CFW variable. They are

1. Public Iterator iterator()
2. Public Object[] toArray()
3. Public Object get(int)
4. Public ListIterator listIterator()

ListIterator:-

→ It is one of the pre-defined sub interface of Iterator present in Java.util.*

→ The purpose of ListIterator interface object is that to retrieve the ~~from~~ data from any 1-D CFW variable in both Forward & Backward directions.

→ Like Iterator interface object ListIterator interface object also by default pointing to just before the first element.

Methods:-

1. Public boolean hasNext()
2. Public Object next()
   inherited from iterator
3. Public boolean hasPrevious()
4. Public Object previous()

Method-3 returns true provided it has previous elements. Otherwise returns false. Metho-4 is used for obtaining previous element provided method-3 must returns true.

Methods in Java.util.Set :-

WKT Set is the sub interface of Collection so that all the methods of Collection are inherited into Set.

"Set interface doesn't contain any special methods except Collection interface methods"

Eventhough the methods of Collection & Set are same, Collection methods are defined in some other pre-defined class in such a way that duplicates are allowed. Whereas Set methods are defined in some other pre defined class in such a way that unique elements are allowed.

In otherwords eventhough the methods of collection & set interfaces are same, whose implementations are different in different pre-defined classes for satisfing their properties.

Methods in Java.util.SortedSet :-

WKT SortedSet is the sub interface of Set interface so that all methods of Set are inherited into SortedSet.

As a part of JDK1.5 & further versions onwards we've the following special methods.

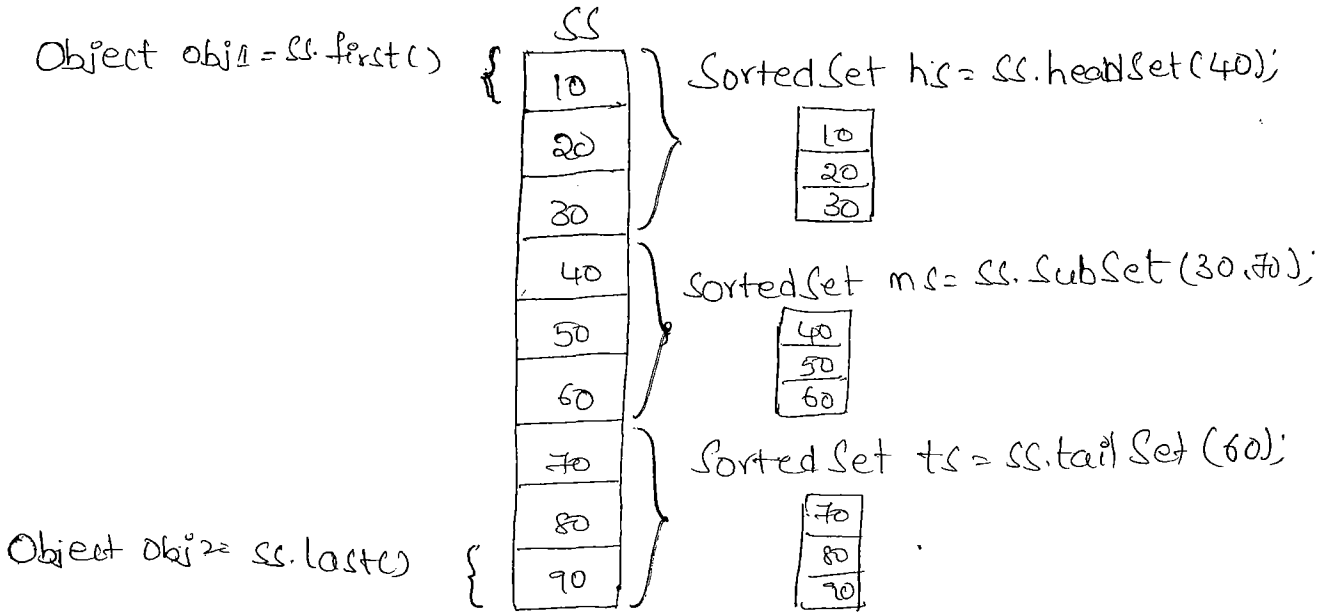
1. Public Object first()
2. Public Object last()
3. Public SortedSet headSet(Object obj*) → xi < obj (Target)
4. Public SortedSet subSet(Object obj1, Object obj2) obj1 < xi < obj2
5. Public SortedSet tailSet(Object obj) → xi > obj

→ methods - 1&2 are used for obtaining 1st & last elements of Sorted Set

→ Method-3 is used for obtaining few top most values. (x_i) which are less than target object obj i.e., $x_i < obj$

→ Method-4 is used for obtaining few middle values (x_i) which are greater than target Object obj1 & less than target Object obj2 i.e., $obj1 < x_i < obj2$

→ Method-5 is used for obtaining few bottom most values (x_i) which are greater than target Object obj i.e., $x_i > obj$



1-D CFW Classes

→ These classes contains def^n for those abstract methods which are inherited from 1-D CFW interfaces.

→ The following table gives 1-D CFW interface name, corresponding 1-D CFW class name & their hierarchy.

1-D CFW Intf name	1-D CFW class Name	Hierarchy
1. Collection	1. AbstractCollection	← implements Collection
2. List	2. AbstractList	← extends AbstractCollection ← implements List
3. Set	3. AbstractSet	← extends AbstractCollection ← implements Set
4. Sorted Set		
	4. AbstractSequentialList	← extends AbstractList
	5. LinkedList 6. ArrayList	← extends AbstractSequentialList
	7. HashSet	← extends AbstractSet
	8. TreeSet	← extends AbstractSet ← implements SortedSet

In the above table the classes 1,2,3 & 4 are belongs to Pre-defined I-D CFW abstract classes. & they are reserved by SUN MS developer for the future innovations in I-D CFW Programming. Since these are abstract we can't create their objects directly & there is no direct usage of these classes.

The classes 5,6,7 & 8 are belongs to I-D CFW concrete classes & we use in our real time applications for meeting with the current industry requirements.

NOTE:- As a part of CFW (any topic of Java) we've 3 types of layers they are

a, Interfaces layer
b, Abstract classes layer
c, Concrete classes layer

LinkedList

→ LinkedList is one of the concept of data structures.

→ The advantages of LinkedList over arrays concept are

* LinkedList allowses to organize homogeneous & hetrogeneous values.
* LinkedList provides dynamic size
* LinkedList provides dynamic inserting operations (we can insert an element either in the begining or in the middle or in the last)
* LinkedList concept is by default belongs to serializable.

To get the above advantages of LinkedList, SUN MS developers provides a predefined class called Java.util.LinkedList

Programatically, getting the advantages of LinkedList is ntngbut create an object of LinkedList class

Ex:-

```
LinkedList ll = new LinkedList();
```

Here, ll is called CFN variable

In Linkedlist data is organizing in the form of nodes. Node contains 2 parts & whose structure is given below.

NodeName

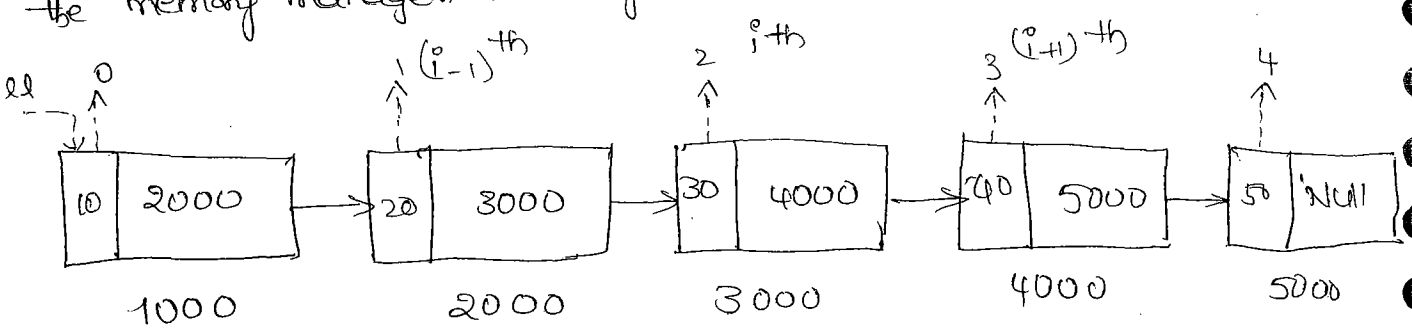
Data Part	Address Part

Here node name represents object name of linkedlist class. Data Part represents type of data being stored. Address part represents address of next node. & address part of last node contains null which indicates end of the LinkedList.

Ex:- Store the values 10, 20, 30, 40, 50 in LinkedList

```
ll.add(10);
ll.add(20);
ll.add(30);
ll.add(40);
ll.add(50);
```

The above elements after adding to LinkedList object the memory management diagram is

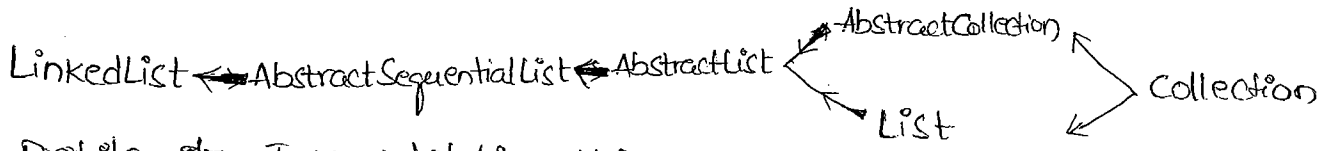


Data organizing data in ll :-

The data organized in linkedlist in such a way that address of ith node stored in address part of $(i-1)^{th}$ node, address part of ith node contain address of $(i+1)^{th}$ part & address part of last node contains Null which indicates end of the LinkedList.

16/9/15

The hierarchy of LinkedList is shown below.



Profile of Java.util.LinkedList :-

Constructors :-

1. LinkedList( )
2. LinkedList(int)

Instance method :-

3. public void addFirst(Object)
4. Public void addLast(Object)
5. public Object removeFirst( )
6. public Object removeLast( )
7. Public Object getFirst( )
8. Public Object getLast( )

→ Constructor-1 is used for creating an object of LinkedList class without specifing no. of nodes to created.

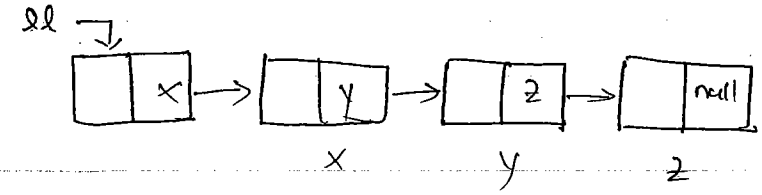
Ex:- LinkedList ll = new LinkedList();

ll

Data Part	address Part

→ Constructor-2 is used for creating an object of LinkedList by specifing the no. of nodes to be created.

Ex:- LinkedList ll = new LinkedList(n);



→ Methods-3&4 are used for adding the elements in the begining & ending of the LinkedList respectively

Ex:-

```
ll.addFirst(10);
    (or)
ll.add(0, 10);
```

first element position

```
ll.addLast(50);
    (or)
ll.add(ll.size(), 50);
```

last element position.

→ Methods-5&6 are used for removing the 1st & last elements of the LinkedList respectively.

Ex:-

```
Object Fobj = ll.removeFirst();
    (or)
Object Fobj = ll.removeElementAt(0);
```

1st element position

```
Object Lobj = ll.removeLast();
    (or)
Object Lobj = ll.removeElementAt(ll.size()-1);
```

last element position

→ Methods-7&8 are used for obtaining 1st & last elements of LinkedList.

Ex:-

```
Object Fobj = ll.getFirst();
    (or)
Object Fobj = ll.get(0);
```

1st element position

```
Object Lobj = ll.getLast();
    (or)
Object Lobj = ll.get(ll.size()-1);
```

last element position

* Write a java program which illustrate the concept of LinkedList & perform various operations

```
// LL.java
import java.util.*;
Class LL
{
  Public static void main(String [] args)
  {
    LinkedList ll = new LinkedList();
    sop("contents of ll = "+ll); // []
    sop(" size of ll = "+ll.size()); // 0
// add the elements to ll - 1-D CFW variable
      ll.add(20);
      ll.add(30);
      ll.add(40);
    sop(" contents of ll = "+ll); // [20 30 40]
    sop(" size of ll = "+ll.size()); // 3
// add the elements 10 & 50 at first & last position
      ll.add(0,10);
      ll.add(ll.size(),50);
    sop(" contents of ll = "+ll); // [10 20 30 40 50]
    sop(" size of ll = "+ll.size()); // 5
// extract the data from ll
    sop("=========");
    sop(" extract the data from ll-iterator()");
    sop("=========");
    int s=0;
    Iterator itr = ll.iterator();
```

```
while(itr.hasNext())
 {
   Object obj=itr.next(); //Step-4
   Integer io=(Integer) Obj; //Step-5
   int x=io.intValue(); //Step-6
    sop("x");
    S=S+x;
 }
sop("sum-iterator()="+S);
sop("=========");
 sop("extract the data from ll-ListIterator()-Forward Dir");
 sop("=========");
 ListIterator litr=ll.ListIterator();
  while(litr.hasNext())
   {
     Object obj=litr.next();
     sop(obj); // 10 20 30 40 50

   } //while
  sop("=========");
 sop("extract the data from ll-listIterator()-Backward direction");
 sop("=========");
 int S1=0;
  while(litr.hasPrevious())
   {
     Object obj=litr.previous();
     Integer io=(Integer) obj;
     int x=io.intValue();
     sop(x);
```

```
sop(" Sum _listIterator() = "+ s1);
sop("=========================");
sop("extract the data from ll-toArray()");
sop("=========================");
Object obj[] = ll.toArray();
for(int i=0; i<obj.length; i++)
{
    sop(obj[i]);
}
sop("=========================");
sop("random data from ll-get");
sop("=========================");
Object robj = ll.get(2);
sop("random element = "+robj); //30
sop("=========================");
}
}
```

18/9

* Write a java program which accept 3 integer values & find the biggest element return all 3 source value along with biggest value.

```
// Big.Java
Package bp;
import java.util.*;
Public class Big
{
    int a,b,c;
    Public void readValues()
    {
        Scanner s=new Scanner(System.in);
```

```
SOP("Enter First Value");
a=Integer.parseInt(s.nextLine());
SOP("Enter second value");
b=Integer.parseInt(s.nextline());
SOP("Enter third Value");
c=Integer.parseInt(s.next Line());
}
Public LinkedList ~~ll=new~~ findBig()
{
   LinkedList ll= new LinkedList();
   if((a==b) && (b==c) && (c==a))
   {
      ll.add("ALL VALUES ARE SAME");
   }
   else
   {
      int big=a;
      if (b>big)
      {
         big=b;
      }
      if (c>big)
      {
         big=c;
      }
// Send the values a,b,c & big
      ll.add("val of a="+a);
      ll.add("val of b="+b);
```

```
    ll.add("val of c="+c);
    ll.add(" biggest= "+big);
  }//else
    return ll;
}
Public void disp(LinkedList ll)
{
    Sop("_____________");
    Sop(" Result");
    Sop("_____________");
    Iterator itr= ll.iterator();
    while (itr.hasNext())
    {
        Object obj=itr.next();
        Sop(obj);
    }
    Sop("_____________");
  }
}
```

Compile the above program

javac -d . Big.java

bp → Big.class

A method of Java is not only returning CFW object but also taking CFW object.

* Write a java main program which makes use of bp.Big class for finding biggest of 3 numbers.

```
// BigDemo.java
import java.util.*;
import bp.Big;
class BigDemo
{
  Public static void main(String [] args)
  {
    Big bo= new Big();
     bo.readValues();
    LinkedList ll = bo.findBig();
     bo.disp(ll);
  }
}
```

Limitations of LinkedList :-

→ LinkedList based applications takes more memory space

→ LinkedList based applications takes more execution time

→ The overall performance of LinkedList is less. (The reason behind these limitations is that data part & address part of a node of LinkedList is explicitly created in heap memory which will take more memory space. & to retrieve the data from heap memory takes more time)

To overcome these limitations we've another pre-defined class called java.util.ArrayList.

NOTE :- If we store any values in Associative memory then they takes neglizible amount of memory space. If we retrieving ... then it takes neglizible

ArrayList :-

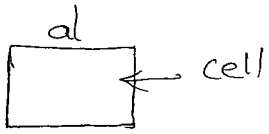
→ It is one of the bottom most concrete sub class in 1-D CFW

→ ArrayList class developed by SUN MS Developers by following the implementation of Associative memory.

→ In ArrayList data is organizing in the form of cells. Cell values stored in Heap memory & cell addresses are storing in Associative memory.

→ Creating an ArrayList is nthg but creating an object of ArrayList class.

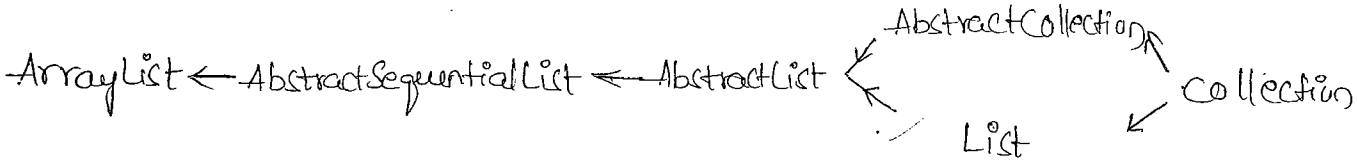
Ex:- ArrayList al=new ArrayList();



Advantages of ArrayList over LinkedList :-

1. ArrayList based applications takes less memory space.
2. ArrayList based applications takes less execution time.
3. Overall performance of ArrayList is more.

The hierarchy of ArrayList is shown below.



Profile of java.util.ArrayList :- Constructor :-

ArrayList():-

This constructor is used for creating an object of ArrayList class.

NOTE:- If we come across LinkedList in our maintenance projects it must be replaced by ArrayList class.

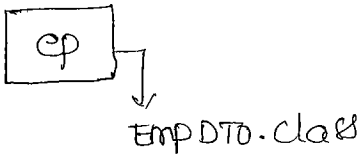
* WAJP which illestrate Data transfer Object (DTO) Data access Object (DAO).

```
//EmpDTO.java
Package cp;
import java.util.*;
Public class EmpDTO
{
  Public ArrayList empDTO()
  {
   Scanner s=new Scanner(System.in);
   Sop(" Enter Emp Number:");
    int eno=Integer.parseInt(s.nextLine());
    Sop(" Enter Emp Name:");
    String name=s.nextline();
    Sop("Enter Emp Sal:");
    int Sal=Integer.parseInt(s.nextLine());
  }  Sop(" Enter Emp Desig:");
    String des=s.nextLine();
  // return all the emp Values
    Arraylist al=new Arraylist();
    al.add("Emp Number="+eno);
    al.add(" Emp Name="+name);
    al.add(" Emp Sal="+Sal);
    al.add(" Emp Desig="+des);
    return al;
}
```

```
public void dispDTO(ArrayList al)
{
    sop("------------------------");
    sop(" Emp DTO Values");
    sop("------------------------");
    Object obj[] = al.toArray();
    for(int i=0; i<obj.length; i++)
    {
      sop (obj[i]);
    }
    sop ("------------------------");
  }
}
```

Compile the above program

javac -d . EmpDTO.java



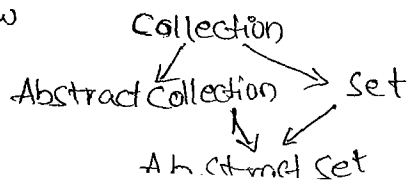
* WAP main program which makes use of cp.EmpDTO for obtaining all the emp values at once & sending all values at once.

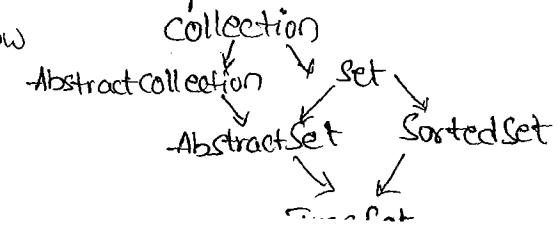
```
// EmpDTO Demo
import cp.EmpDTO;
import java.util.ArrayList;
class EmpDTODemo
{
    public static void main(String [] args)
    {
        EmpDTO eo=new EmpDTO();
        ArrayList al= eo.empDTO();
        eo.dispDTO(al);
    }
}
```

19/9/15

Hash set & Tree set

HashSet	TreeSet
1. An object of HashSet never allows us to add duplicates.	1. An object of TreeSet never allows us to add duplicates.
2. An object of HashSet organizes the data in the form of Hash table by following Hashing mechanism.	2. An object of TreeSet organizes the data in the form of nodes by following Binary Trees Concept.
3. We can't determine in which order HashSet class object display the data i.e, undetermined form beoz SUN MS Developers did not dis-close which type of Hashing mec-hanism followed by SUN MSDev at the time of implementation of HashSet.	3. An Object of TreeSet always gives in sorted order.
4. On HashSet class object the operations like insertion, deletion, modification & Lookup operations takes more time.	4. On Treeset the operations like insertion, deletion, modification and Lookup takes less time.
5. Retrieving the data from HashSet takes more time.	5. Retrieving the data from TreeSet class takes less time.
6. Creating HashSet class is ntng but creating an object of HashSet class	6. Creating TreeSet is ntng but creating an object of TreeSet class
7. The hierarchy of HashSet is shown below	7. The hierarchy of TreeSet is shown below





* WAJP which illustrate the concept of HashSet & TreeSet Operations

```
//hsts.java
import java.util.*;
class hsts
{
  public static void main()
  {
    // HashSet hs=new HashSet(); it gives undetermined order
                                  of o/p.
    TreeSet hs=new TreeSet();
    Sop(" contents of hs = "+hs); //[ ]
    Sop(" size of hs = "+hs.size()); //0
    hs.add(10);
    hs.add(100);
    hs.add(1);
    hs.add(90);
    hs.add(90); //executed but not added to hs
    Sop(" contents in hs = "+hs); //[---]
    Sop(" size of hs = "+hs.size()); // 4
  // extract data from hs
    Sop(" extracting the data from hs.iterator");
    Iterator itr=hs.iterator();
    while(itr.hasNext())
    {
      Object obj=itr.next();
      Sop(obj);
    }
    Sop("extracting data from hs.toArray");
      Object obj[] = hs.toArray();
    for(int i=obj.length-1; i>=0; i--)
    {
      Sop(obj[i]);
    }
    Sop(" random element= "+ obj[2]);
```

* Here ListIterator & get are not used becoz they belongs to List interface only

*WAP which will accept 'N' integer values & print them in Ascending order, descending order, highest & lowest element.

```
// Sortop.java
Package sp;
import java.util.*;
Public class Sortop
{
  private TreeSet ts=new TreeSet();
  Public TreeSet readValues()
  {
    Scanner s= new Scanner(System.in);
    Sop("enter how many values u want to sort");
    int n= Integer.ParseInt (s.nextLine());
// Accept n values from keyboard
    for(int i=0; i<n; i++)
     {
      Sop(" Enter "+(i+1) + "value");
      int v=Integer.parseInt (s.nextLine());
      ts.add(v);
     }
      return ts;
  }
   Public void printDiffOrder(TreeSet ts)
   {
     Sop(" ____________");
     Sop("Asc order");
     Sop(" ____________");
     Object obj[] = ts.toArray();
```

```
for(int i=0; i<obj.length; i++)
 {
   sop(obj[i])
 }
 sop("_____");
 sop(" Desc order");
 sop("_____");
 for(int i=obj.length-1; i>=0; i--)
  {
    sop(obj[i]);
  }
 sop("min val= "+obj[0]);
 sop(" Max val="+obj[obj.length-1]);
 }
} // Sortop - common BLC
```

Compile the above program

```
javac -d . Sortop.java
```

sp → Sortop.java

* KTAMP which makes use of sp.Sortop class

```
// SortDemo.java
import sp.SortOp;
import java.util.*;
class SortDemo
{
  public static void main(String [] args)
  {
    Sortop so=new Sortop();
    TreeSet ts = so.readValues();
    so.printDiffOrders(ts);
    // (or)
    // new Sortop().printDiffOrders(new SortOp().readValues());
  }
}
```

2-D CFW

→ In 2-D CFW the data is organizing in the form of (Key, Value)

→ In (K,V) Value of Key represents unique

↳ Value of Value represents may/may not be unique

(i.e.,) duplicates are allowed

Ex:- Organize Student no's & names

Student ← 2-D CFW Variable Name

Name of the Key → Stno | Sname ← Name of the Value

values of Key →

Stno	Sname
10	Sathya
11	Sathya
12	JG
13	Ravi

← Values of Value

Ex:- Organize account no's & balances

Deposit — 2-D CFW variable

Name of the Key → ACNO | Bal → Name of the value

values of Key →

ACNO	Bal
10	1.5
20	2.5
30	1.5

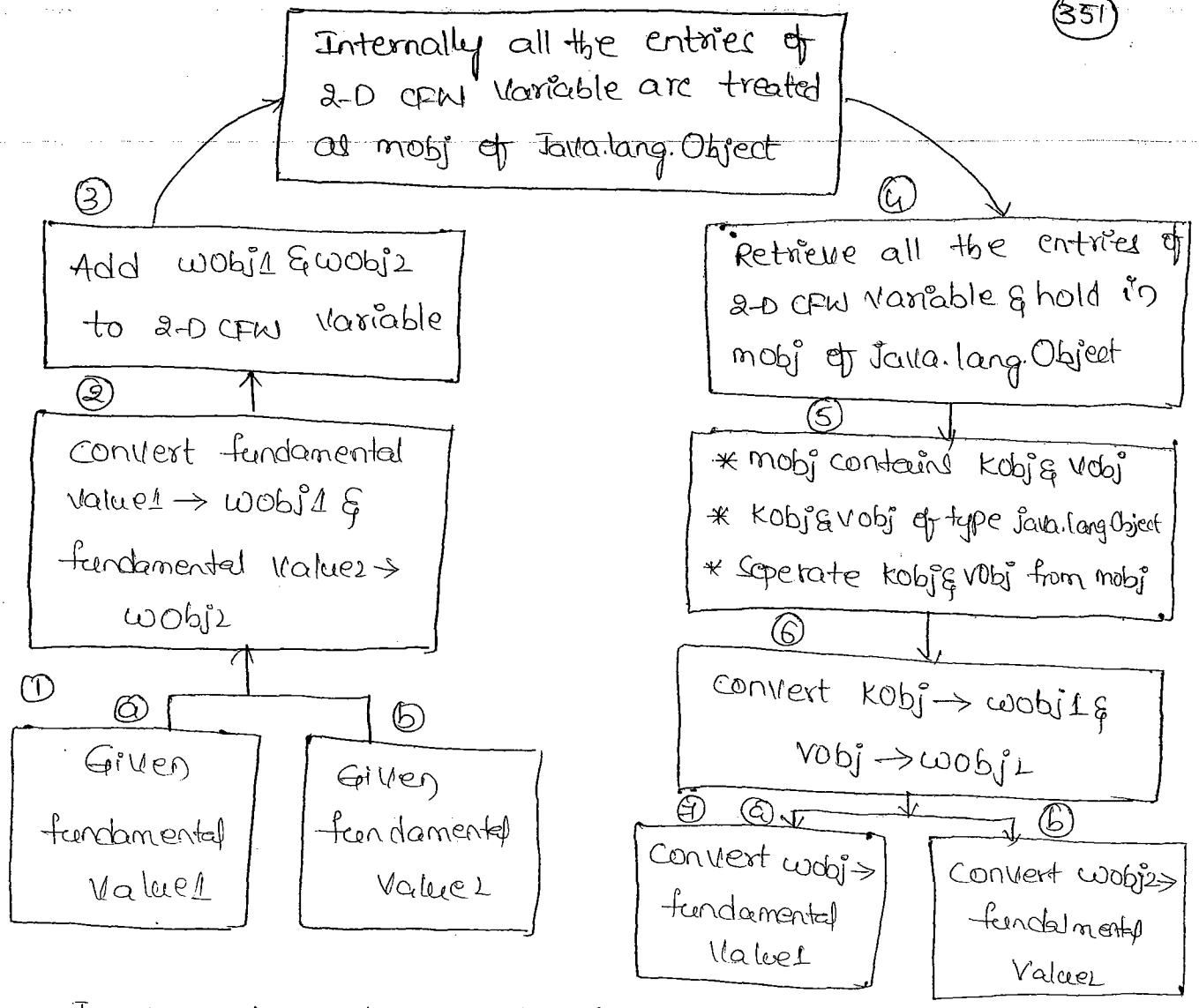
Values of Value

2-D CFW process:-

This process contains 2 phases. They are

1. grouping/assembling phase
2. Un grouping/de assembling phase

The following diagram gives sequence of steps for both grouping &



In the above diagram ①②③ steps are used for grouping phase & ④⑤⑥⑦ are used for un grouping phase.

To deal with 2-D CFW programming we need to learn 2-D CFW interfaces & corresponding classes. & they are present in Java.util. * package

2-D CFW / ~~Inter~~ double/map Interfaces:-

We've 3 types of interfaces. They are

1. Java.util.Map
2. Java.util.Map.Entry
3. Java.util.SortedMap

Java.util.Map :-

→ It is one of the top most pre-defined interface present in 2-D CFW

→ Map interface object organizes the data in the form of (K,V)

→ Map interface object displays the data in random order. (which is nthg but whatever the order we add in the same order the data will be displayed.

Profile of java.util.Map :-

This meth

1. Public int size() :-

This method is used for finding the no. of entries present in 2-D CFW variable.

2. public boolean isEmpty() :-

It returns true provided 2-D CFW variable is empty otherwise it returns false.

3. Public void Put(Object, Object) :-

This method is used for inserting (K,V) in 2-D CFW variable & it can also be used for modifying the existing value of 2-D CFW variable with new value.

Ex:-

m ← 2-D CFW variable

```
m.put(10,1.5f);
m.put(20, 2.5f);
m.put(30, 1.5f);
```

Inserted Entries →

10	1.5
20	2.5
30	1.5

```
sop(m); // { (10=1.5), (20=2.5), (30=1.5) }
```

```
m.put(10, 9.5f); }
```

modified entry →

10	9.5
20	2.5
30	1.5

```
sop(m); // { (10=9.5), (20=2.5), (30=1.5) }
```

4. Public Object get(Object):-

This method is used for obtaining the value of Value by passing the value of Key.

Ex:- Get the balance of ACNO:-10.

```
Object bal = m.get(10);   // found
```

[9.5]

```
sop(bal); // 9.5
Object bal1 = m.get(100);   // Not found
```

[Null]

```
sop(bal1); // Null
```

5. Public Set keySet():-

This method is used for obtaining the set of keys from 2-D CFW variable. Pass these set of keys to get() & obtain values of Value.

Ex:-

```
sop(m); // {(10=1.5), (20=2.5), (30=1.5)}
Set s = m.keySet();
// s = [10,20,30]
```

m ← 2-D CFW variable

10	1.5
20	2.5
30	3.5

Part-II: keySet() → Set → iterator()

```
Iterator itr = s.iterator();
```

itr → [10] → [20] → [30]

```
while(itr.hasNext())
{
    Object kobj = itr.next();
    Object vobj = m.get(kobj);
    Integer io = (Integer)kobj;
    Float fo = (Float) vobj;
```

```
int acno = io.intValue();
float bal = fo.floatValue();
sop(bal + " is the bal of " + acno);
}
```

EX 2 :-

```
sop(m); // { (10=1.5), (20=2.5), (30=1.5) }
Set s = m.keySet();
    // s = [10 20 30]
```

m ← 2-D CFW variable

10	1.5
20	2.5
30	1.5

Part-2 : keySet() → Set → toArray()

```
Object kobj[] = s.toArray();
for(int i=0;i<kobj.length;i++)
{
  Object vobj = m.get(kobj[i]);
  Integer io = (Integer) kobj[i];
  Float fo = (Float) vobj;
  int acno = io.intValue();
  float bal = fo.floatValue();
  sop(bal + " is the bal of " + acno);
}
```

kobj

10	0
20	1
30	2

221

6. public Set entrySet() :-

This method is used for retrieving/extracting all the entries of 2-D CFW variable & holding in the object of Set.

EX-3

```
sop(m); // { (10=1.5), (20=2.5), (30=1.5) }
Set s = m.entrySet();
    // s = [(10=1.5), (20=2.5), (30=1.5)];
```

m → 2-D CFW variable

10	1.5
20	2.5
30	1.5

Inter entrySet() → Set → iterator()

```
Iterator itr=S.iterator();
```

itr → [10=1.5] → [20=2.5] → [30=1.5]

```
while(itr.hasNext())
{
  Object mobj=itr.next(); //④
  Map.Entry me=(Map.Entry) mobj;   // 10=1.5
  Object kobj=me.getKey();
  Object vobj=me.getValue();      //⑤
  Integer io=(Integer) kobj;
  Float fo=(Float) vobj;          //⑥
  int acno=io.intValue();
  float bal=fo.floatValue();      //⑦ ④&⑤
}
```

→ Object type casting

EX 4 :- `Sop(m); // {(10=1.5), (20=2.5), (30=1.5)}`

m → 2-D CPM variable

10	1.5
20	2.5
30	1.5

```
Set S=m.entrySet();
// S = [(10=1.5), (20=2.5), (30=1.5)]
```

Path :- entrySet() → Set → toArray()

mobj

10=1.5	0
20=2.5	1
30=1.5	2

```
Object mobj[]=S.toArray();
for(int i=0; i<mobj.length(); i++)
{
  Map.Entry me=(Map.Entry) mobj[i];
  Object kobj=me.getKey();
  Object vobj=me.getValue();      //⑤
  Integer io=(Integer) kobj;
  Float fo=(Float) vobj;          //⑥
  int acno=io.intValue();
  float bal=fo.floatValue();      //⑦
```

→ Object type casting

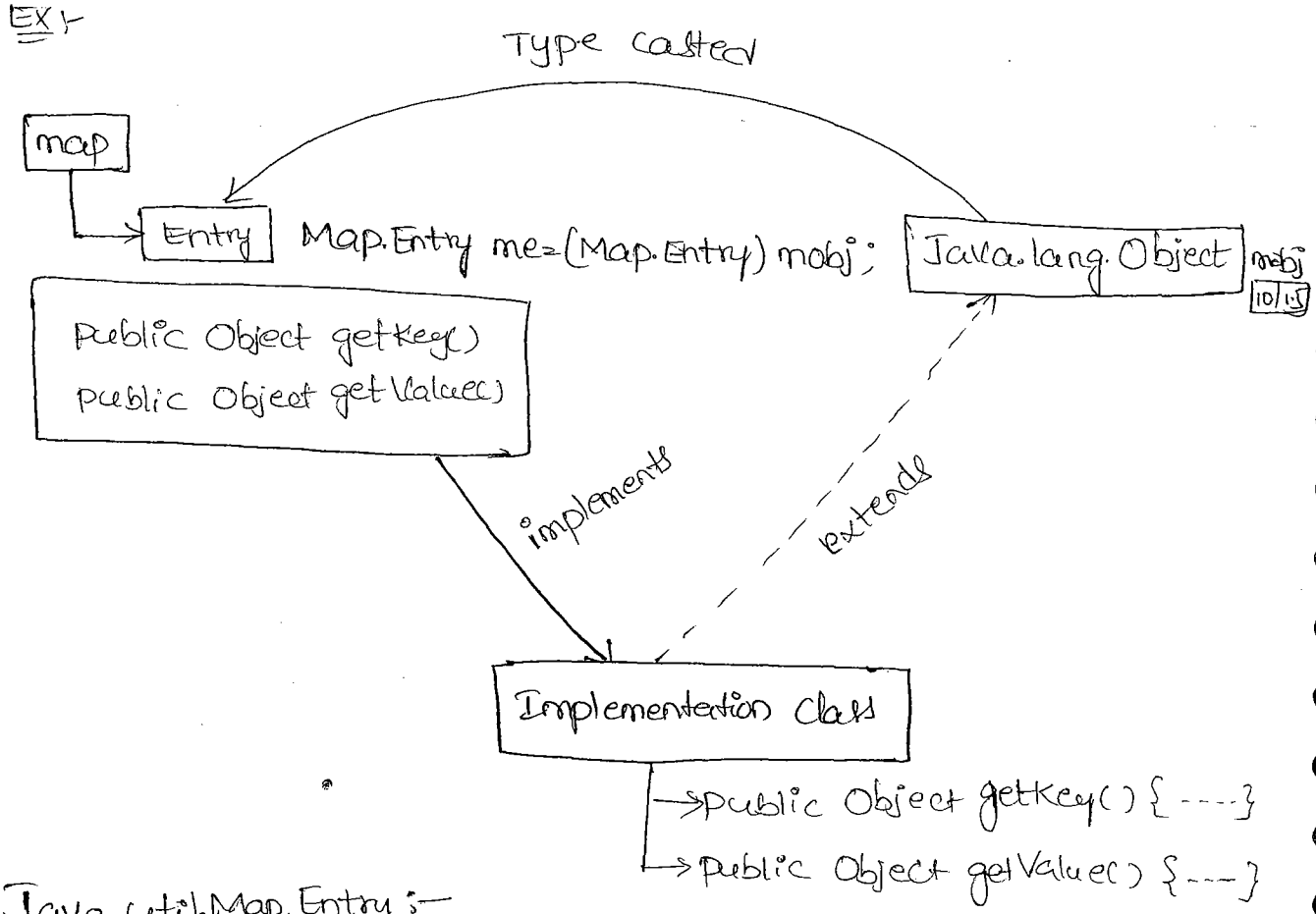
In the above examples the type of object typecasting is that Object of java.lang.Object is typecasted to interface object (Map.Entry).

Formulae

(object to interface typecasting)

An object of java.lang.Object which is super class of implementation class of super interface is typecasted to object of super interface of implementation class which is the sub class of java.lang.Object.

EX:-



Java.util.Map.Entry :-

→ In Map.Entry, Entry is one of the nested/inner interface & Map is one of outer interface.

→ The purpose of Map.Entry is that to seperate key value & value from any 2-D CFW variable.

Methods :-

1. Public Object getKey()
2. Public Object getValue()

The above two methods are used for seperating value of Key & value of value from any 2-D CFld variable.

Java.util.SortedMap :-

→ It is one of the sub interface of Map so that all the methods of Map are inherited into SortedMap.

→ SortedMap interface object also organizes the data in the form of (key, value).

→ SortedMap interface object displays the data in sorted order based on the value of key

Methods :-

1. Public Object first()
2. Public Object last()
3. Public SortedMap headMap(object Kobj) → $x_i < Kobj$ (Target object)
4. Public SortedMap subMap(Object kobj1, object kobj2) → $kobj_1 < x_i < kobj_2$
5. Public SortedMap tailMap(object kobj) → $x_i > Kobj$

For more explanation see the methods of SortedSet

Object Fmobj = sm.first();

10.1.5

10	1.5
20	2.5
30	1.5
40	2.5
50	2.8
60	1.9
70	8.5
80	1.5
90	9.5

SortedMap hm = sm.headMap(40);

10	1.5
20	2.5
30	1.5

SortedMap mm = sm.subMap(30, 70);

40	2.5
50	2.8
60	1.9

SortedMap tm = sm.tailMap(60);

70	8.5
80	1.5

Object Lmobj = sm.last()

2-D double CFW classes / ~~double~~ / Map classes :-

These classes contains implementation for the abstract methods of 2-D CFW interfaces.

The following table gives 2-D CFW interface name corresponding class name & their hierarchy.

2-D CFW interface Name	2-D CFW Class Name	Hierarchy
1. Map	1. AbstractMap	$\xleftarrow{\text{implements}}$ Map
2. Map.Entry	2. a. HashMap.Entry	$\xleftarrow{\text{implements}}$ Map.Entry
	b. TreeMap.Entry	$\xleftarrow{\text{implements}}$ Map.Entry
3. SortedMap	3. AbstractSortedMap	$\xleftarrow{\text{extends}}$ AbstractMap; $\xleftarrow{\text{implements}}$ SortedMap
	4. HashMap	$\xleftarrow{\text{extends}}$ AbstractMap
	5. TreeMap	$\xleftarrow{\text{extends}}$ AbstractSortedMap

In the above hierarchy chart the classes ① & ③ are belongs to pre defined abstract classes in 2-D CFW & they reserved by SUN MS developers for future innovations in 2-D CFW & we may not be using directly in our real world applications.

The classes ④ & ⑤ are two bottom most pre defined concrete classes in 2-D CFW & we use them in our applications.

~~HashMap & TreeMap~~

* Differences btw HashMap & TreeMap :-

HashMap	TreeMap
1. An object of HashMap never allows us to add duplicates.	1. An object of TreeMap never allows us to add duplicates
2. An object of HashMap organizes the data in the form of Hash table by following Hashing mechanism.	2. An object of TreeMap organizes the data in the form of nodes by following Binary Trees concept.
3. We can't determine in which order HashMap class object display the data ie, undetermined form because SUN MS Dev didn't disclose which type of Hashing mechanism followed by ~~Hashing~~ SUN MS Dev at the time of implementation of HashMap	3. An object of TreeMap always gives in sorted order
4. On HashMap class object the operations like insertion, deletion, modification & lookup operations takes more time	4. On TreeMap the operations like insertion, deletion, modification & lookup operations takes less time.
5. Retrieving the data from HashMap takes more time.	5. Retrieving the data from TreeMap takes less time.
6. Creating HashMap class is nthg but creating an object of Hash Map class.	6. Creating TreeMap is nthg but creating an object of TreeMap class
7. The Hierarchy of HashMap is shown in below.	7. The hierarchy of TreeMap is shown in below.

Map
↓
AbstractMap
↓
HashMap

Map
↓
AbstractMap SortedMap
↘ ↙
AbstractedSortedMap
↓

23/9/15

* WAJP which illustrate the concept of HashMap & TreeMap.

```
//hmtm.java
import java.util.*;
Class hmtm
{
 Public static void main(String [] args)
 {
  //HashMap hm=new HashMap();
  TreeMap tm=new TreeMap();
  sop("contents of tm = "+tm); // { }
  sop(" size of tm: "+tm.size()); //0
// add the data to tm
    tm.put(100, 2.4f);
    tm.put(1,2.5f);
    tm.put(10, 2.4f);
    tm.put(90,2.4f);
    tm.put(90, 12.4f); //executed but not added & considered
     as modified entry
    sop("Contents of tm="+tm); // {---}
    sop("size of tm="+tm.size()); //4
// extract the data from 2-D CFW variable
    sop("Extract the data from tm_keySet()_Set_iterator()
    _Ex1_ Seeeee in the book");
    sop("_____");
     set s = tm.KeySet();
     Iterator itr = s.iterator();
     for (;itr. hasNext();)
```

```
Object kobj=itr.next();
Object vobj=hm.get(kobj);
Integer io=(Integer)kobj;
Float fo=(Float) vobj;
int acno=io.intValue();
float bal=fo.floatValue();
Sop("bal +"is the bal of"+ acno);
}//for
Sop("_____");
Sop("Extract the data from hm-entrySet()-Set-toArray()-
  Exp-seeeee in the book");
Sop("_____");
  Set s1=hm.entrySet();
  Object mobj[]=s1.toArray();
  for(int i=0;i<mobj.length;i++)
   {
     Map.Entry me=(Map.Entry)mobj[i];
     Object kobj=me.getKey();
     Object vobj=me.getValue();
     Integer io=(Integer)kobj;
     Float fo=(Float) vobj;
     int acno=io.intValue();
     float bal=fo.floatValue();
     Sop(bal+"--->"+acno);
   }
  Sop("_____");
}
}
```

* KLAJP which will implement Phone book operations such as adding no. of contacts, search for a contact no, display the contacts etc.

```
// PhoneBook.java
Package Pb;
import java.util.*;
Public class PhoneBook
{
  private Scanner s=new Scanner(System.in);
  private Treemap Pnb=new Treemap();
  Public void createphoneBook()
  {
   SOP("Enter How many contancts");
    int noC=Integer. parseInt(s.nextLine());
  //accept individual contacts(cname, phno)
  for(int i=0; i<noC; i++)
  {
    SOP("Enter "+(i+) + "contact details");
    SOP("----------------------------------");
    SOP("Enter contact Name");
    String cname=S.nextline();
    SOP("Enter contact Number");
    int cno=Integer. parseInt(S.nextLine());
    Pnb. put(cname, cno);
    SOP("----------------------------------");
  } //for
```

2. // CreatePhoneBook()

```
Public void searchPhoneBook()
{
    sop("Enter contact Name:");
    String cname=S.nextLine();
    Object phno=phb.get(cname);
    sop("_____________________");
    if(phno!=null)
    {
      sop(phno+"is the contact number of"+cname);
    }
    else
    {
      sop(cname+" related contacts does not exists");
    }//else
    sop("_____________________");
  }//search phoneBook()

Public void dispPhoneBook()
{
    Set S=phb.entrySet();
    Iterator itr=S.iterator();
    while sop("_____________________");
    sop(" contact name | contact Number");
    sop("_____________________");
    while(itr.hasNext())
    {
      Object mobj=itr.next();
      Map.Entry me=(Map.Entry) mobj;
      Object cname=me.getKey();
      Object phno=me.getValue();
```

```
sop("cname + "    " + phno);
} // while
sop("____________________");
} //dispPhoneBook()
} // phoneBook - common BLC
```

compile the above program — javac -d . phoneBook.java

* WAJ main program which makes use of Pb.phoneBook class

Pb → phoneBook.class

```
// phoneBookDemo.java
import pb.phonebook;
class phoneBookDemo
{
    public static void main(String [] args)
    {
        PhoneBook p = new PhoneBook();
        p.CreatePhoneBook();
        p.SearchPhoneBook();
        p.dispPhoneBook();
    }
}
```

* WAJP which will accept no. of states & capitals & organize them in a single unit. Accept the state name & find its capital.

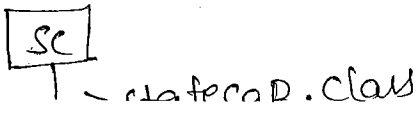
```
// Statecap.java
package sc;
import java.util.*;
public class Statecap
{
Public void acceptStateCap()
```

```
Treemap stc=new Treemap();
public void ~~acceptstate~~ createstatecap()
{
  sop("Enter How many states");
  int nos=Integer.parseInt(s.nextLine());
//accept individual states(state,cap);
  for(int i=0; i<nos; i++)
  {
    sop("Enter"+(i+1)+" state details");
    sop("_____________________");
    sop("Enter state name");
    String Sname=s.nextLine());
    sop("Enter capital");
    String cap=s.nextLine());
    stc.put(state, cap);
    sop("_____________________");
  }//for
}// createstatecap()
public void acceptstatecap()
{
  sop("Enter state name");
  String state=s.nextLine());
  Object cap=stc.get(state);
  sop("_____________________");
  if(cap!=null)
  {
    sop(cap+" is the capital of "+same);
  }
```

```
else
 {
  sop("enter correct capital of statename);
 }
 sop("_____");
} // acceptstatecap
Public void dispstate()
 {
   set p=sn.entrySet();
   Iterator itr=p.iterator();
   sop("_____");
   sop(" State |capital");
   sop("_____");
   while(itr.hasNext())
   {
     Object mobj=itr.next();
     Map.Entry me=(Map.Entry)mobj;
     Object sname=me.getKey();
     Object cap=me.getValue();
     sop("capital of "+sname+" is "+cap);
   }
   sop("_____");
 } //dispstate()
} // Statecap
```

compile the above program

javac -d . Statecap.java



→ Main program

```
//StateDemo.java
import sc.statecap;
class stateDemo
{
  psvm(String [] args)
  {
    statecap se=new statecap();
        se.createstatecap();
        se.acceptstatecap();
        se.dispstate();
  }
}
```

24/9/15

Legacy collection Framework

Def:- Renamed form of existing data structures of java is called legacy CFW.

Differences btw LCFW, NCFW:-

Legacy CFW	New CFW
1. Renamed form of existing data structures of java is called LCFW.	1. Revised form of existing data structures of java is called NCFW.
2. The methods of classes & interfaces of LCFW are belongs to synchronized.	2. The methods of classes & interfaces of NCFW are belongs to synchronized.
3. LCFW by default provides thread safety results.	3. NCFW by default provides non thread safety results.
4. Industry is always recommended to use LCFW as a part of non server side appls. (client side appls)	4. NCFW is always recommended to use in server side appls (beoz eventhough NCFW is non synchronized, in built synchronization is provided by server s/ws)

Like NCFW, LCFW also contains 1-D & 2-D CFW's with collection of classes & interfaces & they are present in java.util.* package. They are

1. Enumeration
2. Vector
3. Stack } 1D CFW
4. Dictionary
5. Hashtable
6. Properties

Enumeration:-

It is one of the pre-defined interface present in java.util.*

→ The purpose of Enumeration interfaces object is that to extract the data from any LCFW variable in forward direction only.

→ The functionality of Enumeration is more or less similar to Iterator interface object but Enumeration info object belongs to Synchronized & Iterator info object belongs to non-synchronized

→ Like iterator info object Enumeration interface object also pointing just before the first element.

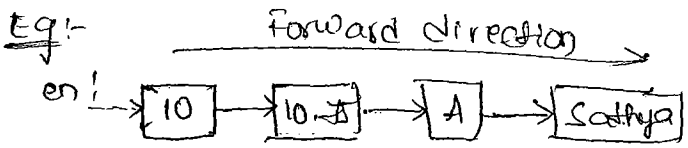
Methods :-

1. public boolean hasMoreElements()
2. public Object nextElement()

Method:1 returns true provided it contains more elements. Otherwise it returns false.

Method:2 is used to obtain next element of any Legacy CFW provided method:1 returns true.

Eg:-



```
while(en.hasMoreElements())
{
  Object obj=en.nextElement();
  Class c = obj.getClass();
        ↳ [ Class Java.lang.Integer (or) float etc ]
  String cname=c.getName();
        ↳ [ Java.lang.Integer ]
  if(cname.equals("java.lang.Integer")):
  {
    Integer io=(Integer)obj;
      int x=io.intValue();  // Dynamic object Typecasting
  }
  if(cname.equals("java.lang.Float"))
  {
    Float fo= (Float)obj;
      float x= fo.floatValue();   // DOTC
  }
  - - - - - - - -
  - - - - - - -
}//while
```

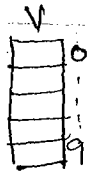
Vector:-

It is one of the concrete class in LCFW, the functionality of vector is more or less similar to Arraylist but vector belongs to synchronized & Arraylist belongs to non-synchronized.

→ In Arraylist the data is organized in the form of cells. cell values are stored in Heap memory & cell addresses are stored in associative memory.

→ Creating a vector is nothing but creating an object of vector class

Eg:- Vector v = new vector();



The default capacity of vector class object is 10 cells & default size of vector class object = 0.

Profile of java.util.Vectors:-

Constructors:-

1. vector()
2. Vector(int)

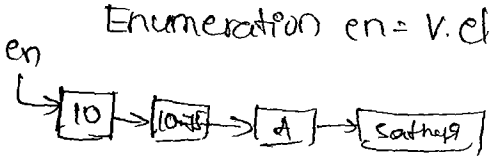
Instance methods:-

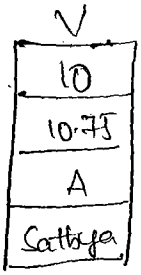
3. Public int capacity();
4. public int size();
5. public void addElement(object);
6. public void insertElementAt(int, Object); → add elements at middle
7. public void setElementAt(int, Object) → replaces
8. public Object removeElementAt(int) → removes elements at (position)
9. Public void remove(object) →
10. Public void removeAll(); → remove all
11. public Object getElementAt(int) → gives element at (position)
12. public Enumeration elements();

Method-12 is used for retrieving all the elements from vector & forms forward directional chain & it returns an object of Enumeration.

Ex:- sop(v); [10, 10.75, A, Sathya]

Enumeration en = v.elements();





```
while(en.hasMoreElements())
{
  object obj=en.nextElement();
  Sop(obj); //10 10.75 A sathya
}
```

* WAJP with illustrate the concept of operations on vector along with dynamic object Typecasting.

```
// VectDemo.java
import java.util.*;
class VectDemo
{
  public static void main(String [] args)
  {
    Vector V=new Vector();
    Sop("content of V=" +V); // [ ]
    Sop(" size of V= "+V.size()); //10
    Sop(" Cap of V=" +V.capacity()); //10
// add elements to V
    V.addElements(10);
    V.addElement(10.75);
    V.addElement('A');
    V.addElement(true);
    V.addElement(20);
    V.addElement(20.35f);
    Sop("content of V= "+V); //[----]
    Sop(" size of V=" + V.size()); //4
    Sop(" cap of V=" + V.capacity()); //10
// extract the data from V
    Sop(" ————————");
    Sop(" Data of V ");
```

```
int is=0;
float fs=0.0f;
Enumeration en=v.elements();
while(en.hasMoreElements())
{
  Object obj=en.nextElement();
   sop(obj);
  Class c=obj.getClass();
  String cname=c.getName();
  if(cname.equals("java.lang.Integer"))
  {
    Integer io=(Integer)obj;
     int x=io.intValue();
     is=is+x;
  }
  if(cname.equals("java.lang.Float"))
  {
     Float fo=(Float)obj;
      float x=io.FloatValue();
      fs=fs+x;
  }
} //while
sop("------------------------");
 sop("Int sum="+is);
 sop("Float sum="+fs);
 sop("------------------------");
}
}
```

NOTE:- Rewrite Arraylist class example [EmpDTO.java] with vector.

~~Next~~

Stack :-

It is one of the distinct feature of Data structures & whose realtime implementations are given below.

1. Converting Infix expressions into either prefix (or) postfix expressions at compile time.
2. Evaluation of either prefix (or) postfix expressions at runtime.
3. Evaluation of ordinary method calls (method chaining) & constructors communication.
4. Evaluation of recursive method calls

To get these internal implementations of stack as a part of java programming we have a predefined class called java.util.Stack. which is one of the sub class of vector.

→ The basic working internal principle of the stack is LIFO (Last in First Out - is says the element which is inserted last is that element will be removed first)

→ Creating the stack is nthg but creating an object of stack class.

Ex:- Stack S = new Stack();

26/9 **Stack memory management :-**

Stack Relative Positions	Stack absolute positions	S	Array absolute positions	Array Relative positions
5	4	10	0	1
4	3	20	1	2
3	2	30	2	3
2	1	40	3	4
1	0	50	4	5

Sop(s); // 10 20 30 40 50 R
Associatively

Profile of java.util.Stack:-

Constructors:-

1. Stack()

Instance methods

2. public boolean empty()
3. public void push(object)
4. public Object pop()
5. Public object peak()
6. Public int search(Object)

→ Constructor:1 is used for creating an object of Stack class.

→ Method:2 returns true provided Stack is empty otherwise it returns false

→ Method:3 is used for inserting an element into the stack.

→ Method:4 is used for removing the top most element.

→ Method:5 is used for extracting top most element from the stack.

→ Method:6 is used for searching an element from the stack, it returns stack relative Position (SRP) when an element is found. Otherwise it returns -1 (indicates element not found)

* WAP which illustrates stack operations.

```
// StackDemo.java
import java.util.*;
class StackDemo
{
 Public static void main(String[] args)
 {
   Stack s = new Stack();
```

```
sopc" contents of s = "+s); //[]
sopc" size of s = "+s.size()); //0
sopc" is s empty? = "+s.empty()); // true
// add the elements to 's'
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);
    s.push(50);
sopc" contents of s = "+s); //[10 20 30 40 50]
sop(" size of s = " +s.size()); //5
sopc"is s empty? = "+s.empty()); // false
// delete top most element
    sopc" deleted element = "s.pop()); //50
    sopc" contents of s after pop = "+s); // [10 20 30 40]
// extract top most element
    sopc"top most element = "s.peek()); //40
    sopc" contents of s after peek = "+s); // [10 20 30 40]
// search elements 10 & 100
    int srp1 = s.search(10);
    sop(" stack rel" position of 10 = " +srp1); //4
    int srp2 = s.search(100);
    sopc" stack rel" position of 100 = "+srp2); // -1
  }
}
```

Dictionary ≃ Map :-

→ contain pure abstract methods

→ It is one of the legacy 2DCFW interface

→ It also having (K,V)

Instance Methods :-

1. public int size()
2. public boolean isEmpty()
3. public void put(Object, Object)
4. public Object get(Object)
5. public void remove(Object) ← Key
6. public Enumeration keys()

} same as Map interface in 2D-CFW

Method:6 is used for extracting set of keys from 2-D legacy CFW variable & pass those keys to get() & obtain the value of value.

D → Object of Dictionary

10	1.5
20	2.5
30	1.5

Ex:- SOP(D); // {(10=1.5), (20=2.5), (30=1.5)}

```
Enumeration en = D.keys();
```

en → [10] → [20] → [30]

```
while(en.hasMoreElements())
{
    Object kobj = en.nextElement();
    Object vobj = D.get(kobj);
    SOP(vobj + " is the val of " + kobj);
}
```

HashTable ≃ HashMap :-

HashTable ↓ Syn — not contains null values

HashMap ↓ Non-Syn — contains null

→ It is one of the 2-D LCFW subclass of dictionary & present in java.util.*

→ HashTable class object organizes the data in the form of (K,V) by following hashing mechanism.

→ We can't determine in which order HashTable class object displays the data.

→ The functionality of HashTable is more (or) less similar to HashMap but HashTable class object belongs to synchronized & it never allows null values, where as HashMap class object belongs to non-synchronized & once it allows null values.

* Rewrite HashMap related program (hmtm.java see previous pages) by using HashTable)

Limitations of HashTable :-

1. HashTable class object is unable to read from properties/ resource bundle file.
2. HashTable class object is unable to read environmental variable names & environmental variable values.
3. HashTable class object is unable to develop flexible appls.

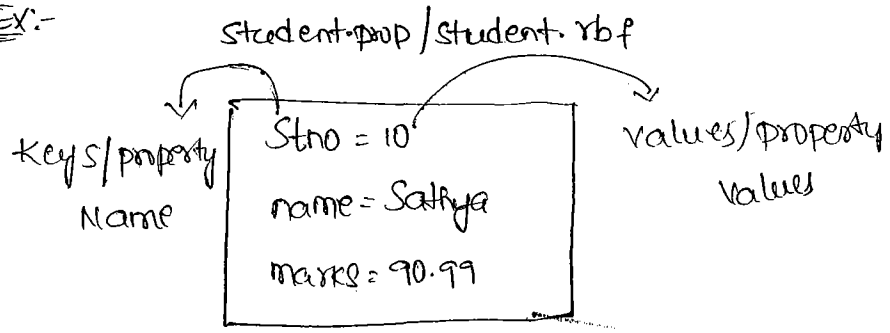
To overcome the above limitations of HashTable we use an another pre-defined class called java.util.Properties.

Property file :-

→ A property file is one of the textfile, it must be saved on some filename with an extension .prop (or) .rbf

→ A property file organizes the data in the form of (K,V)

Ex:-



→ Property files resides in Secondary memory

→ Here we obtain property values by passing property names

→ To read the data from property file, open the propery file in read mode with the help of a pre-defined class called FileInputStream

Ex:- Open the student.prop in read mode

```
FileInputStream fis=new FileInputStream("Student.prop");
```

Java.util.Properties :-

It is the subclass of HashTable, the advantages of properties over HashTable are.

1. Properties class object allows us to read data from properties file.
2. Properties class object allows us to read environmental variable names & environmental variable values.
3. Properties class object allows us to develop flexible appls

Profile of java.util.Properties :-

Constructor:-

1. properties()

Instance Methods:-

2. public void load(FileInputStream)
3. public String getProperty(String)

→ Constructor:1 is used for creating an object of properties class

Ex:- Properties p=new Properties();

→ Method:2 is used for transferring/ loading the content of properties file from secondary memory into the object of properties class by opening the property file in read mode.

Ex:- FileInputStream fis=new FileInputStream("Student.prop");
p.load(fis);

→ <u>Method :</u> is used for obtaining property value by passing property name.

P → main memory

Stno = 100
name = Sathya
marks = 99.90

Ex:- String stno = p.getProperty("stno");

String name = p.getProperty("name");

String marks = p.getProperty("marks");

21/9/15

File Programming (Stream handling)

→ In the context of file programming we've two types appll. They are

1. Non persistant Applications
2. Persistant Applications

→ In non-persistant appll developement the result of the program stored in main memory of the computer.

→ WKT the data present in main memory is temparary.

Ex:- All our previous programs comes under non-persistant applications.

→ In persistant appll developement the result of the program must be stored permanently. (Known as data persistancy)

→ We can achieve the data persistancy in two ways. They are

1. By using files
2. By using DB S/w

→ If we store the data permanently in the form of files then any un authorized user can modify the data of the file becoz files concept of any language is unable to provide security in the form of user name & password.

→ If we store the data in the form of DB S/w's then we can't have any unauthorized modifications becoz most of the DB S/w Products provides security in the form of user name & Password.

→ As a part of JSE we are achieving the data persistancy in the form of files by importing a pre-defined package called Java.io.*.

Def" of file :- (Flat file)

A Collection of record is called file / Flat file

Ex :-

Student. data

1	Sathya	99.99	JNTU	← Record1
2	Ram	86.80	OS	← Record2
3	Ravi	70.86	JNTU	← Record3

Files alway resides in secondary memory. If we want to store anything in secondary memory then it should be stored in the form of file name.

Programatically, in Java every Object data will become record in the file & record of a file of Secondary memory will become an object in main memory.

Def" of Record :-

Collection of interrelated fields values.

Ex :-

(10, Sathya, 99.99, JNTU)

→ Every value of record is called data member in the class/object

Def" of Stream :-

The flow of data btw main memory & Second memory is known as Stream.

Types of Streams :-

WKT the flow of data btw main memory & Secondary memory is called stream. Based on the flow of data btw main memory & Secondary memory we've two types of streams. They are

1. Byte Streams 2. Character Streams

Operations on files:-

On files we can perform 2 types of operations they are

1. write / output stream
2. Read / Input Stream

Write operation:-

→ The purpose of write operation is transfering temporary data from main memory into file of secondary memory

→ To perform write operation we must use the following steps.

a. Choose the file name

b. Open the filename in write mode

c. Perform cycle of write operations

→ While we are performing write operations we get the following consequences.

a. We are attempting to transfer the temparory data of main memory into the file of secondary memory where there is not sufficient memory space.

b. Trying to write the temparory data into the Read only file. Due

Due to these 2 consequences we get a pre-defined exception called java.io.IOException.

Read operation:-

Read operation is used for transferring data of file of secondary memory into the object of main memory.

→ To perform read operation we follow the following steps.

(a) Choose the filename

(b) Open the filename in read mode

(c) Perform the cycle of read operations

→ While we are performing read operations we get the following consequences

(a) We are attempting to open a file & it is not existing in secondary memory. Due to this we get Java.io.FileNotFoundException

(b) Trying to read the data from corrupted file. Due to this we get a predefined exception called Java.io.IOException

Byte streams :-

→ Byte streams transfer byte by byte data btw main memory and secondary memory.

→ Byte streams contains two categories of classes. They are

(a) Some of the category of classes participates in write operation & they will transfer one byte at a time from main memory to file of secondary memory.

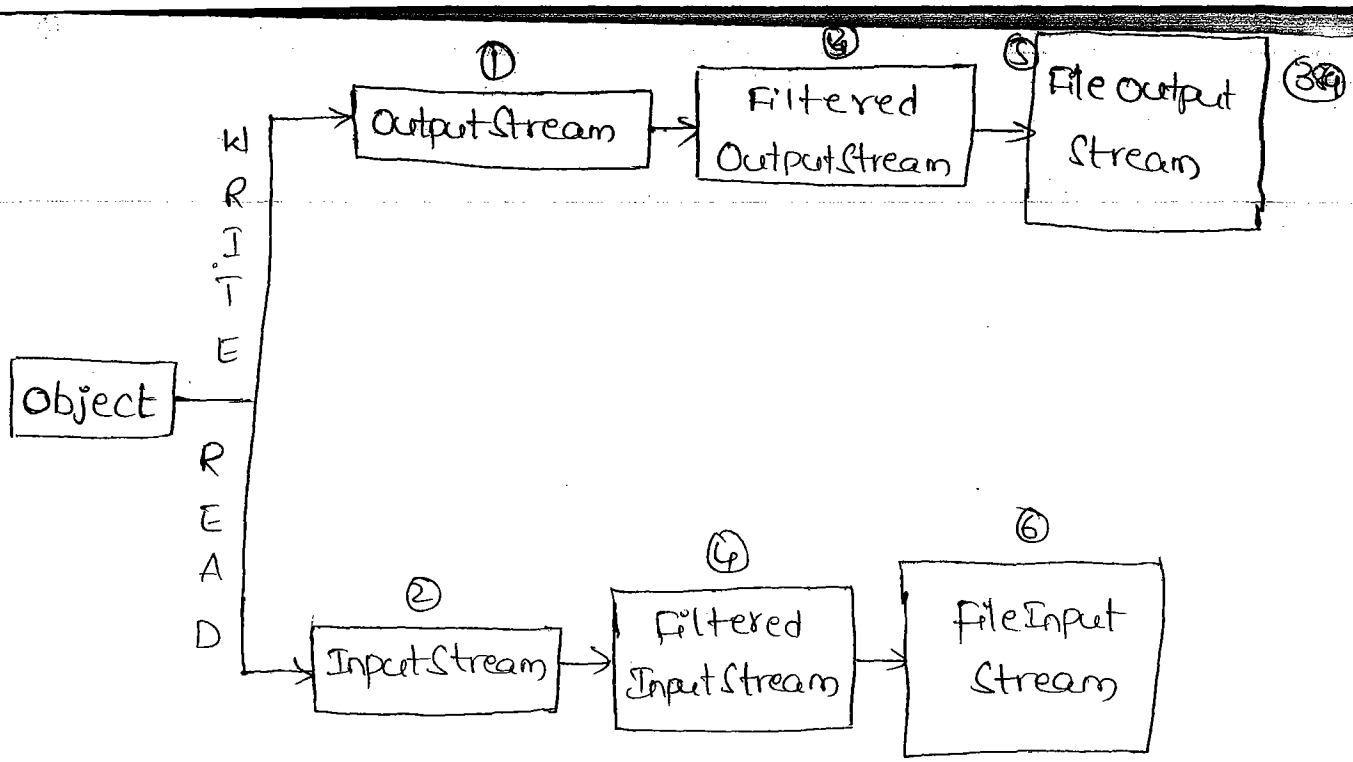
(b) Some of the category of classes participates in read operation & it can transfer one byte at a time from file of secondary memory into main memory.

→ These two category of classes present in Java.io.*.

Byte streams hierarchy chart :-

Hierarchy chart always makes us to understand how the features are inherited from top most base class to intermediate base classes & IBC to bottom most derived classes.

Hierarchy chart gives list of classes present in



In the above hierarchy chart the classes ① ② ③ ④ are called pre defined abstract classes in Byte Streams. Since, they are abstract we can't create their object directly.

The classes ⑤ & ⑥ are the bottom most concrete classes in Byte Streams. Since, these are concrete we can create their objects directly & we use in our application development.

FileOutputStream :-

The purpose of FileOutputStream class is to open the file in write mode. In otherwords, opening the file name in write mode is nthg but creating an object of FileOutputStream class.

Profile of Java.io.FileOutputStream:-

Constructors:-

1. FileOutputStream(String)
2. FileOutputStream(String, boolean)

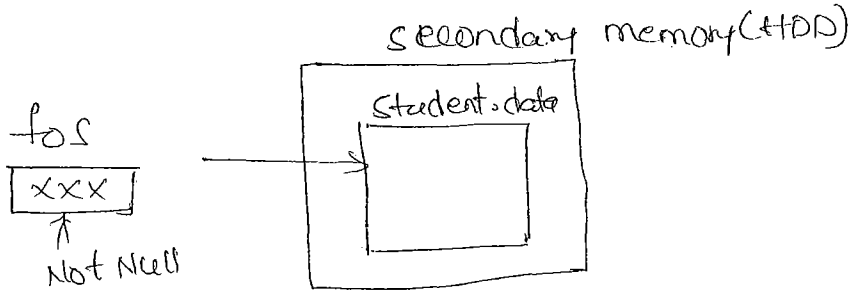
Instance methods:-

3. Public void writeXxx(xxx);
4. Public void writeLine(String);
5. public void write(xxx);
6. Public void close();

229 constructors ①&② are used for opening either the new file in write mode (or) existing file in write mode with appending approach (boolean = true) (or) opening the existing file in write mode with overlapping approach (boolean = false)

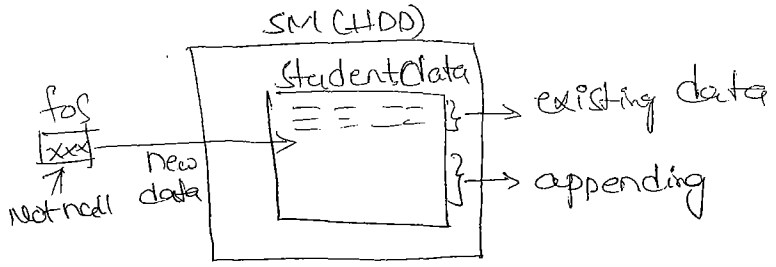
Ex1:- Open a new file student.data in write mode.

```
FileOutputStream fos = new FileOutputStream("Student.data");
FileOutputStream fos = new FileOutputStream("Student.data", false);
FileOutputStream foc = new FileOutputStream("Student.data", true);
```



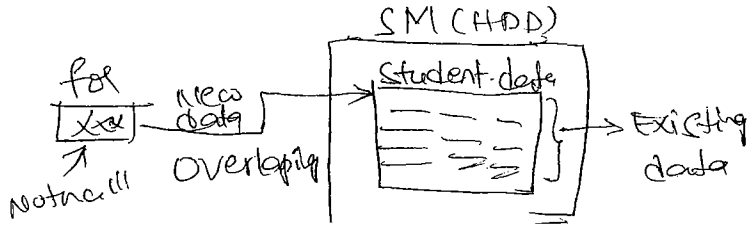
Ex2:- open the existing file student.data with appending approach.

```
FileOutputStream foc = new FileOutputStream("Student.data", true);
```



Ex3:- open the existing file student.data with overlapping approach.

```
FileOutputStream fos = new FileOutputStream("Student.data");
FileOutputStream fos = new FileOutputStream("Student.data", false);
```



As & when write operation is in progress fos is advancing to next line

→ If there is not sufficient memory space in secondary memory to open the file in write mode (or) the file which we are opening the write mode is belongs to read only file then it throws two cases

1. fos contains null & throws IOException

→ In method:3 xxx represents any fundamental type data. Method:3 is used for writing fundamental type data to the file.

→ Method:4 is used for writing any type of data in the form of String type

→ In Method:5 xxx represents any fundamental type & String type data Method:5 is one of the overloaded method & it is used for writing fundamental & String type data to the file.

→ Method:6 is used to close the file.

* The following code segment allows to close the fill which was opened in write mode.

```
finally
{
    try
    {
        if(fos!=null)
        {
            fos.close();
        }
    } catch(Exception e)
    {
        sep(e);
    }
}
```

FileInputStream :-

The purpose of this class is to open the file in read mode. In otherwords opening the file in read mode is nthg but creating an object of FileInputStream class.

Profile Java.io.FileInputStream :-

Constructor :-

1. FileInputStream (String) throws FileNotFoundException, IOException

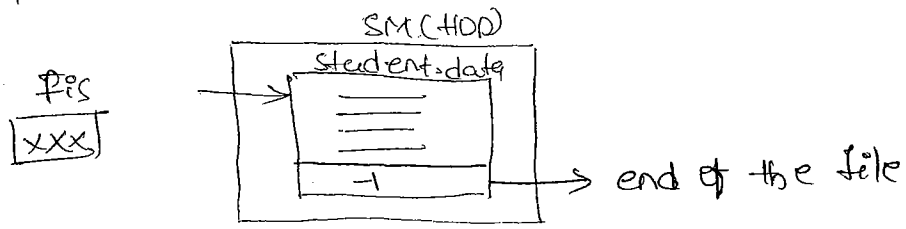
Instance method :-

2. Public xxx readXxx();
3. Public String readLine();
4. Public int read();
5. Public void close();

Constructor :1 is used for opening the file in read mode. If the specified file is not existing then we get FileNotFoundException & if the specified file is a corrupted file then we get IOException

Ex :- Open Student.data in read mode

FileInputStream fis = new FileInputStream("Student.data");



When reading operation is in progress, fis is advancing to next lines in the file until end of file is taking place.

In Java programming end of the file is indicated by -1

If Student.data doesn't exist & if it is a corrupted file then fis contains null & throws FileNotFoundException & IOException.

→ In Method:2 xxx represents any fundamental type. Method:2 is used for reading fundamental type data from the file.

→ Method:3 is used for reading string data from the file

→ Method:4 is used for reading any type of data until end of file is taking place

→ Method:5 is used for closing the file which is opened in read mode.

Eg:- The following code segment makes us to close the file.

```
finally
{
  try
  {
    if(fis != null)
    {
      fis.close();
    }
  } Catch (Exception e)
  {
    Sep(e);
  }
}
```

* Write the java programs for performing the following.

(a) write 1 to 10 numbers to the file.

(b) Read 1 to 10 numbers from the file.

```
//Filewrite.java
import java.util.Scanner;
import java.io.*;
class Filewrite
{
  Public static void main(String [] args)
  {
```

```
FileOutputStream fos=null;
try
{
//Step:-1
  Scanner s=new Scanner(System.in);
  Sop("Enter the file name to write 1-10 nos");
  String fname=s.nextline();
//Step:-2
  fos=new FileOutputStream(fname,true);
//step:-3
  for(int i=1;i<=10;i++)
  {
    fos.write(i);
  } //for
  sop("data written to the file-verify");
} catch (IOException ioe)
{
  sep("unable to open the file in write mode");
}
catch(Exception e)
{
  sep(e);
}
// close the file
finally
{
  try
  {
```

```
    {
      fos.close();
      Sop("file closed");
    }
  }catch(Exception e)
  {
    sep(e);
  }
 }// finally
}// main()
}// Filewrite-class
```

b) Read

```
// FileRead.java
import java.util.Scanner;
import java.io.*;
class FileRead
{
  Public static void main(String [] args)
  {
    FileInputStream fis=null;
   try
   {
 // Step :1
    Scanner s=new Scanner(System.in);
    SOP("Enter the file name to read 1-10 nos");
    String fname=s.nextline();
// Step 2
    fis=new FileInputStream(fname);
```

```
//Step:3
  int i;
   while((i=fis.read())!=-1)
    {
      sop(i);
    }
  } catch(FileNotFoundException fe)
  {
   sepc("file does not exists");
  }
  catch(IOException ie)
  {
   sepc("unable to read from courrpted file");
  }
 Catch(Exception e)
  {
   cep(e);
  }
// Close the file
 finally
  {
   try
   {
     if(fis!=null)
     {
       fis.close();
       sop("file closed");
     }
   } catch(Exception e)
```

```
    {
      sep(e);
    }
  } //finally
} //main()
} // FileRead - class
```

23/9 *WAJP which will copy the content of one file into another file. (or) Implement copy cmd of DOS

```
// Sathyacopy.java
import java.io.*;
Class Sathyacopy
 {
   Public static void main (String [] args)
   {
     if(args.length!=2)
     {
       SOP("Invalid syntax");
     }
     else
     {
       FileInputStream fis=null;
       FileOutputStream fos=null;
       try
       {
         String sfile=args[0];
         String dfile=args[1];
         fis=new FileInputStream(sfile);
         fos=new FileInputStream(dfile,true);
       //copy process
         int i;
```

```
do
{
    i=fis.read();
    Char ch=(Char)i; // type casting
    fos.write(ch);
  } while(i!=-1);
  .sop("\n" +sfile+ "content is copied into "+ dfile);
} catch(FileNotFoundException fe)
{
    sep("file does not exists");
}
 Catch(IOExeeption ioe)
{
  sepc" unable to read from corrupted file");
}
Catch(Exeeption e)
{
    sep(e);
}
// close the sfile & dfile
finally
{
    try
    {
      if(fis!=null)
      {
          fis.close();
          Sop("source file closed");
      }
      if(fos!=null
```

```
    {
      fos.close();
      sop(" Destination file closed);
    }
  } Catch( Exception e)
  {
    sep(e);
  }
 }// finally
 }// else
}// main()
}// SathyaCopy
```

* WAJP which will display the content of the specified file (Implement type cmd of DOS)

```
//SathyType.java
import java.util.*;
class Sathyatype
{
  public static void main(String [] args)
  {
    if(args.length != 1)
    {
      sop("Invalid syntax");
    }
    else
    {
      FileInputStream fis=null;
      try
      {
      String sfile=args[0];
      fis=new FileInputStream(sfile);
```

```
//type process
    int i;
    while((i=fis.read())!=-1)
     {
       char ch=(char)i; //type casting
       System.out.print(ch);
     }//while
  }catch(FileNotFoundException fe)
  {
    System.out.println("file does not exist");
  }
  catch(IOException ioe)
  {
   System.out.println("unable to read from corrupted file");
  }
  catch(Exception e)
  {
   System.out.print(e)
  }
//close the file
 finally
 {
  try
  {
    if(fis!=null)
     {
       fis.close();
       System.out.println("source file closed");
     }
```

```
} catch (Exception e)
{
    sep(e);
}
} //finally
} // else
} //main()
} // Sathya Copy Type.
```

Serialization and De-Serialization

Serialization:-

Let us assume there exists an object whose size is n-bytes. If we decided to transfer the object content into the file of secondary memory by using ordinary streams (Byte stream/Char Stream) then they takes either n-write operations (Byte streams) (or) n/2-write operations (Character streams). this is one of the time consuming process.

To eliminate these redundant write operations we use an other concept called "Serialization". In otherwords the advantage of Serialization concept is that eliminating the redundant write operations & with single write operations the entire object content stored in the file of secondary memory as a record.

Defn of Serialization:-

The process of transfering/saving the entire content of the Object into the file of secondary memory by performing single write operation is known as "Serialization".

Serialization concept participates in write operation.

De-Serialization:-

Let us assume there exists a record whose size is n-bytes resides in a file of secondary memory. If we want to transfer the entire record content from the file of secondary memory with ordinary streams then they takes either n-read operations (Byte streams) (or) n/2 read operations (character streams). This process leads to time consuming process.

To overcome/eliminate these redundant read operations we use another concept called "De-serialization". In otherwords the advantage of De-serialization concept is that eliminating redundant read operations & with single read operation the entire content of record of a file will be transfered into the object of main memory.

Defn of De-serialization:-

The process of transfering the entire record of a file of secondary memory into the object of main memory by performing single read operation is known as "Deserialization"

De-serialization concept always participates in read operation

Hence, the advantage of serialization & Deserialization concept is that eliminating redundant write & read operations. respectively.

No. of. Phases required for developing serialization and Deserialization applications:-

To develope serialization & Deserialization appls we must learn 3 phases. They are.

1. Development serializable sub class
2. Development of serialization process
3. Devolopment of deserialization process

Phase:1 Development of Serializable sub class

If any class object want to participate in Serialization Process then the corresponding class must implements a predefined called Java.io.Serializable. It is one of the marker/tagged intf.

Def of marker interface:-

Marker interface does not contain any methods explicitly (implicitly it contain methods & they are JVM dependable overridden methods) & provides run time behaviour to its sub class.

Ex:- Java.io.Serializable
Java.lang.Cloneable
Java.rmi.Remote
Javax.Servalent.SingleThreadModel etc

Steps:-

Step:1 Choose an appropriate package for placing serializable subclass for common access & ensure the package stmt must 1st executable stmt

Step:2 Choose an appropriate user defined class & whose modifier must be public

Step:3 Whatever the class is selecting Step:2 must implements Java.io.Seriable & Step:2 Class is called Serializable sub class

Step:4 Choose set of data members in serializable sub class

Step:5 Define set of setXxxx(-) for each & every data member of serializable sub class for the purpose of setting the values to serializable sub class object (such type of methods are called Mutators)

Step:6 Define set of getXxxx() for each & every data member of serializable sub class for the purpose of getting the data from serializable sub class object (such type of methods are called Inspectors)

Step:7 Whatever the serializable sub class we are placing in the package that class must be given as the file name with an extension .java.

Ex:- Develop a sericlizable sub class for student.

```
// Student.java
package sp;
Public class student implements java.io.Serializable
{
  int stno;
  String name;
  float marks;
Public void setStno(int stno)
 {
   this.stno = stno;
 }
Public void setName(String name)
 {
  this.name=name;
 }
Public void setMarks(float marks)
 {
  this.marks = marks;
 }
Public int getStno()
 {
   return stno;
 }
Public String getName()
 {
  return name;
 }
Public float getMarks()
 {
  return marks;
 }
```

Compile the above program

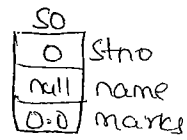
javac -d. Student.java

Hence, an object of Serializable sub class is an object of java.lang.Object.

Phase:2 Development of Serialization process

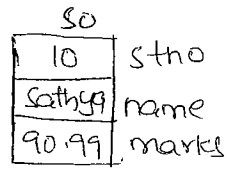
Step:1 create an object of Serializable sub class

Ex:- Student so = new Student();



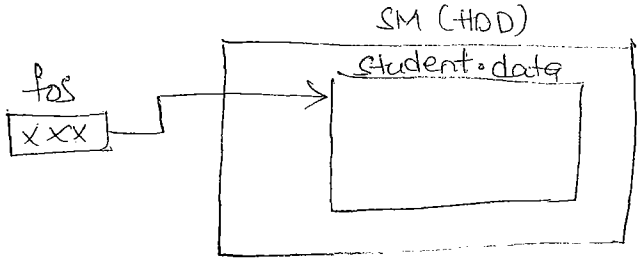
Step:2 Set the values to the Serializable sub class object

Ex:- so.setStno(10);
so.setName(sathya);
so.setMarks(90.99f);



Step:3 Choose the file name & open it into write mode

Ex:- FileOutputStream fos = new FileOutputStream("Student.data");



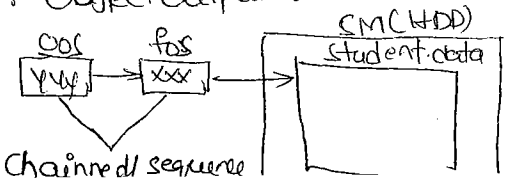
Step:4 create an object of ObjectOutputStream (becoz fos can't write entire object data to the file)

java.io.ObjectOutputStream

a. ObjectOutputStream(FileOutputStream) — OPC

b. public void writeObject(Object)

Ex:- ObjectOutputStream oos = new ObjectOutputStream(fos);



This stmt provides an environment to write serializable sub class object data to the file.

Note:- If one stream object is pointing to another stream object & if both of them belongs to same category then they are called chainned/sequence streams.

Step:5 Transfer/write the entire content of serializable sub class object into file of secondary memory.

Ex:- oos.write(so);

SOP("Student data saved successfully in the file");

Step:6 Close the file which was opened in write mode

Ex:- oos.close();

fos.close();

24/9/15

* WAJP which illustrate the concept of serialization process (save student object data into the file)

```
// Serp.java
import sp.Student;
import java.util.Scanner;
import java.io.*;
Class Serp
{
  public static void main(String [] args)
  {
    try
    {
    //Step-1
      Student so=new Student();
    // Step-2
      Scanner s=new Scanner(System.in);
      System.out.Enter number.
```

```
int sno=Integer.ParseInt(s.nextLine());
sop("Enter student name:");
String name=s.nextLine();
sop("Enter student marks:");
float marks=Float.parseFloat(s.nextLine());
so.SetSno(sno);
so.SetName(name);
so.SetMarks(marks);
//Step-3
sop("Enter the file name to write student data:");
String fname=s.nextLine();
FloatOutputStream fos=new FileOutputStream(fname);
//Step-4
ObjectOutputStream oos=new ObjectOutputStream(fos);
//Step-5
oos.writeObject(so);
sop("student data saved into the file successfully");
//Step-6
oos.close();
fos.close();
}catch(IOException ioe)
{
sep("unable to open the file in write mode");
}
catch(Exception e)
{
sep(e);
}
}//main()
}//class
```

Phase:3 Development of Deserialization process

Step:1 Create an object of serializable subclass for holding deserialization data.

Step:2 Choose the file name & open it into read mode.

Step:3 Create an Object of ObjectInputStream(becoz FileInputStream class Object is unable to read entire record)

java.io.ObjectInputStream

a. ObjectInputStream (FileInputStream)

b. public void readObject( )

Ex:- ObjectInputStream ois = new ObjectInputStream (fis);

Step:4 Read the entire content of a record from file of secondary memory in to the object of main memory.

Object obj = ois.readObject();

100	Sathya	75.00

← Deserialized data

Step:5 Typecast an object of java.lang.Object into serializable subclass object.

Step:6 Obtain & display deserialized data

Step:7 Terminate/close the file which was opened in read mode.

* WAJP which illustrates the concept of deserialization (read student record from the file)

```
// Deserp.java
import sp.Student;
import java.io.*;
import java.util.Scanner;
Class Deserp
{
```

```
public static void main(String [] args)
{
 try
  {
 // Step-1
     Student so=new Student();
 // Step-2
     Scanner s=new Scanner(System.in);
     sop("Enter the filename to read Student data:");
     String fname=s.nextLine();
     FileInputStream fis=new FileInputStream(fname);
 //Step-3
     ObjectInputStream ois=new ObjectInputStream(fis);

 // Step-4
     Object obj=ois.readObject();
 // Step-5
     so=(Student)obj; // object typecasting
 //Step-6
     sop("Student record details");
     sop("------------------------");
     sop("Student no = "+so.getStno());
     sop("Student name=" +so.getName());
     sop("Student Marks=" +so.getMarks());
     sop("------------------------");
 // Step-7
     ois.close();
     fis.close();
  } catch(FileNotFoundException fe)
  {
     sep("File does not exists");
  }
```

```
catch (Exception e)
 {
   sep(e);
 }
} //main()
} //DSerp --- class
```

Types of serializations:-

We have 2 types of serializations. They are

1. complete serialization
2. selective serialization

In complete serialization all the data members of serializable subclass participates in serialization process.

Ex:- Student.java (See previous pages)

Here an object of ~~subclass~~ Student class participates in complete serialization process.

In selective serialization selected data members of serializable subclass parcipates in serialization.

If we don't want to make any data member of serializable sub class to participates in serialization process then that data member declaration must be made as transint.

Hence transint variables never participates in serialization process

Ex:- class Bank implements java.io.Serializable

```
{
  transint int tkno;
  String cname;
  int acno;
  float bal;
}
```

Here an object of Bank class particates in selective serialization

→ An object of subclass of serializable subclass also participate in serialization & it is known as Automatic serialization.

Ex:-

```
Class AxisBank extends Bank
{
    ========
}
```

AxisBank ↑ subclass of serializable sub class

Bank ↑ ser. subclass

An object of AxisBank participates in Automatic serialization.

Character Streams:-

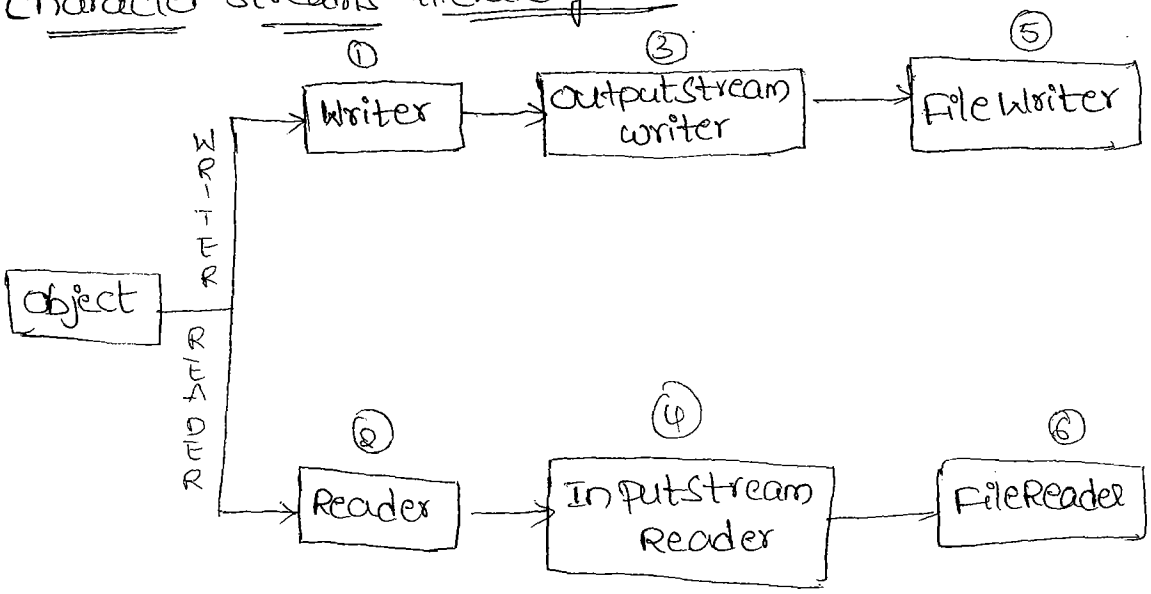
Character streams transfers 2 bytes at a time from main memory to secondary memory. Character streams are more effective than Byte streams.

→ Character streams contains 2 categories of classes. They are

(a) Some of the category of classes participates in write operation and they will transfer 2 bytes from main memory to file of secondary memory.

(b) Some other category of classes participates in read operation and they will transfer 2 bytes at a time from file of secondary memory to main memory & these two categories of classes present in java.io.*.

Character Streams Hierarchy chart:-



Here 1,2,3 & 4 classes are pre-defined abstract classes in character streams & there is no direct usage of these classes in our appll

The classes 5&6 are the bottom most concrete sub classes in character streams & we use in our appll development.

* The functionality of FileWriter class is more (or) less similar to FileOutputStream class but FileWriter class object writes 2 bytes & FOS write one byte (methods & constructors same)
* The functionality of FileReader is similar to FileInputStream class but FileReader class object reads 2 bytes & FIS reads one byte (methods & constructors are same)

Q:- How do you clone an object.

cloning an object is nthg but duplicating the object

Ex:-

```
// CloneDemo1.java
class Emp implements cloneable
{
  void disp()
  {
    sop ("emp-disp");
  }

 Public  Emp clone()
 {          ↑ co-variant data type
  try
  {
    return((Emp)super.clone());
  }catch(cloneNotSupportedException ce)
  {
    sep("cloning is not possible");
  }
  return this;
 }
```

```
class CloneDemo1
{
  Public static void main(String [] args)
  {
    Emp oo=new Emp();
    Emp co=oo.clone();
     oo.disp();
     co.disp();
  }//main
}//CloneDemo1.
```

Def^n of co-varient datatype:-

If we substitute subtype in place of supertype at the time of overriding the method then subtype of supertype is called co-varient data type.

Ex:- The original return type of clone() is java.lang.Object. under the time of overriding this clone() in the context of Emp class we can write sub type Emp in place of java.lang.Object

Ex:-

```
//CVD.java
interface Names
{
  Object getName();
}
class UrName implements Names
{
  Public String getName()
          └ co-varient data type
  {
    return "Sathya";
  }
}
class CVD
{
  Public static void main(String [] args)
```

```
{
    UrName U0=new UrName();
    String name=U0.getName();
    Sop(name);
    Names n0=new UrName();
    String name1=(String)n0.getName(); // type casted
    Sop(name1);
  }
}
```

20/1/15

String handling

A character is an identifier which will be enclosed within single quotes.

Valid Ex:- 'A', '$', '9' ... etc

In-valid Ex:- 'Java', 'Sathya Tech'

String:- collection/sequence of character enclosed within double quotes is called String.

Ex:- "Java", "Sathya Tech" etc

"A", "$", "9" etc

By default every character can be treated as String & by default String can't be treated as character. (Programatically, we can convert character String into character.)

Inorder to store string data & to perform various operations on String data. We've 4 types pre-defined classes.

1. Java.lang.String
2. Java.lang.StringBuffer
3. Java.lang.StringBuilder
4. Java.util.StringTokenizer

Q:- What are the differences btw String & StringBuffer.

String	StringBuffer
1. The double quoted data is by default treated as an object of String class.	1. Double quoted data will not be treated by default as an object of StringBuffer.
Ex:- String s="JAVA"	Ex:- StringBuffer sb="SATHYA" — error StringBuffer sb=new StringBuffer("SATHYA")

2. When we create an object of String class, we never get any additional memory space

2. When we create an object of StringBuffer, we get 16 additional characters memory space.

3. When we do any modifications on String data, the existing memory space is destroyed, new memory space is created & modified values are placed i.e., modifications are not permitted in the same space.

3. When we do the modifications on StringBuffer, they can be performed in the same space.

4. String class object is also known as immutable object

4. StringBuffer class object is also known as Mutable Object

* Similarities btw String & StringBuffer:-

1. These two classes are public final
2. They never participates in inheritance (is-A relationship is not possible) use these classes by following has-A & uses-A relationships.
3. We can't override methods of these classes

1. By using java.lang.String :-

We can store the String data in the main memory in two ways.

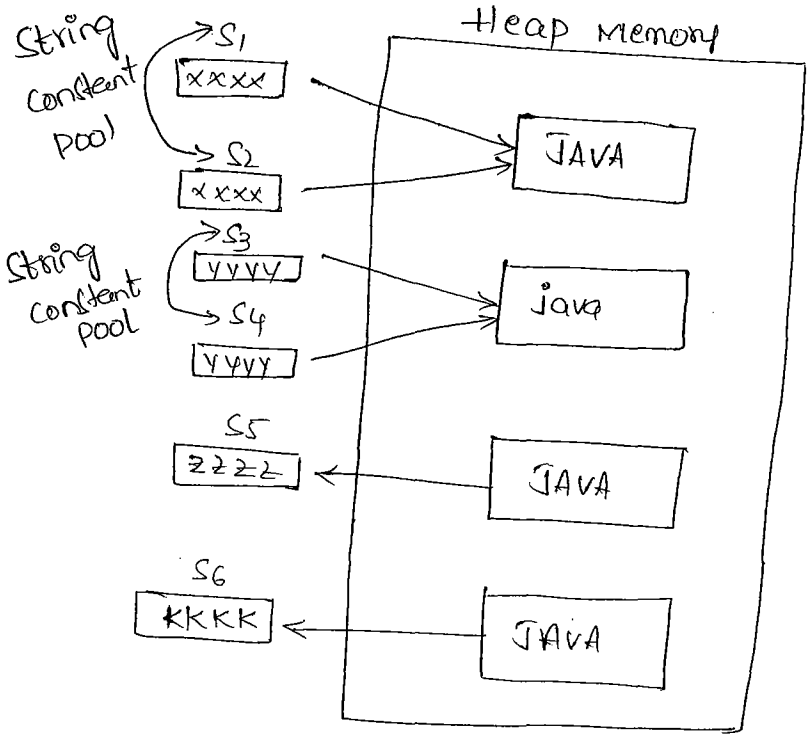
a. Directly without new operator:-

Ex:- String S1 = "JAVA";
String S2 = "JAVA";
String S3 = "Java";
String S4 = "Java";

b. By using new operator:-

Ex:- String s5 = new String("JAVA");
String s6 = new String("JAVA");

For above two approaches the memory management diagram is shown below.



Profile of Java.lang.String :-

Constructors :-

1. String() :-

It is used to create empty String object.

Ex:- String s = new String();
sop(s); // empty

2. String(char[]) :-

It is used to convert array of characters into string.

Ex:- char ch[] = {'J', 'A', 'V', 'A'};
String s = new String(ch);
sop(s); // Java

3. String (String):-

* This constructor is used for converting one String object into another String object (or)

* Copying the content of one String object into another String object.

EX:-

```
String s1 = "JAVA";
String s2 = new String (s1);
    sop (s1); //JAVA
    sop (s2); //JAVA
```

4. String(StringBuffer):-

This constructor is used for converting StringBuffer object into String object.

EX:- StringBuffer sb = new StringBuffer("Java");

→ [Java 16-additional char memory space]

String s = new String(sb);

→ [JAVA]

In other words, we convert mutable object into immutable object.

5. String(StringBuilder):-

It is used to convert StringBuilder object into String class object.

4/10/15

Instance methods :-

1. Public int length();-

This method is used for finding the no. of chars present in string.

Ex:-

```
String s = "JAVA";
    Sop(s); //JAVA
int noc = s.length();
    Sop(noc); //4
```

Ex:-

```
int noc = "Sathya".length();
    Sop(noc); //6
```

Ex:-

```
String crs[] = {"CPP", "JAVA", "oracle"};
Sop("No. of. courses = "+crs.length); //3
    int noc = crs[1].length();
        Sop(noc); //4
```

NOTE:- "length" attribute to be applied on arrays "length()" to be applied on string objects but not to be applied on opposite.

2. Pubic char charAt(int):-

This method is used for finding a character by passing valid position otherwise we get a pre-defined un checked exception called Java.lang.StringIndexOutOfBoundsException.

Ex:-

```
String s = "JAVA";
Char ch = s.charAt(2);   // valid position
Sop(ch); //V
```

Char ch = s.charAt(20); Invalid position

throws → StringIndexOutOfBoundsException

3. Public char[] toCharArray():-

This method is used for converting string data into character array.

Ex:-

```
String s = "JAVA";
  sop(s); // JAVA
Char ch[] = s.toCharArray();
for(int i=0;i<ch.length;i++)
  {
    sop(ch[i]); // JAVA
  }
```

ch

J
A
V
A
~~\0~~

4. Public String trim():-

This method is used for eliminating leading & trailing spaces but not the spaces which are present btw the words

Ex:- String pwd = " JAVA PROG ";

leading spaces — trailing (or) yeilding spaces

```
int noc = pwd.length();
  sop(noc); // 15
String ts = pwd.trim();
int noc1 = ts.length();
  sop(noc1); // 9
```

5. Public String toUpperCase() :-

6. Public String toLowerCase() :-

The above methods are used for converting lower case data into upper case & vice versa. respectively.

EX:-
```
String os = "Java";
String ls = os.toLowerCase();
    sop(ls); // java
String us = os.toUpperCase();
    sop(us); // JAVA
```

7. Public String substring(int) :-

This method is used for obtaining a part of the string by passing the position where to start & it takes up to last characters.

8. Public String substring(int s, int e-1) :-

This method is used for obtaining the range of the characters by specifing the positions where to start(s) & where to end (e-1) ie, we get s to e-1 characters as a substring

EX:-
```
String s = "JAVA PROG";
String ss1 = s.substring(5);
    sop(ss1); // PROG
String ss2 = s.substring(2,7); // 2 to 7-1
    sop(ss2); // VA PR
```

9. Public boolean equals(String):-

This method is used for comparing two string contents. This method returns true provided the content of source string & destination string must be same in case & meaning. Otherwise it returns false.

10. Public boolean equalsIgnoreCase(String):-

This method returns true provided the contents of source string & destination string are same in meaning without considering their cases. This method returns false the contents of source string & destination string are different in meaning.

Ex:

```
String S1 = "JAVA";
String S2 = "java";

boolean b = S1.equals(S2);
        Sop(b); // false
        b = S1.equals("JAVA");
        Sop(b); // True.

        b = S1.equalsIgnoreCase(S2);
        Sop(b); // true
        b = S1.equalsIgnoreCase("KAVA");
        Sop(b); // false.
```

Q:- What is the difference btw equals() & "==" operator

equals() is always used for comparision of contents of two String objects whereas "==" operator is used for comparing references of two objects

Ex:-

```
String S1 = "JAVA";
String S2 = "JAVA";
String S3 = new String("JAVA");
String S4 = new String(S2);
if( S1 == S2)
{
  Sop(" S1 & S2 contains same ref"); // valid executed
}
else
{
  SOP( "S1 & S2 contains different ref");
}
if( S2 == S4)
{
  SOP("S2 & S4 contains same ref");
}
else
{
  SOP("S2 & S4 contains Diff ref"); //valid
}
```

```
if(S1.equals(S2))
{
  sop("Same Content"); //valid
}
else
{
  sop("Diff: content");
}
```

11. Public boolean startsWith(String) :-

12. Public boolean endsWith(String) :-

The above methods are used for pattern matching. These methods returns true provided the destination string starts with/ endswith source string respectively otherwise they returns false.

EX:- String s="JAVA is an OPP Lang";

```
boolean b = s.startsWith("JAVA");
        sop(b); // true
        b=s.startsWith("Lang");
         sop(b); // false
        b=s.endsWith("JAVA");
         sop(b); // false
        b=s.endsWith("Lang");
          sop(b); //true
```

13. Public String concat(String):-

This method is used for concatinating contents of two strings but not more than two string contents. To do more than two strings content concatination we use "+" operator

EX:-

```
String s1 = "Sathya";
String s2 = "Tech";
String s3 = s1.concat(s2);
        sop(s1); // sathya
        sop(s2); // Tech
        sop(s3); // Sathya Tech
```

EX:-

```
String s3 = s1 + s2 + "Hyd";
        sop(s3); // Sathya Tech Hyd
```

14. Public int compareTO(String):-

This method returns either 0 (zero) (or) +ve (or) -ve value

EX:-

```
String s1 = "JAVA";
String s2 = "Java";
int cv = s1.compareTO("JAVA");
    sop(cv); // 0 ⇒ equals
    cv = s1.compareto(s2);
      sop(cv); // -ve ⇒ Destination String (s2) greater than source string(s1)
    cv = s1.compareto(ABC);
      sop(cv); // +ve → Source String(s1) is greater than Dest. String(ABC)
```

Static Methods:-

Public static String valueOf(Xxx):-

In this method Xxx represents any fundamental datatype, it is One of the overloaded static factory method. & it is used for converting fundamental type values into String type values.

Ex:- int a=10;

String iv=String.valueOf(a);

→ "10"

sop(iv); //10

float x=1.5f;

String fv=String.valueOf(x);

→ "1.5f"

sop(fv); // 1.5

//10

2. Java.lang.StringBuffer:-

It is one of the pre-defined public final class. & it never partic-pates in inheritance process. When we creat an object of StringBuffer we get 16 additional character memory space & whose object is known as mutable object.

To perform the operations on string data by using StringBuffer class we need learn its profile.

Profile java.lang.StringBuffer:-

Constructors:-

1. StringBuffer(char[])
2. StringBuffer(StringBuffer)
3. StringBuffer(String) → converting String cls object into StringBuffer
4. StringBuffer(StringBuilder)

→ Constructor:1 is used for converting array of characters into StringBuffer object.

EX:- char ch[] = { 'J', 'A', 'V', 'A' };

StringBuffer sb = new StringBuffer(ch);

→ JAVA 16 additional chars space

sop(sb); // JAVA

→ constructor:2 is used for converting one StringBuffer object into another StringBuffer object

EX:-

StringBuffer sb = new StringBuffer("JAVA");

StringBuffer sb1 = new StringBuffer(sb); // OPC

sop(sb); // JAVA

sop(sb1); // JAVA

→ constructor:3 is used for converting String class object data into StringBuffer object. In otherwords this constructor converts immuteable object into mutable object.

EX:- String s = "JAVA";

→ JAVA

sop(s); // JAVA

StringBuffer sb = new StringBuffer(s);

→ JAVA 16 additional chars

sop(s); // JAVA

sop(sb); // JAVA

→ constructor:4 is used for converting StringBuilder object into StringBuffer class

NOTE:- String constant pool exists only for String class objects becoz one can initialize the String class object without new operator whereas String constant pool will not exist for StringBuffer class objects becoz every [illegible] with new operator but not directly.

Instance Methods :-

1. Public int length()
2. Public int capacity()

The above methods are used for finding no. of characters & capacity (no. of chars + 16 - additional chars) of StringBuffer object.

EX :-

```
StringBuffer sb = new StringBuffer("JAVA");
```

→ JAVA 16 add'l chars

```
sop(" No. of chars = "+ sb.capacity());//4
sop(" cap of sb= "+ sb.capacity());//20 (4+16)
```

3. Public StringBuffer reverse()

This method is used for obtaining sentence reverse of StringBuffer object.

EX :-

```
StringBuffer sb= new StringBuffer("JAVA PROG");
StringBuffer sb1 = sb.reverse();
    sop(sb); //JAVA PROG
    sop(sb1); // GORP AVAJ
```

4. Public String toString()

This method is used for converting StringBuffer object (mutable) into String class object (immutable).

EX :-

```
StringBuffer sb= new StringBuffer("JAVA");
String s= sb.toString();
    sop(s); // JAVA
```

5. Public void trimToSize():-

This method is used for eliminating additional characters memory space & adjust the StringBuffer object memory ~~upto~~ only to its length.

Ex:-

```
StringBuffer sb=new StringBuffer("JAVA");
sop("cap of sb="+ sb.capacity());  //20
sb.trimToSize();
sop("cap of sb after trim="+sb.capacity());  //4
```

6. Public int codePointAt(char):-

This method is used for obtaining unique code value of a specific character which is present in the object of StringBuffer

Ex:-

```
StringBuffer sb=new StringBuffer("JAVA");
int uc = sb.codePointAt('A');
sop(uc);  //65
```

7. Public void setCharAt(int, char):-

This method is used for replacing the existing character with new character by specifying the position of existing character.

Ex:-

```
StringBuffer sb=new StringBuffer("KAVA");
sop(sb);  // KAVA
sb.setCharAt(0,'J');
sop(sb);  // JAVA
```

8. Public StringBuffer append(Object)
9. Public StringBuffer append(String)
10. Public StringBuffer append(StringBuffer)
11. Public StringBuffer append(xxx)// → Here xxx represents fundamental data types

The above methods are used for appending string data, StringBuffer data, fundamental data & any type of object data to the content of existing StringBuffer object data.

Ex:-

```
StringBuffer sb=new StringBuffer("JAVA);
StringBuffer sb1 = sb.append(" PROG");
sop(sb1); // JAVAPROG
StringBuffer sb2= sb1.append (10);
sop(sb2); // JAVAPROG10
```

12. Public StringBuffer delete(int, int):-

This method is used for removing range of characters by specifying the position where to start & where to end.

Ex:-

```
StringBuffer sb2 new StringBuffer("JAVA PROG");
sop(sb); //JAVAPROG
StringBuffer sb1 = sb.delete(4,7)
sop(sb1); // JAVA
StringBuffer sb2= sb1.deleteCharAt(3);
sop(sb2)= // JAV
```

13. Public StringBuffer deleteCharAt(int):-

This method is used for deleting a particular character by specifying its position

14. Public StringBuffer replace(int, int, String):-

This method is used for replacing range of characters with the specified new string.

Ex:-

```
StringBuffer sb = new StringBuffer("JAVAPROG");
sop(sb); // JAVAPROG
StringBuffer sb1 = sb.replace(4,7,"LANG");
sop(sb1); // JAVALANG
```

15. Public stringBuffer insert(int, string):-

This method is used for inserting a new string in the existing data of StringBuffer by specifying the existing position.

Ex:-

```
stringBuffer sb1 = new stringBuffer("JAVALANG");
sop(sb1); JAVALANG
StringBuffer sb2 = sb1.insert(4,"PROG");
sop(sb2); // JAVAPROG LANG
```

3. Java.Lang.stringBuilder:-

The functionality of StringBuilder is exactly similar to StringBuffer but stringBuilder methods are non-synchronized gives non thread safety results & recomended to use in server side appls where as stringBuffer class Object & whose methods are synchronized & gives thread safety results & recomended to use in client side appls. [constructs of stringBuilder are similar to constructors of StringBuffer, methods of stringBuilder are similar to stringBuffer methods without synchronized.]

18/10

4. Java.util.StringTokenizer:-

→ It is one of the predefined class present in Java.util.*

→ The purpose of StringTokenizer is split the given string into different Substring based on delimeter

→ some of the delimeters are . , ; :

→ StringTokenizer always seperates the given string by considering delimeter. Delimeter seperated values are called Tokens

Profile of Java.util.StringTokenizer :-

Constructors:-

1. StringTokenizer(String)
2. StringTokenizer(String, String)

Instance Method:-

3. Public int countTokens()
4. Public boolean hasMoreTokens()
5. Public String nextToken()

Constructor:1 is used for creating an object of StringTokenizer without specifying delimeter.

Constructor:2 is used for creating StringTokenizer by specifying delimeter.

Method:3 is used for counting the no. of tokens which are seperated by delimeter.

Method:4 returns true provided an object of StringTokenizer having more no. of tokens otherwise it returns false

Method:5 is used for obtaining next token w.r.t the object of StringTokenizer provided method:4 returns true.

* Write a Java program which illustrate the concept of StringTokenizer.

```
//StrTok.Java
import Java.util.StringTokenizer;
class StrTok
```

```
Public static void main(String[] args)
 {
    StringTokenizer st = new StringTokenizer("Java # is # an
        oop # lang", "#"); //Here '#' is called delimeter

    int not = st.countTokens();
    sop("no. of Tokens = "+not);
  while(st.hasMoreTokens())
  {
    String tok = st.nextToken();
    sop(tok);
  }
 }
}
```

Inner/nested classes:-

Def:- If we define a class Y inside the deffination of another class X then the class Y is called inner class & the class X is called outer class

The process of defining one class defination into another class defination is known as Containership.

We have 4 types of Inner classes

1. Static inner class
2. member inner class
3. local inner class
4. anonymous inner class

Static inner class:-

→ The purpose of static inner class is perfoming one type common operations by taking static features.

→ Just static classes are not there in Java but one can write inner static classes.

→ We can't refer outer class instance properties (data member of methods) from the context of inner static class.

→ programatically, in the static inner class we can take both static & Instance features (recomended to take static features).

Syntax for static inner class:-

```
class<outer class>
{
  ____
  static class<inner class>
  {
    ____
  } // inner class
} // outer class
```

* WAP which illustrate the concept of static inner classes.

```
// xy.Java
class x
{
  int a=10;
  static int b=20;
  void xyz();
  {
    sop("val of a of x = "+a);
    sop(" val of b of X = "+ b);
    sop(" val of c of Y = "+ new y().c);
    sop(" val of d of Y = " +y.d);
  }
  static class y
  {
    int c=30;
    static int d = 40;
```

```
void abc()
{
    // Sop("val of a from x="+a); Invalid
    Sop("val of b from x="+b);
    Sop("val of c from y="+c);
    Sop("val of d from y="+d);
}
} // y - inner class
} // x - outer class
Class xy
{
Public Static void main(String[] args)
{
    X o1=new x(); // creating an object of outer class
    o1.xyz();
    x.y o2=new x.y(); // creating an object of static inner class
    o2.abc();
}
}
```

Instance inner class:-

→ Instance inner classes contains only instance feature & it never contains static features.

→ The purpose of instance inner class is performing some repeated operation by containing instance features.

→ Instance inner class can be access all the features of outer class & outer class also access instance inner class feature

*WAJP which illustrate concept of instance inner class.

```
// PQ.Java
class P
{
  int a = 10;
  Static int b = 20;
  void abc()
  {
    sop("val of a of P="+a);
    sop("val of b of P="+b);
    sop("val of c of Q="+new Q().c);
  }
 Class Q  //instance class
 {
   int c=30;
  // Static int d=20; invalid
  // Static void pqr() { } -invalid
   void xyz()
   {
     sop(" val of a of P="+a);
     sop("val of b of P="+b);
     sop("val of c of Q="+c);
   } //xyz
 } // Q
} // P
class PQ
{
 Public Static void main( String [] args)
```

```
{
    P o1 = new P();
    o1.abc();
    P.Q o2 = new P().new Q();  // creating an object of Instance inner class
    o2.xyz();
  }
}
```

local inner class :-

→ If we define a class defination inside the method of another class then class defination present inside the method definition is called local inner class.

→ local inner class must be instance class but it should not belongs to public, protected, private & static.

→ local inner class method can be accessed within the containing method only but not in other method of outer class.

* WAJP which illustrate local inner class.

```
// LIC.Java
class Sathya
{
  void learnJava()
  {
    sop("learning java from Sathya");
    class Faculty
    {
      void teachJava()
      {
        sop("learning from xyz faculty");
      }
    } // Faculty - local inner class
```

```
Faculty f= new Faculty();
    f.teachJava();
  } // learnJava()
} // Sathya - Outer class
Class LIC
{
    public static void main(String[] args)
    {  Sathya s=new Sathya();
       s.learnJava();
    }
}
```

Anonymous inner class:-

→ An Anonymous inner class doesn't contain any explicit name given by the programmer

→ The purpose of Anonymous inner classes is to implement abstract method of abstract classes & interfaces.

→ For more explanation, for more example see inner interfaces in previous pages (Interface topics).