

# CORE JAVA

K.V. RAO

Sree kaaram digital xerox  
Gayatri Nagar,Behind  
Mythrivanam,  
Ameerpet. Ph:9908705527

**Published by :**



**Phno:9014213741**



**Phno:9908116001**

Q) What is JAVA?

Java is a Internet SW developed by Sun Micro Systems to develop distributed applications.

Q) What is Software?

Ans: A collection of programs, which will convert Imaginaries / dreams into reality.

Types of SW

1. System SW
2. Application SW
3. Internet SW (Java)

System SW: Is one whose ~~works~~ to

aims: → Development of functionalities of HW devices

→ Development of language compilers & Interpreters

→ Development of RTOS (Real Time operating systems)

There are 64 RTOS are available.

To develop these C and ALP are used.

System SW is used to manage the background of the system to implement the foreground.

- Application SW:

The converting of manual work into  
is known as Information sys.

(Automated)  
Computerized work

Aims:

→ To develop Info systems.

javap java.lang.Integer

java -sun -I party  
java -os -II party  
windows -OS -III party  
software -

### pen/pencil/paper drawbacks

- errors
- HB mistakes
- storing problem
- Retrieval
- poor service

There are 64 operating systems.

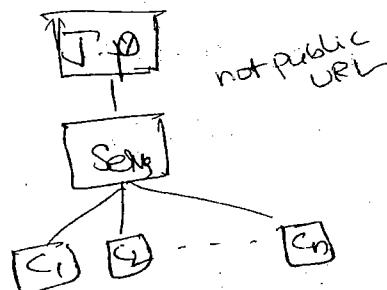
DOS OS can understand everything in MZARFA (MZ) format

UNIX OS can understand everything in Embedded Linking Format (ELF)

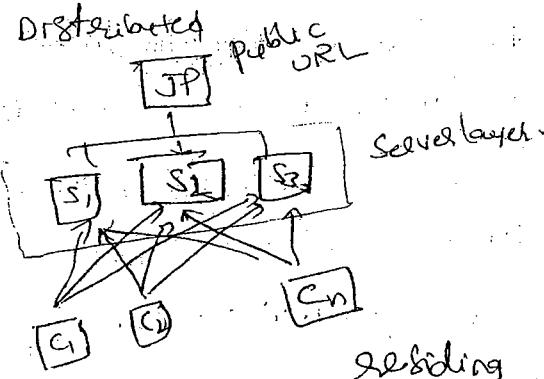
Networked Aim: sharing of data b/w the systems

Untrusted < collection of interconnected autonomous systems

→ trusted Centralized



not public URL



URI → the place where the project is located (physical)

URL → the place where the app is running (logical)

## Surroundings of Java

According to industry standards we have three types of softwares. They are

1. System software
2. Application software and
3. Internet software.

### System SW:

The basic aims of system sw are

- a. development of functionality of hardware devices
- b. development of language compilers & interpreters
- c. development of RTOS (Real Time operating systems)

Most of the CMH-5 level companies uses the following languages for development of system software.

- C and
- ALP (Assembly Language programming)

In general any kind of program which is running in the background to support foreground programs, these background programs are known as system level programs, foreground programs are known as userlevel applications.

### Application SW:

The aim of application sw is to develop information system (the process of converting manual work of the organization into automated work or electronic work).

Some of the organizations where we can develop the application software are

- a. Financial sector (Banks, LICs, Auditing companies, etc)
- b. Healthcare sectors (Hospitals, Diagnostic centers, etc)
- c. Transportation sectors (Bus Reservation sys, Train Reservation sys, Airlines)
- d. Super Market, etc.

In general if we are able to develop a project for the organization then that project will comes under application software.

To develop application software need two types of technologies

a. frontend | GUI | windows Technologies

- i. developer 2000 - Oracle Corp
- ii. Visual Basic - Microsoft
- iii. VB. Net - Microsoft
- iv. Awt / Applets / Swings - Sun Microsoft

When we develop any frontend application (i), (ii), (iii) technologies, those applications runs on Microsoft providing OS only but not on non Microsoft OS. These types of applications are known as platform dependent.

When we develop any GUI app / frontend application by using (iv) technologies, those applications runs on every OS so that these applications are called platform independent.

As on today most of the organizations prefers to develop

platform independent applications.

b. Backend Technologies (Database products)

- i. dBase, dbase II plus foxpro (outdated)
- ii. Oracle 10g - Oracle Corp
- SQL Server - MS
- DB2 - IBM
- MySQL - SUN
- Sybase - Redhat
- Paradox - IBM
- Ingres - Ingres Corp
- etc -

} Industry used

Note: In order to store the data permanently, we have two approaches in information technology. They are

- through files concept
- through database concept

Industries are not recommended to store the data permanently in the form of files because the data of the file can be manipulated by any unauthorized users. and more over files concept is not providing user names and passwords.

Industry is always recommended to store the data permanently in the form in the form of most popular database products. And more over the databases provides effective security to the data in the form of user names and passwords.

## Internet sw:

As a software engineer in the industry we developed two types of applications. They are

1. Standalone app.
2. Distributed application

A standalone application is one which is running in the context of local disk and whose results are not sharable.

Ex: All the applications of system sw and application sw comes under standalone applications.

Distributed applications are those which are running in the context of Browser / www and whose results are sharable across the globe.

Ex: mail.yahoo.com, gmail.com, etc.

In general all the websites comes under distributed applications.

## Internet sw:

The basic aim of internet sw is to develop distributed applications.

To develop a distributed application we are using the following languages / technologies

a. JAVA developed by Sun Microsystem INC, USA

b. .NET developed by Microsoft INC, USA

(Q) What is JAVA?

Ans: JAVA is an internet software developed by Sun Microsystem and it is used for developing the distributed applications.

## Architecture required for development of distributed app:

According to industry standards, distributed applications will be developed by a well known architecture called client-server architecture.

According to industry standards client server architecture having 3 categories. They are:

### 1. Two 2-Tier Architecture:

It contains

- Set of client programs

- Set of server programs

### 2. 3-Tier Architecture:

It contains

- Set of client programs

- Set of service programs

- Set of database s/w

### 3. n-tier architecture:

It contains

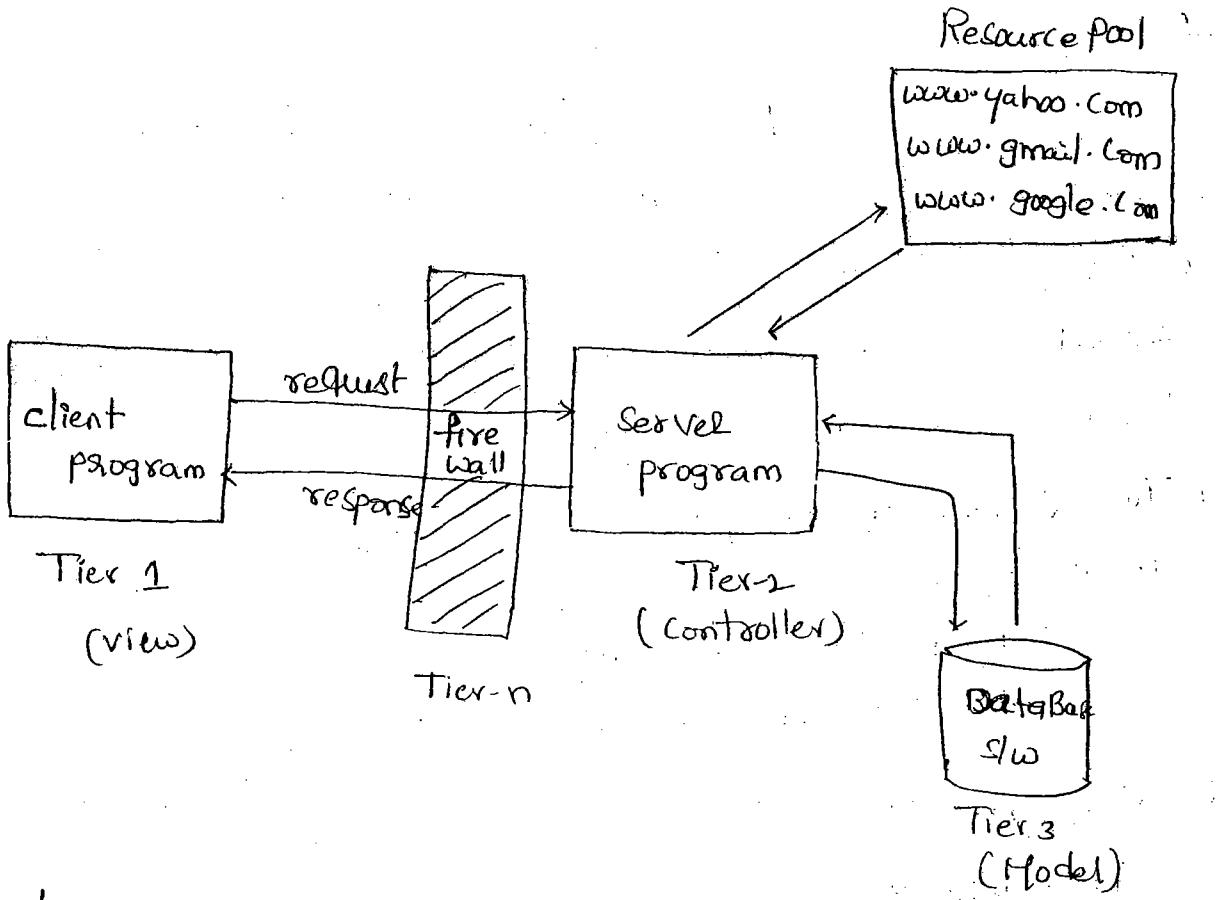
- Set of client programs

- set of firewall

- Set of server side

- Set of database s/w

The following diagram used gives diagrammatic view of client server architecture.



client: A client is the Java program which always makes a request to the server to get the services.

server: A server is also a program which always receives client request, process the client request and gives response back to clients concurrently.

The basic advantage of server software is that to get concurrent access.

The following table gives list of server SW names and their vendors

<u>Server SW Name</u>	<u>Server Vendor Name</u>
TomCat	Apache Jakarta SW foundation
webLogic	BEA Corp
webSphere	IBM
Oracle 10g	Oracle Corp
Pramathi	pramathi SW services, hyd
etc	

Firewall: A firewall is the security program which will validate whether the client request is virus free or virus effected.

Database SW: It is used for to store the data permanent with high effective services.

In industry all the database SW are known as backend technology.

Protocol: The set of rules, used for exchanging the data b/w Client and Server, application is known as protocol.

In the industry all the internet applications are making use of http

Resource pool: It is the place where the set of server programs are residing in the context of server SW.

History of Java:

When we are learning any programming language newly, it is highly recommended for the programmer to learn the history of the language.

We are learning Java language and whose history will be given in the following points.

1. Java is a programming language/technology used by the industry for development of client distributed applications by making use of client server architecture.
2. Java language developed at SUN Micro system in the year 1990 by a person called James Gosling and others.
3. What ever the trial version developed and released in the year 1990, it was named as "OAK", which is

the original name of Java and it has taken 18 months to develop. Scientifically the original name of OAK is one of the Tree name.

4. Originally Sun MicroSystem is one of the academic university (Stanford University Network). Sun developed rules for Java under the guidance of James Gosling and those rules are programmatically implemented / developed / defined by JavaSoft INC, USA, which is the software division of Sun Micro System only.

5. The slw OAK is solving some of the problems of the industry and some of the problems not solved, in other words, OAK Software is not completely solving industry problems. Hence the slw OAK revised by Sun Micro System in the year 1995 and released to the industry on the name of "Java" (Scientifically it is one of the coffee seed name).

6. The slw Java is available in the industry in three categories. They are

- a. J2SE (Java 2 Standard Edition)
- b. J2EE (Java 2 Enterprise Edition)
- c. J2ME (Java 2 Micro Mobile Edition)

J2SE: is used by the industry for development of client side applications.

J2EE: It is used for developing server side applications. To exchange the data b/w J2SE and J2EE applications we use a protocol called http.

J2ME: It is used for developing mobile / wireless applications by using a protocol called WAP (wireless access / app protocol)

Q) What are the differences b/w http and FTP.

### Http

- \* It is one of the TCP protocol
- \* http is one of the stateless protocol
- http is the one of the acknowledgement based oriented protocol.

### FTP

- It is one of the UDP protocol.
- It is one of the stateful protocol.

FTP is one of the non acknowledgement oriented protocol.

→ A stateless protocol is one which maintains an identity of a client for a period of time.

Ex: http

→ A stateful protocol is one which maintains an identity of a client forever.

Ex: FTP

Note: The following table gives Java technology version and its language version and comparison.

Technology version	Language version	Comparison
JAVA	JDK 1.0 JDK 1.1	Tortoise
JAVA 2	JDK 1.2 JDK 1.3	Dog
	JDK 1.4	Horse
JAVA 5	JDK 1.5 JDK 1.6	Tiger (given by SUN) Mustang (Ford Icon)

As on today J2SE must be taken as JE Similarly  
J2EE as JEE and J2ME as JME.

### FEATURES (OR) BUZZWORDS OF JAVA:

Features of any programming language is nothing but the services or basic facilities provided by the language to the industry programmers. The language JAVA provides 13 features. They are

1. Simple
2. platform independent
3. Architectural neutral
4. portable
- \* 5. Multithreaded
6. Networked
7. distributed
8. High performance
9. Interpreted
10. Dynamic
11. Robust (strong)
12. Secured
13. Object oriented programming Language

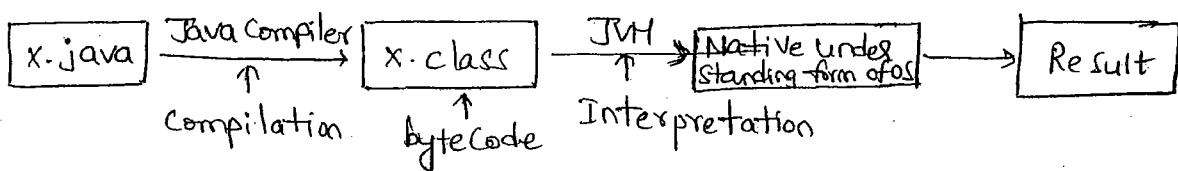
#### 1. Simple :

Java is simple because of the following factors

- a. Java programming does not support pointers. Hence we get less application development time and less execution time. Because of magic of byte code.

Note: Whenever we write any Java program, we save that program on of file name with an extension .java  
The following diagram gives the Java program execution

phases



Q) Define byte code and JVM.

Byte Code is the set of optimized instructions generated by Java compiler during compilation phase. The nature of the byte code is so faster than ordinary pointer code.

JVM is the set of programs developed by Sun Micro System and supplied as a part of JDK for reading line by line of byte code and converting into native understanding form of OS.

To complete the Java program execution, it has to undergo Compilation phase and interpretation phase. Hence

Java programming language is one of the compiler interpreted programming language.

b. Java programming environment is containing in build garbage collector program for collecting unused/un referenced memory locations. So that Java programmer is totally free from writing performance oriented programs. performance of the Java program taken care by Java software.

Q) Define a garbage collector?

A garbage collector is the system background java program which is running along with the regular java program for collecting unused/unreferenced memory space for improving the performance of Java application.

Garbage collector program running internally along with our Java program for regular periodical intervals of time.

Note: Java programming language does not contain destructors (like C++). In place of destructors we have garbage collector program.

c. Java programming contains rich set of API (Application programming Interface)

Def of API:

An API is a collection of packages. A package is a collection of classes, interfaces and sub packages. A sub package is a collection of classes, interfaces, and sub sub packages etc.

d. Java programming environment contains user friendly syntaxes so that one can develop effective applications. Compared to all the programming languages Java syntaxes are so easy.

2. Platform Independent:

A language or technology is said to be platform independent if and only if whole applications must run on every operating system without considering their vendors.

To say a language/technology is platform independent then it has to satisfy two properties. They are

- The language data types must take same amount of memory space on every operating system
- The language must contain some special programs which will convert from one understanding form of one as

to another understanding form of another OS.

The C, C++ applications are platform dependent because

→ The C, C++ data types are taking will take different memory spaces on different OS.

→ The C, C++ softwares are not containing any special programs to convert from one understanding form of one OS to another understanding form of another OS.

The language like Java and their applications are treated as platform independent, because of

→ Java language data types will take same amount of memory space on every OS.

→ Java software (JDK) contains some special programs for converting one format of one OS to another format of another OS.

Note : The slogan of Sun Micro system is

" write once run/reuse anywhere (WORA)"

Note : The OS DOS understands every application in its own understanding format called MZARITA (M2) whereas UNIX operating system understands in Embedded Linking format (ELF)

### 3. Architectural Neutral:

Architectural Neutral applications are those which are running on every processor without considering their architectures and vendors.

The languages like C, C++ are treated as architectural dependent, because C, C++ softwares is not able to convert automatically from the factors of one processor

to another factors of another processor.

The language like Java and whose applications are by default treated as architectural neutral because the software of Java (JDK) contains some special programs which will convert from one factor of one processor to another factor of another processor.

Note: The basic aim of SUN MICRO system is the Java programmer should not think about operating sys and processor compatibilities.

#### 4. Portable:

A portable application is one which can execute on every operating system and processor without considering their architectures and vendors.

The languages like C, C++ applications are not portable, because either os or processor is different they are unable to run by default.

The language like Java and whose applications runs on every os and processor by default. Hence Java is one of the portable language.

According to SUN MICRO system

portability = platform independent + architectural Neutral.

Note: Industry is always recommended to use portable languages for development of distributed application.

If we choose non portable languages for development of distributed applications then the team of software engineers has to spend more amount of time or considerable amount of time to convert non portable to

portable by writing some special programs

## 5. Multithreading:

- The basic aim of multithreading is to achieve concurrent execution.
- Definition of a Thread:  
A flow of control is known as thread.
- The basic usage of a thread is to execute user/programmer defined methods (functions).
- If any Java program is containing multiple flow of controls then the Java program is known as multithreaded.
- The languages like C, C++, Pascal, COBOL, etc., are treated as single threaded modelling languages because their execution environment contains single flow of control. Using these languages we can achieve sequential execution but not concurrent execution and more over these languages does not contain any predefined library for development of multithreading applications.
- The languages like Java and .Net are treated as multithreaded modelling languages, because their execution environment contains multiple flow of controls. Using these languages we can achieve both sequential and concurrent execution and more over the language Java contains an effective library for development of multithreading application.
- ↳ The library for developing multithreading application
  - a) `java.lang.Thread (class)`
  - b) `java.lang.Runnable (interface)`

\* The real time implementation of multithreading concept is that to develop real world servers such as

Tomcat, Weblogic, Websphere, Oracle SQL servers etc.

→ Each and every Java program is containing by default two threads. They are

- Foreground / child thread
- Background / parent thread

A foreground thread is one which is always executing user programmer defined methods. By default there exists a single foreground thread. But as a programmer there is a possibility to create multiple foreground threads to execute multiple methods concurrently.

A background thread is one which is always monitoring the execution status of foreground thread(s). By default there exists single background thread and always recommended to have single background thread even programmatically per Java program.

Therefore in a Java programming execution environment it is highly recommended to have multiple foreground threads and a single background thread per Java program.

→ Multi-threading is one of the specialized distinct feature of Java which was brought from multi-tasking concept of OS.

Q) How do you justify "every Java program is multi-threaded"?

A) Whenever we execute any Java program, to execute the programmer logic there exist one thread known as

foreground | child thread.

To monitor the execution status of foreground thread, internally one more thread is executing (garbage collector) and it is known as background thread.

Hence in every Java execution environment there exist multiple threads. So that we can say every Java program is multi-threaded.

Note: It is highly recommended to the Java programmers not to create unnecessary flow of controls because unnecessary flow of controls leads to inconsistent / dead lock result which is not recommended.

## 6. Networked:

→ The basic aim of networking is to share the data b/w multiple machines either they are located in same network or located in different network.

"Collection of interconnected non-autonomous / autonomous computers" is known as network.

According to industry standards we have two types of networks. They are

- i. Untrusted network
- ii. trusted network.

An untrusted network is one in which there exists "Collection of interconnected non-autonomous computers".

Using Java (network programming) we can develop the applications for running on untrusted network architecture and these types of applications are known as intranet applications.

Intranet applications are those which are able to run and whose result is able to share with in the limited distance.

These applications are preferred by those organizations whose business operations are available with in the limited place (known as small scale organizations).

A trusted network is one in which there exists "collection of interconnected autonomous computers".

Using J2EE Technologies (Servlets, JSP, EJB etc), one can develop internet applications to run on trusted network architecture.

Internet applications are those which are able to run and whose result is able to access across the globe.

These applications are preferred by those organizations whose business operations spread across the globe (known as large scale organizations).

Finally Java programming is containing an effective library for development of network oriented applications.

Note: Network oriented applications, are either internet applications or intranet applications.

## 7. Distributed:

According to industry standards a Java project can be developed in two ways.

i. Centralized application development

ii. Distributed application development

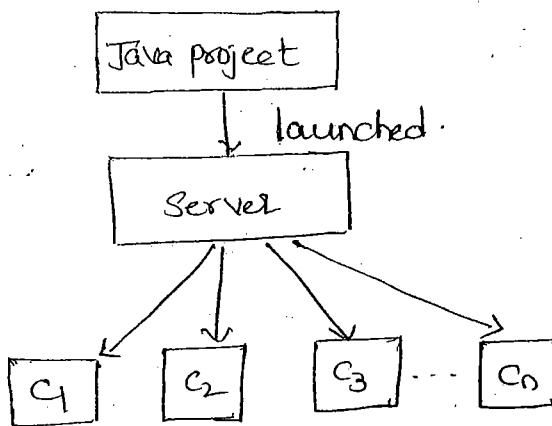
A centralized application is one which runs in the

Context of single server.

In real world centralized applications will not be having public URL. for example, SBI application can be accessed by

12

only the employees of SBI but not by other public people. The limitation of centralized application is that once the server is down, clients are unable to communicate unless and until server is uploaded, i.e. we get less availability of data.



A distributed application is one which is running in the context of multiple servers and whose results are able to access across the globe.

All the real world websites are the examples of distributed applications ([www.mail.yahoo.com](http://www.mail.yahoo.com), [www.gmail.com](http://www.gmail.com), etc.)

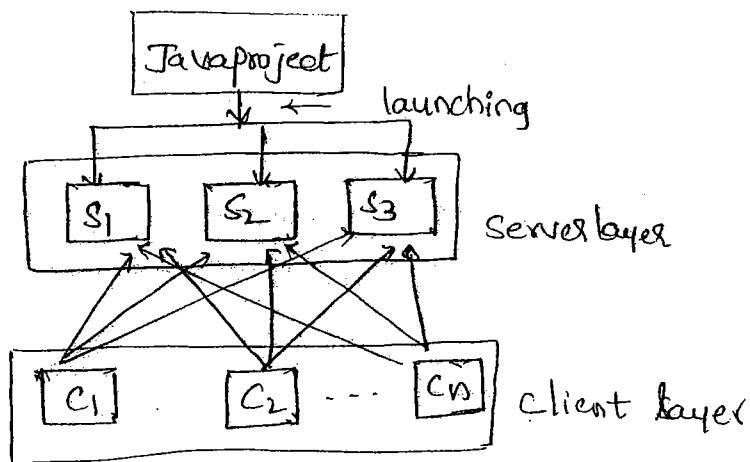
Every distributed application will be having public URL.

The advantage of distributed applications is that even though one server is down, clients are able to communicate from other servers. So that we get more availability of data.

The negligible drawback of distributed applications is that designing cost of the servers is more.

Q) What is the difference b/w URL and URI?

a) The place where the application is running is known as URL where as the place where the application is residing is known as URI.



### 8. High performance:

Java is one of the high performance programming language because of the following points.

- a execution time of an application is very less compared to any programming languages because of the magic of byte code.
- b. Java programming runtime environment automatically provides automatic memory management (garbage collector).
- c. We know that Java programming does not support pointers concept. So that a complex concept is eliminated from Java programming and one can develop the program as easily as possible. It gives more performance to programmer that is programmer performance is improved (according to industry).

### 9. Interpreted:

In the initial versions of Java, compilation phase is so faster and interpretation phase is very slow. This was the complaint given by Industry to the Sun micro systems.

SUN micro system developed set of programs named as JIT (Just In Time Compiler) and added as a part of JVM for speedup or strengthening the interpretation phase.

In the current versions of Java interpretation phase is so faster than compilation phase of Java.

Therefore Java is one of the highly interpreted programming language.

Q) Define JIT ?

A) JIT is the set of programs developed by SUN Micro System and added as a part of JVM to speedup the interpretation phase.

JIT reads the entire section of the byte code and converting into native understanding form of OS.

II. Robust (strong):

In the languages like C, C++, if we enter any invalid input then the program will be closed without giving any appropriate message which is not recommended. This indicates the languages called C, C++ are not robusted languages. Because they are unable to solve runtime problems.

The language like Java is having a capability for addressing runtime problems. Hence Java is one of the robusted programming language.

Def of Exception: Runtime errors of the program ~~are~~ known as exceptions.

Q) Define compile time errors and runtime errors?

A) Compile-time errors are those which are resulted when the programmer is not following syntax rules of the language.

Runtime errors are those which are resulted when the normal user enters invalid input at run-time.

## 10. Dynamic:

In any programming language memory can be allocated in two ways. They are:

- a. static memory allocation
- b. dynamic memory allocation

A static memory allocation is one in which memory will be allocated at compile time. Because of static memory allocation we get the following limits

- Waste of memory space (the no. of values we enter at run time are less than the size of the array)
- loss of data (the no. of values we enter at run time are more than the size of the array)
- Overlapping of existing data (if an array is containing all of values and if you are trying to insert a new value within the existing memory location then the existing value will be replaced by a new value, because as a programmer we are unable to move the elements either downwards or upwards).

Java programming will not follow static memory allocation but it always follows dynamic memory allocation because the concept of dynamic memory allocation eliminates the limitations of static memory allocation.

f) dynamic memory allocation is one in which memory will be allocated at run-time by using new operator and

this operator is known as dynamic memory allocation.

## 12. Secured:

Java is one of the secured programming language because library of Java contains readily available security API which are totally developed by Sun micro system by following network security algorithms.

As a software engineer we need not to write special programs on our own to provide security to the

## Def of Security:

Security is one of the principle used in the software industry to prevent or protect the confidential information from unauthorized users (& unauthorized users are those who does not have security credentials such as username, password, bar code, pin, etc.,.)

Therefore compared to all the programming languages having more secured inbuilt facility or inbuilt program.

## DATA TYPES:

Data types are used for storing input of our program in the main memory of the computer by allocating sufficient amount of memory space.

In any programming language, we have three types of data types. They are

1. Fundamental data types

2. Derived data types

3. User/programmer/ Secondary/ custom defined data types

Fundamental data types are those whose variables allow to store only one value but they never allows us to store multiple values of same type.

Ex: int a;

a=10; // Valid

a=10,20,30; // Invalid

Derived datatypes 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.

In every programming language the concept of arrays is treated as derived datatypes.

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

int b[] = {10, 10.75, 'A'}; // invalid

(Q) Define an array.

A) An array is a collective name given to a group of consecutive memory locations, which are all referred by similar type of elements.

User defined data types are those which are developed by programmers and whose variables allows us to store multiple values either same type or different type or both.

In C programming language we have a concept called structures, Unions, enum, typedef for developing user defined datatypes.

Similarly in Java programming to develop user defined data-types we have a concept called classes and interfaces.

Ex: Student s = new Student();

s

1	sno
Sathya	sname
99.99	marks
A	grade

Here s is a variable which is holding multiple values of same type or different type or both and whose type

is Student. Student is one of the user defined datatype.

### Java Fundamental Data Types:

Java fundamental data types are divided into 8 types and categorized into 4 groups. They are

1. Integer category data types
2. Float category data types
3. Character category data types
4. Boolean category data types.

#### Integer category data types:

Integer Category data types are used for storing integer data in the main memory of the computer by allocating sufficient amount of memory space. This

This category contains 4 data types and they are given in the following table

<u>SNO</u>	<u>Datatype</u>	<u>Size (in bytes)</u>	<u>Range</u>
1	byte	1	+127 to -128
2	short	2	+32767 to -32768
3	int	4	+2147483647 to -2147483648
4	long	8	

Note: To calculate range of any datatype in any programming language we use the following formula.

$$\text{Range of any datatype} = \left( \begin{array}{l} \text{(The no. of bits available} \\ \text{in the language which} \\ \text{is understood by computer)} \end{array} \right) \text{ (No. of bits occupied} \\ \text{by a particular datatype)}$$

Ex: Range of byte in Java =  $(2)^8$

$$\begin{aligned}
 &= 256 \\
 &= 1 \text{ to } 256 \\
 &= 0 \text{ to } 255 \\
 &= 0 \text{ to } \frac{255}{2} \\
 \Rightarrow & 127.5 \Leftrightarrow 127.5 \\
 &\frac{(-0.5)}{+127} \quad \frac{(+0.5)}{-128}
 \end{aligned}$$

$\therefore$  Range +127 to -128.

### Float category datatype:

Float Category datatypes are used for storing real constant values in the main memory of the computer by allocating sufficient amount of memory space.

Float Category contains two datatypes which are given in the following table

SNO	Datatype	Size(in bytes)	Range	No.of decimal places
1	float	4	+x to -(x+1)	8
2	double	8	+y to -(y+1)	16

If we want to store any real constant value in float data type then float data type organizes the data in the main memory in such a way that after the dot (.), it takes 8 decimal places. and moreover the real constant value must be followed by a letter f.

Ex: float f<sub>1</sub> = 10.75f;

10.75000000

If we want to store any real constant value in double data type then the double data type organizes the real constant value in main memory in such a way that after

dot(.) it takes 16 decimals. and more over the real constant value should not be followed by a letter f.

In other words if we use a real constant value directly in the Java program then that real constant value is by default treated as highest data-type in float category i.e. double data-type.

Ex: double d = 10.75;

d  
10.750000000000000

### Character Category Datatype:

A character is an identifier which is enclosed with in single quotes.

Ex: 'A', '\$', 'a', '#' etc.

Collection of characters enclosed within double quotes is known as string.

Ex: "abc", "Java world", etc.

To store the character data, we use a datatype called char in Java.

char datatype takes two bytes in Java because Java programming language follows UNICODE character set (Universal code character set) whereas C, C++ char datatype takes one byte because it follows ASCII character set.

### Def. of UNICODE:

A UNICODE character set is one which contains all the characters which are available in 18 international languages.

ASCII character set is one which contains all the characters which are available in only international languages.

Called English.

The languages like C, C++ are available in English and the language Java is available in 18 international languages (English, Greek, Latin, etc)

Note: As per as English Java programmer is concerned, to represent a character, we need one byte and the rest of the one byte is simply wasted. This is one of the negligible drawback of Java language.

### Boolean category data-types:

Boolean category data-type is always used for representing logical values i.e true and false.

In Java programming boolean category data-type contains a keyword boolean for storing either true or false values.

Boolean datatype in Java takes zero bytes of main memory space.

The datatype boolean implemented by sun micro system with the concept of flip-flop. Flip-flop is the general purpose register which allows us to store one bit of information that is

1 - true

0 - false

Note: In C, C++ we does not have the data-type directly boolean but the boolean values like true and false represented by non zero and zero respectively (int data types)

## Variables in Java:

In any programming language, values will be stored in the main memory of the computer by allocating sufficient amount of memory space with qualified variables that is without variables one can't store the values in the main memory of the computers.

All the variables in various programming languages are treated as building blocks.

### Def. of a variable:

A variable is an identifier whose value will be changed during execution of the program.

### Rules for writing variables:

Before using any variables in our programs, the programmer has to follow the set of rules given by the software industry.

1. The first letter of the variable name must be an alphabet.
2. The length of the variable can be upto 32 characters.
3. No special symbols are allowed in a variable name except underscore ('\_')
4. NO Keywords to be used as variable names.

(Q) Define a keyword or reserve word?

- A) A keyword or reserve word is one which is having a special meaning to the compiler of the particular programming language.

Ex: do, while, const, int, etc.

## Declaration of a variable:

Declaration of a variable is nothing but allocating sufficient amount of memory space along with the variable name. Before using any variable in our program, that variable must be declared just before one line (known as dynamic usage).

Syntax: datatype v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>n</sub> ;

Here datatype represents either fundamental datatype or derived datatype or user defined datatype.

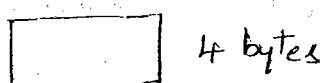
v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>n</sub> represents valid variable names of Java.

Ex: int a, b, c;  
float f<sub>1</sub>, f<sub>2</sub>;  
char c<sub>1</sub>, c<sub>2</sub>;  
boolean b;

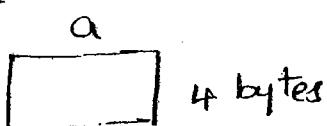
Q) What happens when a variable is declared in the main memory of the computer?

A) Ex: int a;

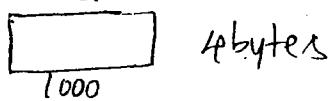
1. When we declare a variable sufficient amount of memory space is created.



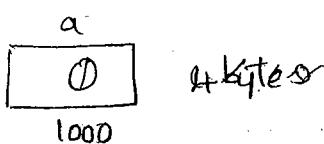
2. The created memory space is qualified by appropriate variable name.



3. Every memory space which was created must be qualified by distinct address 'a'



Memory Space is one of the important aspect in Java programming, it is by default contains a default value depends on datatype.



### Note :

- \* All the integer category datatypes variables contains a default value 0.
- \* All the float category datatypes variables contains a default value 0.0.
- \* All the character category datatypes variables contain a default value nothing.
- \* All the boolean category datatypes variables contains a default value false.

### Declaration Cum Initialization:

Declaration cum initialization is nothing but placing our own values & fixed value in a variable when ever the memory space is created without placing default values.

The concept of initialization taken by the programmer when they decided to place the fixed values or predecided values.

### Syntax:

datatype v<sub>1</sub> = val<sub>1</sub>, v<sub>2</sub> = val<sub>2</sub>, ... , v<sub>n</sub> = val<sub>n</sub>;

In the above syntax val<sub>1</sub>, val<sub>2</sub>, ... val<sub>n</sub> represents the list of values initialized for the variables v<sub>1</sub>, v<sub>2</sub>, ... v<sub>n</sub> respectively.

Ex : int a=10, b=20;

```
float PI = 3.1417f;  
char ch = 'A';  
boolean b1 = true;
```

## Constants in Java:

A constant is an identifier whose value can not be changed during execution of the program.

In order to make any variable value as constant, any method definition as constant and any class definition as constant then we have to use a keyword called 'final'.

'final' is a keyword which is placing an important in Java in three places.

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

### 1. 'final' at variable level :

If we don't want to change the value of the variable then the variable value must be made as constant by using 'final' keyword.

Syntax: final datatype v<sub>1</sub>=val1, v<sub>2</sub>=val2, ..., v<sub>n</sub>=valn;

Ex: final float PI = 3.1417f;  
PI = PI + 2; } // Invalid  
PI = 3.1418f;

Once the variable value is final, which is not possible to modify and it is not possible to reinitialize.

Therefore final variable values can not be modified or cannot be changed.

## 2. 'final' at method level:

We know that in most of the programming languages methods (functions) are used for performing some specific operations that is without methods concept we may not be writing effective programming.

If we don't want to change the definition of the method then the definition of the method must be made as constant by using 'final' keyword.

In other words in software development, if we develop any common method for many no. of Java programmers then it is highly recommended to make such common methods as final.

Syntax:

```
final return-type methodname (list of formal params if any)
{
    Block of stmt(s)
}
```

Ex: final float simpleInterest (float p, float T, float R)
{
 float si = (P \* T \* R) / 100;
 return (si);
}

Therefore final methods can not be overridden.

Note: final methods can be used every where but whose definitions can not be changed.

(Q) Define method overloading and method overriding?

A) method overloading:

A method is said to be overloaded if and only if method name is same but signature is different.

Signature represents any one of the following.

- a. no. of parameters } at least one thing must be
- b. type of parameters } differentiated.
- c. order of parameters }

Ex: `sum(10, 20, 30); —①`

`sum(10, 20); —②`

`sum(10.5f, 20.5f); —③`

`sum(10.5f, 200); —④`

`sum(200, 10.5f); —⑤`

Here the method `sum()` is overloaded method.

Overloaded methods may or may not be final.

Method overriding:

1. Method overriding = method heading is same + method body is different

2. The process of redefining the original method into various implementations or definitions. This process is known as method overriding.

final methods cannot be overridden.

Ex: `void operation(int x, int y)`

{

`int z = x + y; // ORIGINAL METHOD`

=

}

```

void operation( int x, int y)
{
    int z = x-y; // Re-defined method
}

void operation( int x, int y)
{
    int t=x; // Re-defined method
    x=y;
    y=t;
}

```

Here the method operation() is known as overridden method.

If the original operation() method is made as final then redefined methods of operation() are not possible.

#### Note:

- final methods can't be overridden (True / False) : True
- non final methods can be overridden (True / False) : True
- final methods can be overridden (True / False) : false
- 'final' at class level:

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

Syntax: final class <classname>

```

{
}

```

Ex: final class personal

```
{  
    int pin;  
    int cardno;  
    int barcode;  
}
```

class others extends personal //error

```
{  
}  
}  
}
```

Therefore final classes never participates in inheritance concept. In other words final classes are not reusable.

Note: final variable values can't be modified

final methods cannot be overridden

final classes will not be participated in inheritance process.

### 13. Object oriented programming features / principles / concepts:

Object oriented programming features are invented by

OMG (Object Management Group).

In information technology we have two types of paradigms

They are

1. procedure oriented programming languages

2. object oriented programming languages

→ whatever the data we represent in the main memory of the computer by using procedure oriented programming languages, that data can be accessed by some other program which is written in same procedure oriented programming language i.e. the result of one C program

can be accessed by another C program, which is not recommended. This situation makes us to understand procedure oriented programming languages are not providing enough security to protect confidential information from unauthorized users. Therefore procedure oriented programming languages provides 0% security. Hence industry is not recommended to develop the distributed applications with procedure oriented programming languages.

Ex: C, PASCAL, COBOL, FORTRAN, etc.

The basic aim of object oriented programming is that to get 100% security.

Whatever the data we represent with object oriented programming languages, that data is fully protected from unauthorized users.

Ex: Smalltalk, LISP, object COBOL, object PASCAL, Ada, C++ Java, .Net, etc.

Even though C++ is listed under object oriented programming, according to industry experts the language C++ is not a pure object oriented, but it is a partial object oriented because of the following reasons

- a. every C++ program can be exactly written like a 'C' program.
- b. C++ has containing a concept called friend functions which can access private data. So that we loose the security.
- c. A C++ program may satisfy all the principles of OOPS or may not be satisfying all the principles of OOPS.

The languages like Java and .Net are treated as pure Object oriented programming languages because of every Java program will completely satisfy all the OOPS principles and there is no concept of friend functions.

If any programming language is completely satisfying all the principles of object oriented programming then that language is called pure Object oriented.

Object Oriented Programming Contains 8 principles.

They are

1. Classes
2. Objects
3. Data abstraction
4. Data encapsulation
5. Inheritance
6. Polymorphism
7. Dynamic binding
8. Message passing

All the above object oriented principles are common for every object oriented programming but whose implementations or syntaxes are different in various object oriented programming languages.

### 1. Classes:

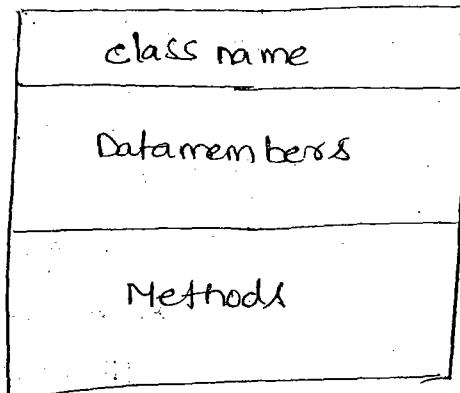
→ The basic aim of class concept is to develop user-defined data types.

→ To develop the concept of class we use a keyword called 'class'

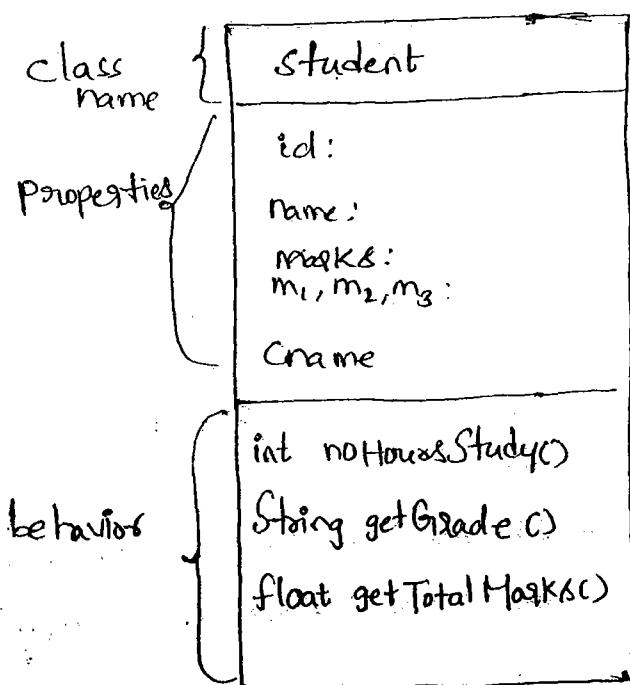
Def of class: The process of binding the data members and associated methods in a single unit. This single unit is

Known as a class.

- \* Data members of a class are also known as properties or attributes whereas methods of a class are also known as behaviors or accessories.
- \* In real time applications the definition of the class can be expressed in 'class diagrams' by using UML softwares like Rational Rhapsody and Turbo Analyst (IBM products).
- \* Structure of class diagram:



Ex: Design a class diagram for student by taking appropriate data members and suitable methods.



- In Object Oriented Programming we have two types of method. They are
  - Member methods
  - Non-member methods
- \* A member method is one which is defined/available with in the scope of the class. Every member method can access the data of the class.
- \* A non-member method is one which is not defined/available with in the scope of the class.
- A non-member method cannot access the data of the class.

In Java programming we have the concept of member methods but we does not have non-member methods.

Concept for exp obtaining effective security.

When ever we define a class, there is no memory space for data members and methods of a class. Memory space will be created when we create an object with respect to a class.

Syntax for defining a class:

```
Class <classname>
{
    variable declaration;
    methods definition;
}
```

Explanation:

In the above syntax

- Class is the keyword used for developing user/programmer defined data types.
- classname represents a valid variable name in java and it is treated as classname. We know that every class name is one of the user defined data-type.

that

In our Java programming class names are always used for creating the objects because when we define a class there is no memory space for data members and methods and whose memory space is creating when we create an object.

→ variable declaration represents data members of the class. The variables will be selected based on type of the class.

→ Methods definition represents the type of methods which are selected based on the class for performing some specific operations. Every method of Java must be defined inside the class only i.e Java programming does not allow to define the methods outside the class.

The definition of the class must be placed with In { } and it may or may not be terminated by ;

Ex: Define a class student

```
class Student
{
    int id;
    String name;
    float m1, m2, m3;
    String crame;
    int nocturnStudy()
    {
        return (4);
    }
    float getTotalMarks()
    {
        return (m1+m2+m3);
    }
    String getGrade()
    {
        return "Distinction";
    }
}
```

}

}

Ex: Develop a class for computing sum of two numbers

class Sum

{

    int a, b, c;

    void input()

{

        a=10;

        b=20;

}

    void process()

{

        c=a+b;

}

    void output()

{

        System.out.println("value of a=" + a);

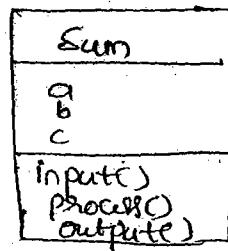
        System.out.println("value of b=" + b);

        System.out.println("value of c=" + c);

        // ("Result = " + c);

}

}



Note: according to industry standards every class is highly recommended to have set of methods for accepting input, processing the input and displaying the output.

(a) Define business logic and business logic method.

(i) The block of statements written by the programmer for solving client requirement is known as business logic.

The set of user defined methods developed by the programmer with business logic is known as business logic method.

Ex: In the above program the methods `input()`, `process()` and `output()` are known as business logic methods.

## 2. Object:

We know that when we define a class there is no memory space for the data members of the class and whose memory space will be created when we create an object. In other words if we want to enter the values for the datamembers of the class then we must create a memory space for the data members of the class in the main memory of the computer by creating an object.

Without object one cannot perform data processing kind of activity in our software development.

## Def of Objects

- \* Instance of a class is known as an object (instance is nothing but allocating sufficient amount of memory space for the data members of the class).
- \* Class variable is known as an object
- \* Grouped item is known as an object (a grouped item is the variable which allows us to hold multiple values of same type or different type or both).
- \* Value form of a class is known as an object.
- \* Blue print of a class is known as an object
- \* Real world entities are known as an object (entity is nothing but collection of values).
- \* The logical runtime entity is known as an object.

## Creation of Object:

Creating an object is nothing but allocating the memory space for data members of the class. We know that Java programming allocates the memory space at runtime, which is known as dynamic memory allocation.

In Java programming to allocate the memory space dynamically we use an operator called new. This operator is known as dynamic memory allocation operator.

## Operations of 'new':

When we use 'new' operator in dynamic memory allocation, internally it performs two types of operations, they are

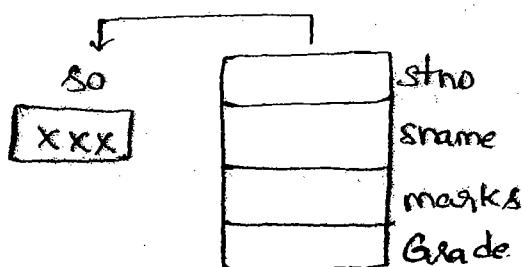
1. It allocates sufficient amount of memory space for the data members of the class.
2. It takes the address of the class and placed its address in the L.H.S. Variable (object name).

In Java programming we have two approaches or syntaxes to create the object.

### Syntax 1 / approach 1:

```
<classname> objname = new <classname>;
```

Ex: Student so = new student();



### Syntax 2 / approach 2:

```
<classname> objname ; → ①
```

```
objname = new <classname> ; → ②
```

Statement 1 represents Object declaration. Whenever an object is declared whose default value is null since there is no memory space for the data members of the class.

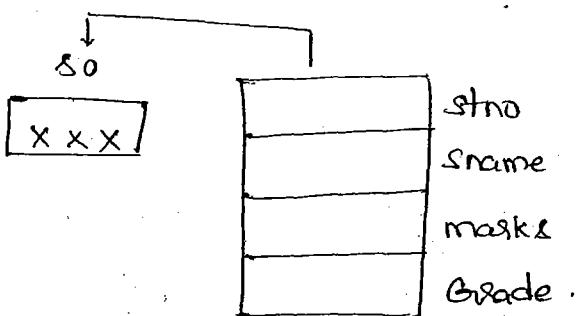
Statement 2 represents Object referencing. Whenever an object is referenced, the default value of the object is not null (address of the class) because we get the memory space for data members of the class.

Ex :

Student s0;

s0  
null

s0 = new Student();



Q) What are the differences between class and object?

Class

object

- \* The process of binding the data members and associated methods into a single unit is known as class.
- \* A class is treated as a logical existence in nature.
- \* Whenever we define a class, there is no memory space for data members of the class.
- \* Class variable is known as an object.
- \* An object is treated as a physical existence in nature.
- \* When we create an object, we get the memory space for data members of the class.

- \* The definition of a particular class will exists only once per Java program.
- \* With respect to one class definition we can create multiple objects.
- \* When we execute Java program, the class will be loaded into main memory first with the help of class loader subsystem.
- \* After loading the class in the main memory, later we can create objects.

Note:

1. All the objects of Java resides in heap memory.
2. All the methods are residing in stack memory.
3. All the constant values of our Java program resides in associative memory.

Q) Define classLoader subsystem?

- A) A classLoader subsystem is a program available as a part of JVM for loading class files into main memory from the secondary memory.

Types of Data members (or) Variables in the class:

Each and every Java program must be written with respect to the class. We know that a class is containing collection of data members and methods.

In Java programming, data members of the class are divided into two types. They are.

1. Instance / non static data members.
2. Static data members.

## Instance Data members

\* Instance data members are those whose memory space is created each and every time when the object is created.

\* Which ever variable of a class is having dependent value then that variable will be taken as instance data member.

\* programmatically, instance data members declaration should not be preceded by a keyword static

### \* Syntax:

datatype v<sub>1</sub>, v<sub>2</sub>, ... v<sub>n</sub>;

\* Instance data members must be accessed with respect to object name.

ObjName :: instance data members  
name

\* Instance data member values are not sharable.

\* Instance data members are also known as Object Level Data members (OLDM) because they depends on object and independent from class.

## Static Data members

\* static data members are those whose memory space is creating only once when the class is loaded in the main memory irrespective of no. of objects are created.

\* Which ever variable is taking a common value for group of objects, that variable must be taken as static data member

\* programmatically static data members declaration must be preceded by a keyword static

### \* Syntax:

static datatype v<sub>1</sub>, v<sub>2</sub>, ... v<sub>n</sub>;

\* Static data members must be accessed with respect to class name.

ClassName :: static data members  
name

\* static data members values are sharable.

\* static data members are also known as Class Level Data members (CLDM) because they depends on class and independent from object.

Ex: int stno, pin;

\* Ex: static string cname, bname;  
branch name;

Note : Each and every final variable must be static

→ A static variable may or may not be final.

→ All the data members of the pre-defined classes are by default belongs to

public static final XXX

Here XXX represents datatype, variable name, Variable Value

Ex: Java.lang.Math ↴

public static final float PI = 3.14159;

### Types Of Methods:

We know that every Java program must be developed with the concept of class. Class is containing collection of data members and methods.

In Java programming methods are classified into two types

They are

1. Instance / non-static methods
2. static methods.

#### Instance method

\* Instance methods are those which are recommended to perform repeated operations such as reading the records from the file, reading the records from the database etc.

\* Programmatically, instance methods definition should not be preceded by a keyword static

#### static method

\* static methods are those which are recommended to perform one time operations such as opening a file either in read mode or write mode, obtaining a data base connection; etc.,

\* Programmatically, static methods definition must be preceded by a keyword static

\* Syntax

returntype methodName( list of  
formal params if any)

{

Block of stmt(s);

}

\*\* Each and every instance method  
must be accessed with respect  
to Object name

Objname. instance methodName()

\* The result of instance methods  
is not sharable among the  
objects

\* Instance methods are also known  
as Object level methods because  
they depends on object and  
Independent from class.

Ex: void total();

{

    tot = m<sub>1</sub> + m<sub>2</sub> + m<sub>3</sub>;

}

\* Syntax

static returntype methodName  
( list of formal params if any)

{

Block of stmt(s);

}

\*\* Each and every static method  
in Java must be accessed with  
respect to class name.

classname. staticmethodName()

\* The result of static methods  
is always sharable among the  
objects

\* static methods are also known  
as class level methods because  
they depends on class name and  
Independent from object name.

Ex:

public static void main(String a[])

{

    =====

    S<sub>1</sub>.total();

    S<sub>2</sub>.total();

    =====

}

Note: A class in Java contains instance data members, static  
data members, instance methods and static methods.

Instance data members and instance methods must be accessed with respect to Object name.

Static data members and static methods must be accessed with respect to classname.

`System.out.println()` —①

`System.out.print()` —②

Statement ① is used for displaying the data / result on the console line by line.

Statement ② is used for displaying the data / result on the console in the same line.

`println()` and `print()` methods are the predefined overloaded instance methods.

\* To access these `println()` and `print()` methods we need to use an object of PrintStream class because these two methods presents in a predefined class called PrintStream.

\* An object of PrintStream class called out created in another predefined class called System as a static data member.

Hence `print()` and `println()` methods must be accessed as follows

`System.out.println()`

`System.out.print()`

The following code gives the development aspect of the above two sentences.

class PrintStream

{

    public void println()

    {

    }

    public void print(→)

    {

class System

{

    static printStream out;

}

### Ex:

1. "Hello Java world"

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

2. int a=10;

System.out.println(a); // 10

System.out.println("value of a = " + a); // value of a = 10

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

3. int a=10, int b=20;

int c=a+b;

System.out.println("sum = " + c); // sum = 30

System.out.println("is the sum"); // 30 is the sum

System.out.println("sum of " + a + " and " + b + " = " + c);

// sum of 10+20 = 30

### Structure of Java program:

In order to write any Java program, the programmer must follow a standard structure.

4. package Details

5. class <clsName>  
{

6. **Data members**

7. **User-defined methods**

8. public static void main (String ar[])

9. {

10. Block of Stmt(s);

11. }

12. Explanation:

In the above structure

1. package Details represents collection of classes and interfaces. If we use any predefined classes or interfaces as a part of our Java program, it is the responsibility of the Java programmer to specify to which package the predefined classes and interfaces presents.
2. 'class' is a keyword used for developing user-defined data types. Every program in Java must be developed with the concept called class.
3. <clsName> represents a valid variable name of Java and it is treated as className. In Java programming every class name is treated as user/programmer defined data type. We know that whenever we define a class there is no memory space for data members of the class and class names are used for creating the objects.

4. Data members represents either instance or static and they will be selected based on class name.
5. User-defined methods represents either instance or static and we know that methods are used for performing some specific operations. Every user defined method of Java must be defined inside the class only. and such methods are known as member methods (business logic methods).
6. Every Java program starts executing from main(). Hence main() is known as program driver.
7. main() never returns any value. Hence its return type must be void
8. main() is executing only once throughout the life of the entire Java program. Hence its nature must be static
9. main() can be accessed by everybody that is universal access. Hence its access specifier must be public.
10. Every main() takes array of objects of String class.
11. Block of statements represents set of executable statements, which inturns calls the user defined methods.
12. Which our class is containing main(), that class name must be given as a filename with an extension .java.

1) Write a Java program which will display a message "Hello Java world".

// First.java

```
class first  
{
```

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

```
{
```

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

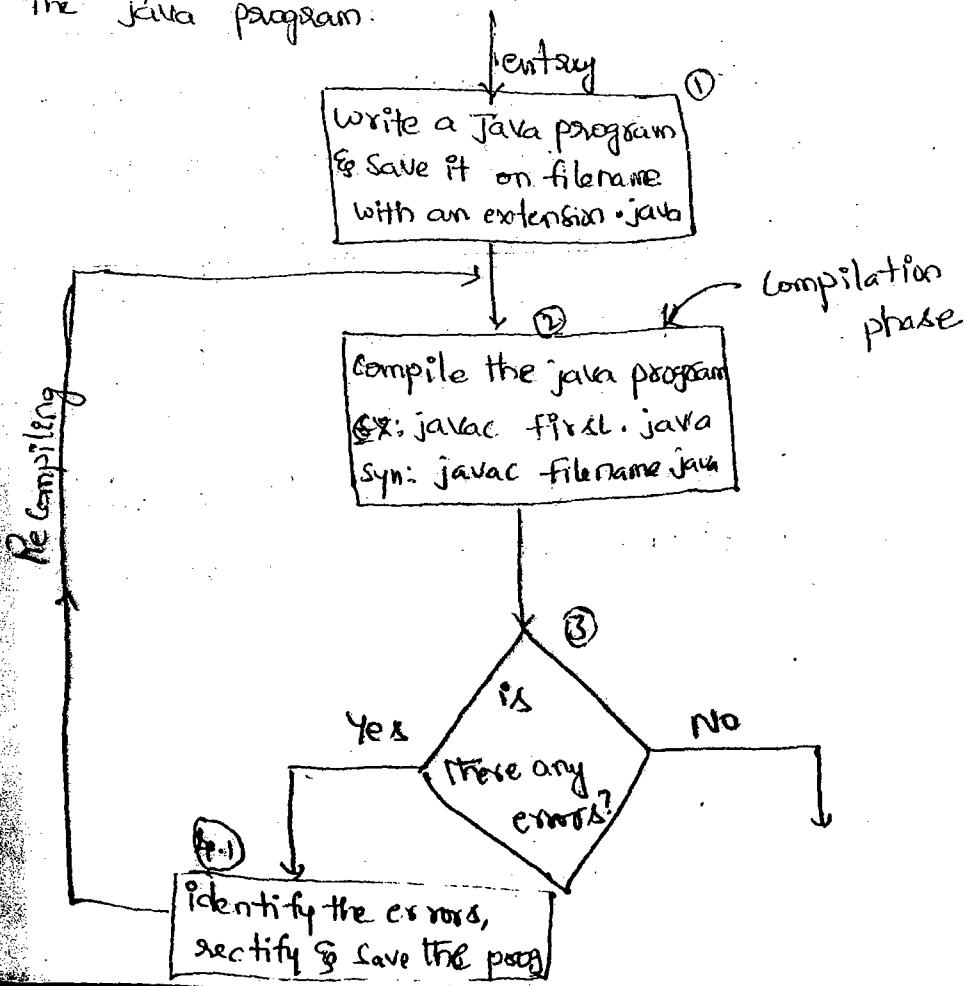
```
        System.out.println ("This is my first program");
```

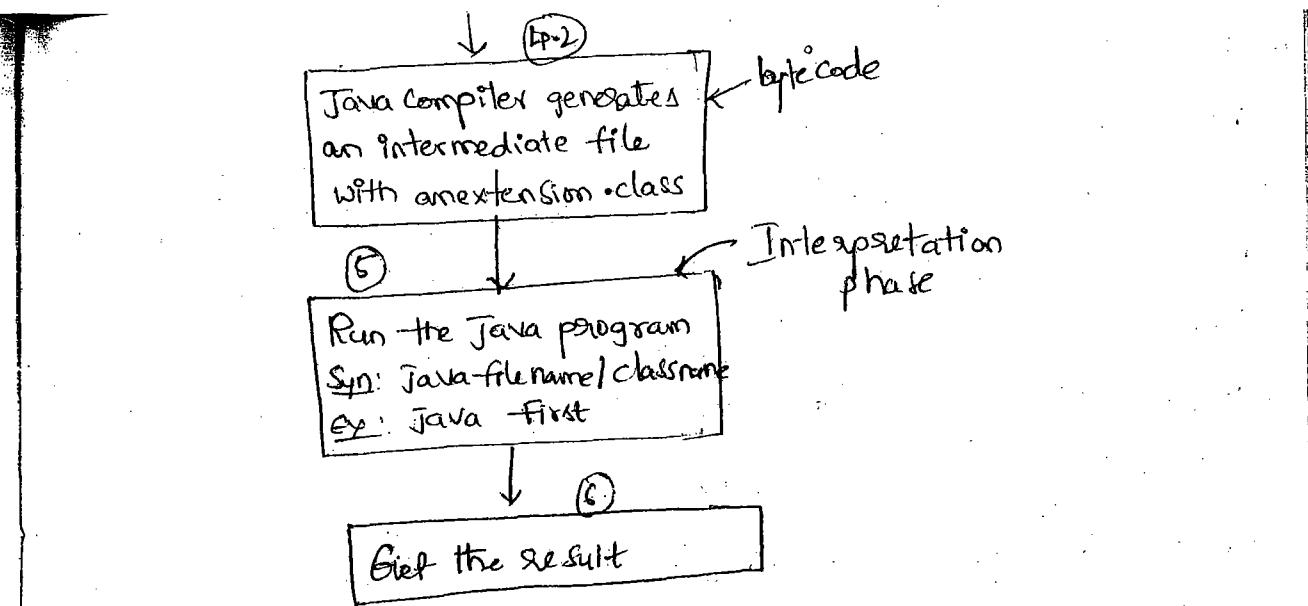
```
}
```

```
}
```

Compiling and running the Java programs:

The following diagram gives steps for compiling and running the Java program.





In the above diagram 'javac' and 'java' are the two exe files / application programs / tools developed by Sun Microsystems and supplied as a part of JDK 1.5\bin folder for compiling and interpreting the Java program.

### Hungarian Notation:

Hungarian notation is one of the naming convention followed by Sunmicro System for development of predefined classes / interfaces, methods and data members.

#### 1. Hungarian Rule for Class / Interface:

If a class name or interface name contains either one word or more than one word then all the words first letters must be capital.

Ex: Normal Notation

System  
actionevent  
NumberFormatException  
arrayIndexOutOfBoundsException  
thin

Hungarian Notation

System  
ActionEvent  
NumberFormatException  
ArrayIndexOutOfBoundsException

### 2. Hungarian rule for methods

If a method is containing either one word or more than one word then the first word first letter is small and rest of the subsequent words first letters must be capital.

NN	HN
println()	println()
actionperformed()	actionPerformed()
itemstatechanged()	itemStateChanged()

### 3. Hungarian rule for datamembers

If we want to use any data members of the predefined class, then they must be used as Capital letters.

If the data member of the pre-defined class contains more than one word then they must be separated by underscore(\_).

In other words Sun micro system has taken every data member as Capital letters and multiple words are separated by underscore.

NN	HN
pi	PI
e	E
maxpriority	MAX_PRIORITY
maxvalue	MAX_VALUE
scrollbarsverticalonly	SCROLLBARS_VERTICAL_ONLY

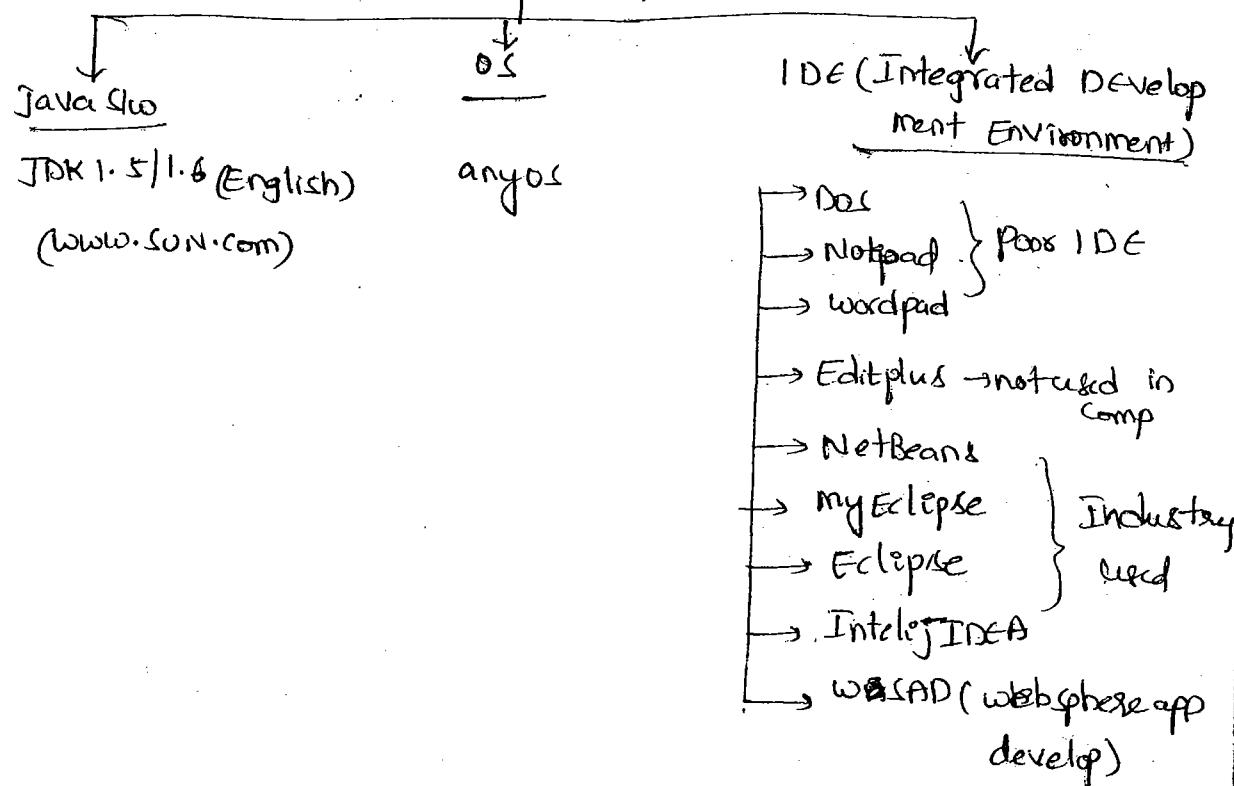
Hungarian notation is mandatory for pre-defined classes/interfaces, predefined methods and predefined data members

Hungarian notation is optional for user-defined classes/interfaces, userdefined methods and user defined data members

In software industry, our working company will always recommended us to follow hungarian notation for both pre defined and user defined classes / interfaces , methods and data members.

Note

Requirements for developing java project / app



Define IDE:

An IDE is the third party vendor software developed for developing java applications by getting some inbuild ~~compatibilities~~ compatabilities.

i \* Write a Java program which illustrate the concept of instance methods and static methods

Class Third

{

void f<sub>3</sub>()

{

System.out.println("I am from f<sub>3</sub>-Instance");

}

void f<sub>1</sub>()

{

f<sub>3</sub>(); // rule-2

Third.f<sub>2</sub>(); // rule-4

System.out.println("I am from f<sub>1</sub>-Instance");

}

static void f<sub>2</sub>()

{

System.out.println("I am from f<sub>2</sub>-static");

}

public static void main(String a[])

{

System.out.println("I am from main()-start");

Third t<sub>1</sub> = new Third();

t<sub>1</sub>.f<sub>1</sub>(); // rule-3

f<sub>2</sub>(); // rule-1 or Third.f<sub>2</sub>();

System.out.println("I am from main()-end");

} // main

} // Third class

### Rules:

Rule 1: one static method can call another static method directly provided both the methods belongs to same class.

Rule 2: one instance method can call another instance method directly provided both the methods belongs to same class.

Rule 3: one static method can call another instance method with respect to object without considering whether they are belongs to same class or different class

Rule 4: One instance method can call another static method with respect to class name, without considering whether they belongs to same class or different class.

→ Write a Java program which will compute sum of two numbers.

Class Sum

```

{
    int a,b,c;
    void assign()
    {
        a=10;
        b=20;
    }
    Process
    void display()
    {
        c=a+b;
    }
    void displayC()
    {
}

```

```
System.out.println("value of a = " + a);
```

```
System.out.println("value of b = " + b);
```

```
System.out.println("Sum = " + c);
```

}

### ↳ // Business Logic class

```
class SumDemo
```

{

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

{

```
    System.out.println("I am from main() beginning");
```

```
    Sum s0 = new Sum();
```

```
    s0.assign();
```

```
    s0.process();
```

```
    s0.display();
```

```
    System.out.println("I am from main() ending");
```

}

### ↳ // Execution Logic class

In the above program we identified the following limitations for getting different result.

- a. Start the program
- b. Change the values wherever it is required
- c. Save the program
- d. Compile the program.

The above cycle of operations must be repeated as and when we want to get different result.

If any program is satisfy the above factors then such program is known as static program.

In another words the static programming is an approach in which the values are passed at compile time or passed with in the program.

In the industry it is not recommended to write static programming and it is highly recommended to write dynamic programming.

A dynamic programming approach is one in which the values to the program will be entered at runtime.

length: 'length' is the one of the implicit variable created by JVM at runtime and supplied to every Java program for two purposes. They are

1. to find the no.of elements / size of the array
2. to treat the fundamental arrays as objects.

Syntax:

arrayVariableName.length;

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

`System.out.println("Size of a = " + a.length); // 3`

`float f[] = {5.5f, 9.99f};`

`System.out.println("Size of f = " + f.length); // 2`

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

`System.out.println(f[i]);`

```

String s[] = {"hyd", "che", "Bang"};
for (int i=0; i < s.length; i++)
{
    System.out.println(s[i]);
}

```

### Accepting the values Dynamically to the Java program

In Java programming accepting the data dynamically can be possible in three ways. They are

1. from cmd prompt

2. from Keyboard

3. from properties file

4. from cmdlinearguments:

The set of values passed to the Java program from the command prompt is known as command line arguments.

E:\7amJ2se> java SumDemo 20 30 ←

Command line  
arguments

main method takes array of objects of String class

for accepting command line arguments.

- \* In Java programming, each and every command line argument is internally treated as string data.

- \* All the command line arguments are passing to main()

- \* To hold all the command line arguments, we need a parameter / variable of String type in main().

- \* Since Java is one of the pure object oriented programming, in main() to hold all the command line arguments we need a variable / parameter of String class

type:

- \* To hold all unlimited no. of command line arguments, we need an array variable / parameter of String class type.
- \* At the time of taking array variable / parameter of String class type, one should not specify the size of the array. (If we specify the size then that variable will hold only limited no. of command line arguments).

Therefore every main() of java program takes array of objects of String class type.

→ Write a Java program to print command line arguments.

class Dataprint

```
{
    public static void main (String a[])
    {
        System.out.println ("No. of cmd line arguments " + a.length);
        for (int i=0; i<a.length; i++)
            System.out.println (a[i]);
    } //main()
}
```

} // Dataprint

> javac Dataprint.java ↵

> java Dataprint 10 10.25 a abc true ↵  
QF

No. of cmdline arguments 5

10

10.25

a

abc

true

## Converting String data into fundamental data:

Whenever we enter any command line arguments to the Java program, those values are treated as String data.

On String data one can't perform numerical operations.

To perform numerical operation on the command line arguments, it is highly recommended to convert String data into numerical data. The following table gives fundamental data type, corresponding wrapper class and conversion method from string data to fundamental data.

datatype	wrapper class	conversion method
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	public static char parseChar (String)
boolean	Boolean	public static boolean parseBoolean (String)

In general each and every wrapper class is containing the following predefined generalized method for converting string data into fundamental data

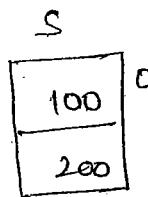
wrapper class ]

public static xxx parseXXX (String)

Here `xxx` represents any fundamental data type.

Ex: `String s[] = {"100", "200"};`

`s.o.p(s[0] + s[1]); // 100200`



`int x = Integer.parseInt(s[0]);`

`int y = Integer.parseInt(s[1]);`

`s.o.p(x+y); // 300`

or `s.o.p("Sum=" + x+y); // 300`

or `s.o.p("Sum=" + x + y); // 100200`

(Q)

Q) Define a wrapper class? explain its purpose.

A) for each and every data type there exists a predefined class. This predefined class is known as wrapper class.

The purpose of wrapper classes is that to convert string data into fundamental data.

→ W.a. java program for swapping of two values by accepting the values dynamically.

Class Swap

{

int a,b;

void assign( int x, int y)

{

a=x;

b=y;

}

void swapValues()

{ int temp;

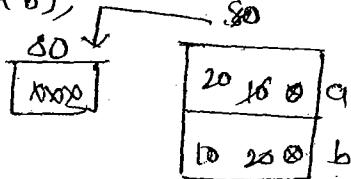
```

temp = a;      | a = a+b
a = b;        | b = a-b
b = temp;     | a = a-b
}

Void display()
{
    S.o.p ("value of a=" + a);
    S.o.p ("value of b=" + b);
}
} // Swap - ELC
Class SwapDemo
{
    public static void main (String k[])
    {
        int x = Integer.parseInt (k[0]);
        int y = Integer.parseInt (k[1]);
        Swap so = new Swap();
        so.assign(x,y);
        S.o.p ("The original values");
        so.display();
        so.swapValues();
        S.o.p ("The values after swapping");
        so.display();
    } // main-end
}
} // SwapDemo - ELC

```

(1) E  
(2)



→ Compile the above program

javac SwapDemo.java ↴

Ensure .class file must be generated.

→ Run the java program

java SwapDemo 10 20 ↴  
①

E  
8  
→

Q) Explain the flow of Java program (see the above program)

- A) When we write any Java program, in general we come across the following sequence of steps.
1. We enter command line arguments to the Java program from the command prompt
  2. Command line arguments are passed to execution logic class.
  3. Within the execution logic class, command line arguments are sending to execution logic method (main())
  4. In main(), cmd line arguments are treated as array of objects of String.
  5. As a Java programmer we convert the string data into fundamental data.
  6. Create an object of business logic class
  7. Send the commandline arguments values to the business logic by calling various methods. So that Calculations will be performed at business logic class and gives result back to execution logic class.
  8. program stops executing.

→ Write a Java program which will generate a multiplication table for a given number dynamically.

Class Mul

```

{
    int n;
    void set(int x)
    {
        n = x;
    }
}

```

```
Noid table()
{
    for (int i=1; i<=10; i++)
    {
        S.o.p (n + "*" + i + "=" + (n*i));
    } // for
} // table
} // Mul - ELC
```

```
Class MulDemo
{
    public static void main (String K[])
    {
        if (K.length != 1)
        {
            S.o.p ("plz enter only one value");
        }
        else
        {
            int x = Integer.parseInt (K[0]);
            if (x <= 0)
            {
                S.o.p (x + " is Invalid");
            }
            else
            {
                Mul mo = new Mul();
                mo.set (x);
                mo.table ();
            }
        }
    }
} // main()
} // MulDemo - ELC
```

Note: In the above program the block of statements we have written within the if-else statements are meant for performing validations.

→ Write a Java program which will compute factorial of a given number.

Class Fact

{

int n;

void set (int x)

{  
n = x;

}

int ~~fact~~ factCalc()

{

int f=1;

while (n > 0)

{

f = f \* n;

n = n - 1;

}

} return (f);

void disp()

{

int res = factCalc();

s.o.p ("Factorial = " + res);

}

} // Fact - BLC

$$n = \begin{cases} +ve & \text{Valid} \\ -ve & \text{Invalid} \\ 0 & \end{cases}$$

```

class FactDemo
{
    public static void main(String ar[])
    {
        if (ar.length != 1)
            S.o.p ("plz enter only one value");
        else
        {
            int x = Integer.parseInt(ar[0]);
            if (x < 0)
                S.o.p (x + " is invalid");
            else
            {
                Fact fo = new Fact();
                fo.set(x);
                fo.disp();
            }
        }
    }
}

```

↳ FactDemo - ELC

→ Write a Java program which will decide whether the given number is prime or not

Class Prime

```

{
    int n;
    void set (int x)
    {
        n=x;
    }
}

```

```

String decide()
{
    int i;
    for (i=2; i<n; i++)
    {
        int r=n%i;
        if (r==0)
            break;
        else
    }
    if (i==n)
        return "prime";
    else
        return "Not prime";
}

```

Y || Prime - BLC

```

class primeDemo
{
    public static void main(String a[])
    {
        if (a.length != 1)
            System.out.println("pls enter only one value");
        else
        {
            int x=Integer.parseInt(a[0]);
            if (x<=1)
                System.out.println(x+" is invalid");
            else
            {
                Prime po=new Prime();
                po.set(x);
                String res=po.decide();
            }
        }
    }
}

```

```

        S.o.p (res);
    } // else
} // else
} // main()

```

↳ If primeDemo - ELC

Note: A method is not only returning fundamental datatypes but it can also return user defined datatypes (class names), and interface names) as a return type

→ Write a Java program which will convert an ordinary number into roman number.

```

class Roman
{
    int n;
    void set (int x)
    {
        n=x;
    }
    void convert()
    {
        if (n<=0)
        {
            S.o.p ("No Roman equivalent... ");
        }
        else
        {
            while (n>=1000)
            {
                S.o.p ("M");
                n=n-1000;
            }
        }
    }
}

```

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

```

if (n>=900)
{
    s.o.p ("CM");
    n=n-900;
}
if (n>=500)
{
    s.o.p ("D");
    n=n-500;
}
if (n>=400)
{
    s.o.p ("CD");
    n=n-400;
}
while (n>=100)
{
    s.o.p ("C");
    n=n-100;
}
if (n>=90)
{
    s.o.p ("XC");
    n=n-90;
}
if (n>=50)
{
    s.o.p ("L");
    n=n-50;
}
if (n>=40)
{
    s.o.p ("XL");
    n=n-40;
}

```

a types  
 red,  
 number  
 while  
 if  
 :  
 f  
 while  
 f  
 if  
 if  
 while  
 f  
 if  
 while

```

if
while (n>=10)
{
    s.o.p ("X");
    n=n-10;
}
if (n>=9)
{
    s.o.p ("IX");
    n=n-9;
}
if (n>=5)
{
    s.o.p ("V");
    n=n-5;
}
if (n>=4)
{
    s.o.p ("IV");
    n=n-4;
}
if while n>=1
{
    s.o.p ("I");
    n=n-1;
}
else
{
    // convert()
}
// class RomanBLC

```

```
class RomanDemo
```

```
{
```

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

```
{
```

```
        if(a.length != 1)
```

```
            S.O.P ("pls enter only one value")
```

```
        else
```

```
{
```

```
            int x = Integer.parseInt(a[0]);
```

```
            Roman r = new Roman();
```

```
            r.set(x);
```

```
            r.convert();
```

```
} // else
```

```
} // main()
```

```
} // RomanDemo - ELC
```

HW.

→ write a Java program which will convert a roman number  
into ordinary number.

MMX → 2010

MMXI → 2011

MCA → Invalid.



- Write a Java program for getting reverse of a given number
- Write a Java program for obtaining factors of a given number.
- Write a Java program to check whether the given number is perfect or not (A perfect no is one whose factors sum is equal to original number)  $6 \rightarrow 1+2+3=6$
- Write a Java program which will print prime numbers within the given number (if we enter 40 then print the prime numbers within the 40)

→ // Reverse of the given number

class Reverse

{

int n;

void set (int x);

{

n=x;

}

void revCal ()

{

int rev;

for (rev=0; n>0; ~~rev~~)

{

rev\*10

rev = ~~rev~~ + (n%10);

n=n/10;

} if (rev == n)

s.o.p ("Reverse of " + n + " is " + rev);

}

}

class ReverseDemo

{

public static void (String a[])

{

if (a.length!=1)

s.o.p ("plz enter only one value...");

else

{

int x=Integer.parseInt(a[0]);

if (x\*\*=0)

System.out.println ("X + " is invalid..");

```

    else
    {
        Reverse r = new Reverse();
        r.set(x);
        r.revCalc();
    }
}

} // main()
} // ReverseDemo.

```

→ // Factors of a number

class Factors

```

    int n;
    void set (int x)
    {
        n = x;
    }

    void calFactors()
    {
        int i;
        for (i=2; i<n; i++)
        {
            if (n % i == 0)
                System.out.print(i);
        }
    }
}

```

} // Factors - BLC

class FactorsDemo

```

{
    public static void main (String a[])
    {
    }
}

```

```

if (a.length != 1)
    S.o.p("please enter only one values ...");
else
{
    if (x <= 0)
        S.o.p("x + " is invalid");
    else
    {
        int x = Integer.parseInt(a[0]);
        if (x <= 0)
            S.o.p(x + " is invalid");
        else
        {
            Factors f = new Factors();
            f.set(x);
            f.calFactors();
        }
    }
}
}

```

→ // perfect Number

class perfect

{

int n;

void set(int x)

{

n = x;

}

```

void process()
{
    int i, sum=0;
    for (i=1; i<n; i++)
    {
        if (n%i==0)
            sum=sum+i;
    }
    if (sum==n)
        s.o.p("perfect");
    else
        s.o.p("Not perfect");
}

```

} If perfect - BLC

```

class perfectDemo
{
    public static void main (String a[])
    {
        if (a.length != 1)
            s.o.p("pls enter only one value");
        else
        {
            int x = Integer.parseInt (a[0]);
            if (x<=0)
                s.o.p(x+" is invalid");
            else
            {
                Perfect p = new perfect();
                p.set(x);
                p.process();
            }
        }
    }
}

```

} // main()

} // Perfect Demo.

→ 1) prime numbers upto a given range.

class prime

{

int n;

void set(int x)

{

n=x;

}

void calPrimeNo()

{

int i, m;

for (int m=1; m<=n; m++)

{

for (i=2; i<=m; i++)

{

int r = m % i;

if (r==0)

break;

}

if (i == m)

S.o.p (i);

} // for

} // calPrimeNo()

} // Prime - BLC

```
class PrimeDemo
{
    public static void main (String a[])
    {
        if (a.length != 1)
            System.out.println ("pls enter only one value...");

        else
        {
            int x = Integer.parseInt (a[0]);
            if (x <= 0)
                System.out.println (x + " is invalid");
            else
            {
                Prime p = new Prime();
                p.set(x);
                p.calprimeNo();
            }
        }
    }
}
```

## CONSTRUCTORS In Java :

A constructor is the special member method which is called by the JVM automatically (implicitly) when the object is created for placing our own values without placing default values.

The basic purpose of constructors is that to initialize the Object.

### Advantages of Constructors:

When we use constructors as the part of our application we get the following advantages

1. It eliminates in placing default values
2. It eliminates in calling ordinary methods <sup>this, super</sup> explicitly.

### Rules/ properties/ characteristics of Constructors:

When we want to use constructors as a part of our applications, the programmer must follow the following rules

1. Constructors will be called automatically when ever an object is created.
2. The constructor name must be similar to class name.
3. Constructors should not return any value even void also.
4. Constructors should not be static because, constructors will be called each and every time as and when we create object (java programming does not containing static constructor).
5. Constructors are not inherited from one class to another class because, every constructor of the class is used for initializing its own data member but not for other class datamembers initialization.

6. A constructor can be private provided an object of one class can not be created in the context of some other classes.  
 (but it can be created in the same class)

Constructor cannot be private provided an object of one class to be created in the context of another classes.

### Types of Constructors:

In Java programming, we have two types of constructors.

They are

1. Default / parameter less / no argument / zero argument constructor
2. Parameterized constructor

Default Constructor: A default constructor is one which will not take any parameters.

If we want to place same values for the multiple objects of same class then we must use default constructor.

Syntax:

```
class <classname>
{
```

```
    _____
```

<classname> ( ) ← default  
constructor

```
{
```

Block of stmt(s),

```
}
```

```
    _____
```

```
{
```

→ Write a Java program which illustrate the concept of default constructor

Class Test

{

int a, b;

Test()

{

System.out.println("I am from default constructor");

a=10;

b=20;

S. O. P ("value of a = " + a);

S. O. P ("value of b = " + b);

}

} // Test

Class TestDemo1

{

public static void main (String a[])

{

Test t1 = new Test();

}

} // Test Demo1.

Output:

I am from default constructor

Value of a = 10

Value of b = 20

Rule 1: When ever we create an object only with default constructor, defining the default constructor is optional. If we define our own default constructor (programmer

- + defined default constructor) then JVM will execute program  
 - or defined default constructor. If we are not defining our own default constructor then JVM will call its own constructor called System defined default constructor and it is used for placing default values.

### Parameterized Constructors:

- i) A Parameterized constructor is one which always takes parameters.  
 ii) If we want to create multiple objects of same class with different values then we must use the concept of parameterized constructor.

Syntax:

```
class <classname>
{
  _____
  <classname> (list of formal params) → parameterized
  {                                         constructor
    _____
    Block of Stmt(s);
    _____
  }
  _____
```

→ Write a Java program which illustrate the concept of parameterized constructor.

```
class Test
{
  int a, b;
  Test (int x, int y)
  {
    s.o.p ("I am from parameterized constructor");
  }
}
```

```

    a=x;
    b=y;
    System.out.println("Value of a = " + a);
    System.out.println("Value of b = " + b);
}

class TestDemo2
{
    public static void main(String ar[])
    {
        Test t1 = new Test(10,20);
    }
}

```

### Output

Value from parameterized constructor

Value of a=10

Value of b=20

Rule2: Whenever we create any object with parameterized constructor, defining the parameterized constructor is mandatory

Rule3: Whenever we create multiple objects of any class w.r.t default & parameterized constructors, it is mandatory for the java programmer to define both the constructors.

### Overloaded Constructors:

A constructor is said to be overloaded iff constructor name is same but its signature is different.  
Signature represents the following.

(a) no. of parameters  
 (b) Type of parameters  
 (c) order of parameters } at least one thing must be differentiated

Ex: Test t<sub>1</sub> = new Test (10, 20); —①

Test t<sub>2</sub> = new Test (100); —②

Test t<sub>3</sub> = new Test (10.5f, 20.5f); —③

Test t<sub>4</sub> = new Test (100, 5.5f); —④

Test t<sub>5</sub> = new Test (5.5f, 200); —⑤

here the constructor 'Test' is called overloaded constructor

### Object parameterized constructor:

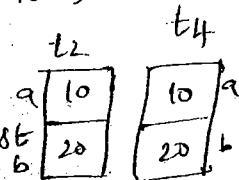
Object parameterized constructor is one which always takes object as a parameter.

The basic purpose of this constructor is to copy the content of one object into another object where both the objects must be of same type.

Object parameterized constructor of Java exactly works like copy constructor of C++ (Java programming does not contain the syntax like copy constructor)

Ex: Test t<sub>2</sub> = new Test (10, 20); —①

Test t<sub>4</sub> = new Test (t<sub>2</sub>); —② // t<sub>2</sub> object Test



⇒ Write a Java program which illustrate the concept of default constructor, parameterized constructor & object parameterized constructors.

class Test

{

int a, b;

Test()

{

S.o.p ("I am from DC");

a=1;

b=2;

S.o.p ("Value of a = " + a);

S.o.p ("Value of b = " + b);

}

Test (int x)

{

S.o.p ("I am from Single Parameterized constructor");

a=x;

b=x;

S.o.p ("Value of a = " + a);

S.o.p ("Value of b = " + b);

}

Test (int x, int y)

{

S.o.p ("I am from double parameterized constructor");

a=x;

b=y;

S.o.p ("Value of a = " + a);

S.o.p ("Value of b = " + b);

}

Test (Test T)

{

a=T.a;

b=T.b;

```
SOP("I am a copy constructor");
SOP("Value of a=" + a);
SOP("Value of b=" + b);
```

{

}

class TestDemo

{

public static void main (String a[])

{

Test t<sub>1</sub> = new Test();Test t<sub>2</sub> = new Test(100);Test t<sub>3</sub> = new Test(1000, 2000);Test t<sub>4</sub> = new Test(t<sub>3</sub>);

}

}

### Verifying profile of a class / interface:

Profile of a class is nothing but collection of data members, constructors and methods. To see the profile of a class / interface we use 'javadoc'

Javadoc is one of the executable / application / tool present in Jdk 1.5 / bin folder

Syntax: javadoc <classname / interface name>

Ex: javadoc Test <

class Test extends java.lang.Object

Output: {

```
    int a;  
    int b;  
    Test();  
    Test (int);  
    Test (int, int);  
    Test (Test);  
}
```

Sy  
→

Note:

1. A Java class can contain a single default constructor and multiple parameterized constructors (overloaded constructors)
2. If we are not writing any constructors in a class then by default the class can contain single system defined default constructor.

'this' Keyword :

'this' is the keyword / implicit object created by JVM and supplied to every Java program for its purposes.

They are.

1. It is pointing to current class object or it contains an address of current class object
2. When ever the formal parameters and data members of the class are similar, there is a possibility that JVM gets an ambiguity (confuse)

In order to differentiate b/w formal parameters and data members of the class , data members of the class must be preceded by a keyword 'this'

Syntax: `this`. current class datamember;  
→ Write a Java program which illustrate the context  
of 'this' keyword.

class Sample

{

int a, b;

Sample (int a, int b)

{

`this.a = a;` } assigning formal parameter values to  
`this.b = b;` } the data members of the class. Here  
`'this'` is mandatory.

`a = a+1;`

`b = b+1;`

→ modifying the values of

{ `this.a = this.a+2;` data members of the class.  
`this.b = this.b+2;` Here 'this' is mandatory

S.O.P ("Val of formal parameter a = " + a);

S.O.P ("Val of formal parameter b = " + b);

}

void display()

{ S.O.P ("Value of a = " + this.a);  
S.O.P ("Value of b = " + this.b); } Here 'this' is  
optional

}

}

class SampleDemo

{ P.S.V. main (String a[])

{ Sample s = new Sample (10, 20);

{ s.display();

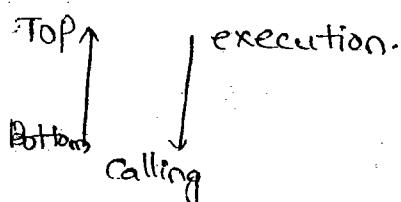
}

## this(), this(---):

In Java programming, we can establish the communication b/w either the methods of same class or the methods of different class or both. Similarly we can also establish the communication b/w constructors of the same class.

To establish the communication b/w constructors of the same class, we have two methods. They are

1. this() ;
2. this(---) ;



### this():

It is used for calling current class default constructor from the parameterized constructors of the same class.

### this(---) :

It is used for calling current class parameterized constructors from other category constructors of the same class.

Rules: When we are using 'this()' and 'this(---)' in the current class constructors, the Java programmer must follow the following sequence of rules.

1. Whenever we use either this(), this(---) in the current class constructors, we must then always as first statement. because when ever an object is creating it is mandatory for the JVM to execute one complete constructor. otherwise we get compile time error.
2. Whenever we are using either 'this()' or 'this(---)' in the current class constructors, it is recommended to

form a path only but not recommended to form cycles  
(loops).

3. We can not use two consecutive either this() or this(--) in the current class constructors. Because every this() or this(--) should be first statement in the current class constructor.

ctrl+G → find

The implementation of the real-time usage of this() or this(--) is for representing different values in the same object during the program execution.

→ Write a Java program which illustrate the concept of this() and this(--)

class Test

{

int a, b;  
Test() // → ③

{  
System.out.println("Test - DC");

a=10;

b=20;

System.out.println("Value of a = " + a);

System.out.println("Value of b = " + b);

}

Test (int x) // → ②

{  
this(); // control goes to ③

System.out.println("Test - SP");

a=x;

b=x;

```

S.O.P ("value of a = " + a);
S.O.P ("value of b = " + b);
}

Test ( int a, int b) // → ①
{
    this(1000); // Control goes to ②
    S.O.P ("Test - DPL");
    this.a = a;
    this.b = b;
    S.O.P ("value of a = " + this.a);
    S.O.P ("value of b = " + this.b);
}
}

class TestDemo
{
    public static void main (String ar[])
    {
        Test t1 = new Test (100, 200);
        // Control goes to ①
    }
}

```

In the above program, constructors are calling in the order of ①, ②, ③ and execution in the order of ③, ②, ①

In general constructors are calling in the order of bottom to top (increasing order) and execution in the order of Top to bottom (decreasing order)

- (Q) Why do you use this() and this--()?
- (A) this() and this--() are used for establishing the communication among the current class constructors  
The real-time implementation of this() and

49

this--> is that to place different values in the same object at regular intervals of time during the program execution.

→ W.A.J.P for sum of two objects

class Test

{  
    int a, b;

    Test()  
    {

        a=10;  
        b=20;

}

    Test(int a, int b)

{

        this.a=a;  
        this.b=b;

}

    TestSum

    Test sum(Test n)

{  
    Test t11 = new Test();  
    t11.a = this.a + n.a;  
    t11.b = this.b + n.b;  
    return t11;  
}

void display()

{  
    S.o.p("Value of a=" + a);  
    S.o.p("Value of b=" + b);  
}

} // Test

— in the —

class objectSum

{ public static void main (String ar[])

{ int x<sub>1</sub> = Integer.parseInt (ar[0]);  
int x<sub>2</sub> = Integer.parseInt (ar[1]);  
int x<sub>3</sub> = Integer.parseInt (ar[2]);  
int x<sub>4</sub> = Integer.parseInt (ar[3]);

Test t<sub>1</sub> = new Test (x<sub>1</sub>, x<sub>2</sub>);

Test t<sub>2</sub> = new Test (x<sub>3</sub>, x<sub>4</sub>);

Test t<sub>3</sub> = new Test ( );

t<sub>3</sub> = t<sub>1</sub> + t<sub>2</sub>; // invalid

t<sub>3</sub> = t<sub>1</sub>.sum (t<sub>2</sub>);

S.O.P ("values of t<sub>1</sub>");

t<sub>1</sub>.display();

S.O.P ("values of t<sub>2</sub>");

t<sub>2</sub>.display();

S.O.P ("Object Sum");

t<sub>3</sub>.display();

} // main()

} // object sum - class.

Note : In the above program

1. t<sub>3</sub> = t<sub>1</sub> + t<sub>2</sub> is not valid, but operator '+' can not be applied on objects. In general, when we apply any operator on two variables they must belong to fundamental-type but not derived data types and user-defined data types. So Java programming does not allow the concept of operators overloading. But it can be easily achieved by the

imple concept called 'methods'

- 2) A method is not only returning fundamental values as a return type, but it can also return pre-defined class names and user defined class names as a return type
- 3) The 'method sum' is one of the factory method.  
 ↓  
 (in the above program)

Factory method :

A factory method is one whose return type must be similar to class where it presents.

The basic purpose of factory method is to create an object without using new operator

In Java programming, factory methods are of two types. They are

- a. Instance factory method
- b. Static factory method.

Consider the following statement

$t_1.sum(t_2)$

Here  $t_1$  = source object, sum = method

$t_2$  = destination object / target object

In the above def of sum function  $t_1$  values will be referred this. Data member of  $t_1$

In general all the source object rules are referred by this. Data member of source object

Point  
al-type  
data  
concept  
ved

$t_2$  values are referred by the sum function n.a and n.b. Here 'N' is called formal object. In general destination / target object values are referred as formal object. Data member of destination object.

12. Write a Java program for adding two String objects

class Test

{

String name;

Test()

{ }

Test(String name)

{

this.name = name;

}

Test add(Test n)

{

Test t1 = new Test();

t1.name = this.name + n.name;

return(t1);

}

void display()

{

s.o.p(name);

}

}

class ObjectAdd

{

public static void main(String a[])

{

String x1 = a[0];

String x2 = a[1];

Test a1 = new Test(x1);

Test a2 = new Test(x2);

Test a3 = new Test(~~x1~~);

// a3 = a1+a2; not valid

a3 = a1.add(a2);

s.o.p("a1 value:");

a1.display();

s.o.p("a2 value:");

a2.display();

s.o.p("String sum:");

a3.display();

}

→ Write a Java program  
to swap the values of  
objects.

I

I  
+

S.

C

D

=

for

to

cl

ki

as

so

E

D

S

its

## Inheritance:

Inheritance is one of the distinct facility in object oriented programming.

The concept of inheritance brought into java language for solving various problems which are generally occurring in software development.

### Def of Inheritance:

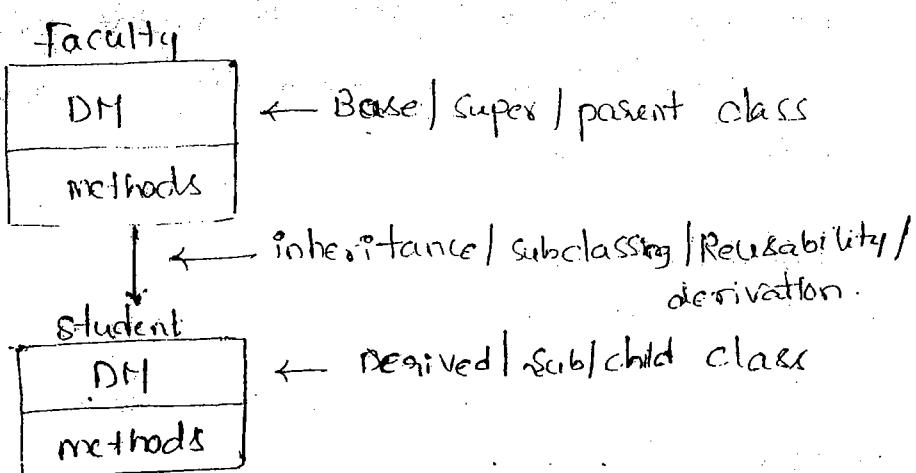
The process of getting the data members and methods from one class to another class is known as inheritance.

The class which is giving data members and methods to another class is known as base/ super/ parent class. The class which is taking data members and methods is known as derived/ sub/ child class.

- \* Data members and methods of a class are known as features

- \* The concept of inheritance is also known as reusability, subclassing / derivation / extendable classes.

Ex:



Def of Derived class: A derived class is one which contains some of the features on its own plus some of the features are inherited from base class.

## Advantages of Inheritance:

When we develop any application with the concept of inheritance, we get the following advantages.

1. application development time is very less.
2. application memory space is less. So that performance of that application is improved.
3. Redundancy (repetition) of the code is reduced. So that work space for the methods is reduced and consistency of the application is achieved.
4. Execution time of the application is less.
5. At all overall the performance of the project is improved.
6. With inheritance concept we are achieving the slogan of Java ie WORA (Write once Run/Reuse Anywhere)

## Inheritance types or Reusable Techniques:

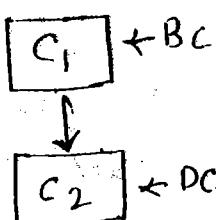
Inheritance types makes us to understand how to get the features from one class to another class.

In object oriented programming we have 5 inheritance types. They are:

### (1) Single inheritance:

Single inheritance is one in which there exists single base class and single derived class.

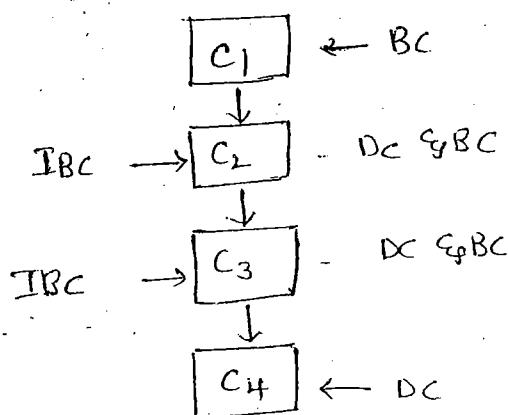
#### Diagram:



Inheritance path  $C_1 \rightarrow C_2$

## 2. Multilevel inheritance:

Multilevel inheritance is one in which there exists single base class, single derived class and multiple intermediate base classes & C. Intermediate base class is one, in one context it acts as base class and in another context it acts as derived class).

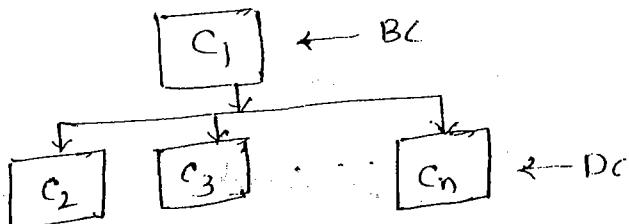


Path :  $C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$

## 3. Hierarchical inheritance:

Hierarchical inheritance is one in which there exists single base class and multiple derived classes.

Diagram

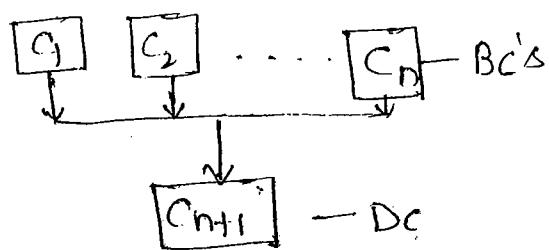


Path :  $C_1 \rightarrow C_2$ ,  $C_1 \rightarrow C_3$ , ...,  $C_1 \rightarrow C_n$

## 4. Multiple inheritance:

Multiple inheritance is one in which there exists multiple base classes and single derived class.

Diagram



Path:  $C_1 \rightarrow C_{n+1}$

$C_2 \rightarrow C_{n+1}$

$\vdots$

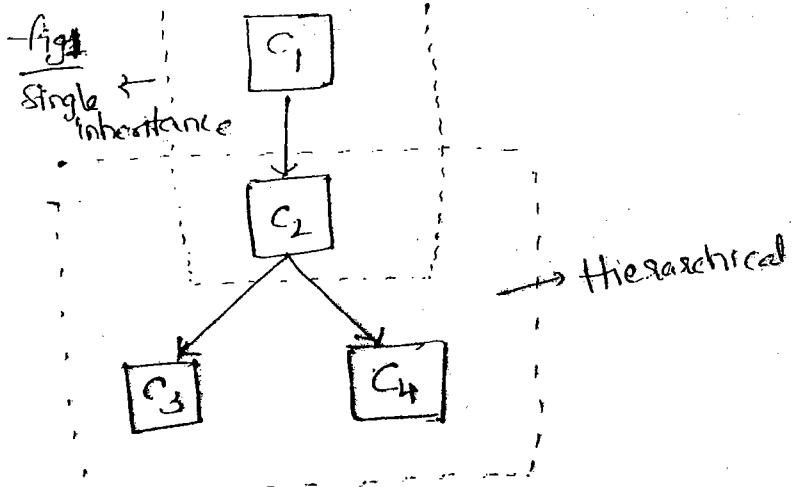
$C_n \rightarrow C_{n+1}$

In Java programming the concept of multiple inheritance is not supported by classes but it is supported by the concept of interfaces.

## 5. Hybrid Inheritance

Hybrid inheritance = Combination of any available inheritance types.

In the combination if one of the combination is multiple inheritance then the entire combination (hybrid inheritance) is not supported in Java by the concept of classes but it is supported by the concept of interfaces.

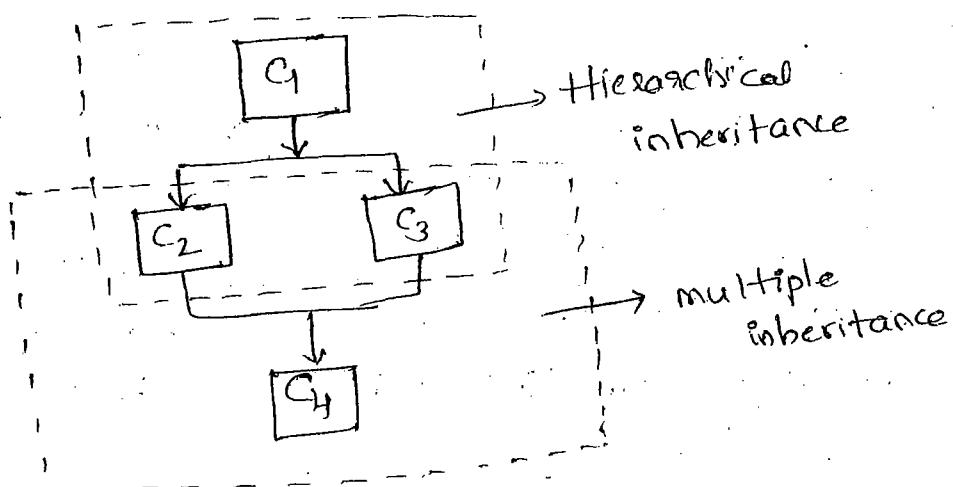


valid by both classes & interfaces

Path:  $C_1 \rightarrow C_2 \rightarrow C_3$

$C_1 \rightarrow C_2 \rightarrow C_4$

Fig:

and  
rept

- Invalid by the concept of classes
- Valid by the concept of interfaces

path :  $C_1 \rightarrow C_2 \rightarrow C_4$

$C_1 \rightarrow C_3 \rightarrow C_4$

ze  
s.

## D) Define workspace:

A) The amount of temporary memory space created by the operating system for execution of the methods in the stack memory is known as workspace.

Syntax for inheriting base class features into derived class:

```

class <clsname-1> extends <clsname-2>
{
    variable declaration;
    method definition;
}
  
```

## Explanation:

In the above syntax

1.  $\langle \text{clsname-1} \rangle$  and  $\langle \text{clsname-2} \rangle$  represents name of the derived and base class respectively.

2. 'extends' is the key word used for inheriting the features of base class to the derived class plus it is improving the functionality of derived class.
3. One derived class can extends only one base class but not multiple base classes because Java programming does not support multiple inheritance through the concept of classes but it can be supported in Java by the concept of interfaces.
4. When we develop any inheritance application, it is highly recommended for the programmer to create an object of bottom most derived class because it contains all the features of intermediate base classes and super most base class.
5. When we create an object of bottom most derived class, first we get the memory space for top most base class data members, later we get the memory space for data members of intermediate base class and at last we get the memory space for bottom most derived class data members.
6. For each and every class in Java, there exist an implicit predefined superclass called java.lang.Object. In other words java.lang.Object is the predefined implicit superclass for ~~the~~ all the classes in Java. Because it provides garbage collector program to the derived classes for collecting unused memory space. So that performance of an application is improved.
7. The constructors of the base class will not be inherited into the derived class. In general constructors never participates in inheritance process.

If we don't want to give the features of base class to the derived class then the definition of the base class must be made as final. In general final base classes never participates in inheritance process.

Q. If we don't want to give some of the features of base class then those features must be made as private. In general private features of the base class will not participate in inheritance process. In other words, a derived class cannot access private features of the base class.

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

```

class bc
{
    int a;
    void disp()
    {
        System.out.println("disp-Base");
    }
}

class dc extends bc
{
    int b;
    void show()
    {
        System.out.println("show-Derived");
        a=100;
        b=200;
        System.out.println("Value of a from bc = "+a);
    }
}

```

```

        .s.o.p(" value of b - from dc = " + b);
    }

    void disp2()
    {
        s.o.p(" Disp2-derived");
    }

} // dc

class Idemo1
{
    public static void main (String a[])
    {
        dc dc1 = new dc();
        dc1.disp();
        dc1.show();
        dc1.disp2();
    } // main()
}

} // Idemo1

```

### Types of relationships in Java:

By Types of relationships makes us to understand how to get the features of one class into another class.

In Java programming we have three types of relationships. They are

1. "is-a" relationship
2. has-a / kind-of / composition relationship
3. uses-a relationship

### "is-a" relationship :

"is-a" relationship is one in which one class gets the features of another class with the concept of inheritance.

along with extends keyword.

Ex:

class c<sub>1</sub>

{

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

  =

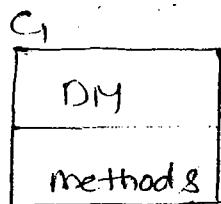
Ex:

class C1

{

=

}



class C2

{

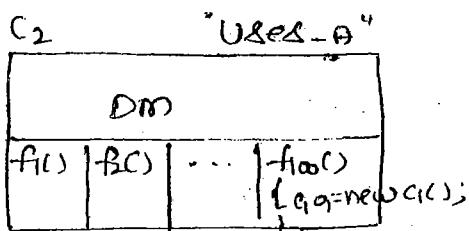
void foo()

}

C1 o1 = new C1();

=

}



### Note :

1. The default relationship in Java is "Is-A" because for each and every class there exist an implicit pre-defined Super class called java.lang.Object.
2. System.out is the universal example for "has-a" (out is an object of PrintStream class created as a static data member in another pre-defined class called System)
3. The universal example for "uses-a" relationship is that "every execution logic method of execution logic class is using an object business logic classes".

### 'Super' Keyword

We know that 'this' is a keyword used for pointing to the current class object and differentiating b/w current class data members and formal parameters.

Similarly we have a keyword called 'super' which will

differentiate b/w data members of base class and data members of derived class.

In other words whenever we inherit the base class features into the derived class, there is a possibility that base class features may be similar to derived class features. So that JVM gets an ambiguity.

In order to differentiate the base class features with derived class features, in the context of derived class base class features must be preceded by a keyword 'super'.

The keyword 'super' playing an important role in Java in three places. They are

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

#### 1. at 'super' at variable level :

When ever we inherit the data members of the base class to the derived class, there is a possibility that base class members may be similar to derived class members. So that JVM gets an ambiguity.

In order to differentiate the base class data members with derived class data members, in the context of derived class base class data members must be preceded by a keyword 'super'.

#### Syntax:

Super. base class DM ;

→ Write a Java program which illustrate the concept of 'super' at variable level

Class bc

{

    int a;  
}

31

ou

>

>

>

Class dc extends bc

{

    int a, c; //a

    void assign( int x, int y)

{

        super.a = x;

        this.a = y; //a=y;

}

    void sum()

{

        //a

        c = super.a + this.a;

}

    void display()

{

        System.out.println("Val of a from bc = " + super.a);

        System.out.println("Value of b from dc = " + this.a);

        System.out.println("Sum= " + c);

}

2.8

-

W

de

a

or

de

bc

,

gr

→ k

or

}

class IdeMod

{ public static void main (String a[])

{  
    int x1 = Integer.parseInt(a[0]);  
    int x2 = Integer.parseInt(a[1]);  
    dc obj = new dc();

of.

```

dot.assign(x, x);
dot.sum();
dot.display();
}

```

The diagram shows a rectangular box labeled "dot" containing the variable "xxx". An arrow points from "dot" to another rectangular box labeled "Super.a". Inside "Super.a", there are three methods: "sum()", "display()", and "c".

### { II Idemo2

#### Output:

```

> javac Idemo.java
> java Idemo 10 20
Value of a -from bc=10
Value of a -from dc=20
Sum= 30.

```

#### 2. 'Super' at method level:

When ever we inherit the base class methods into the derived class, there is a possibility that base class methods are similar to derived class methods. So that JVM gets an ambiguity.

In order to differentiate b/w base class methods and derived class methods, in the context of derived class base class methods must be preceded by a keyword 'super'.

Syntax: super. base class method name

→ Write a Java program which will illustrate the concept of 'super' keyword at method level.

```
class bc
{
    void disp()
    {
        S.o.p("disp - Base");
    }
}
```

```
} // bc
```

class dc extends bc

```
{
    void disp()
    {
        S.o.p("disp - Derived");
    }
}
```

super.disp(); // will call super class disp() from  
the context of derived class disp().

```
} // dc
```

```
class Edemo3
```

```
{
    public static void main(String a[])
    {
        dc d2 = new dc();
        d2.disp();
    }
}
```

// We can not call the base class disp() with respect to  
the derived class object in the context of main()

```
}
```

Output : disp-Derived  
disp-Base.

Q) How do you call a base class method from the context of derived class method when both the methods same?

Ans: super.baseclassmethodname

the above statment must be written with in the context of derived class methods.

Note: In the above program, If we remove 'Super' keyword then we get a runtime error StackOverflowError (recursively called by the derived class function).

Method overriding:

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

or

The process of redefining the original method in various derived classes by inheriting the original method from base class - for performing various operations is known as method overriding.

→ Write a Java program which illustrate the concept of 'super' at method overriding

Class bc

{

void operation( int x, int y)

{

S.O.P("Operation- original");

int k=x+y;

S.O.P("Sum=" + k);

}

} // bc

from  
disp():

?spect to  
main()

```

class dc extends bc
{
    & o.p void operation( int x, int y)
    { super. operation(x,y); // will call super class original
        s.o.p("Operation- Redefined"); } // method from the context
        int k = x-y;                      of derived class over
        s.o.p("Sub= "+k);                ridden// redefined method
    } //
} // dc

```

class idemo4

```

{
    p.s.v. main (String a[])
    {
        int x1 = Integer.parseInt(a[0]);
        int x2 = Integer.parseInt(a[1]);
        dc d03 = new dc();
        d03.operation(x1,x2);
    }
}

```

Output

```

java idemo4 10 20
operation-original
Sum = 30
operation- Redefined
Sub= -10

```

Note: There is no way for us to call super class original method from the context of main() with respect to derived class object.

Q2: In Java programming original methods always reside in base class and overriding and/or redefined methods always resides in derived classes.

a)

Q) How do you call a base class original method from the context of derived class, overridden method?

Ans: Super, original method name

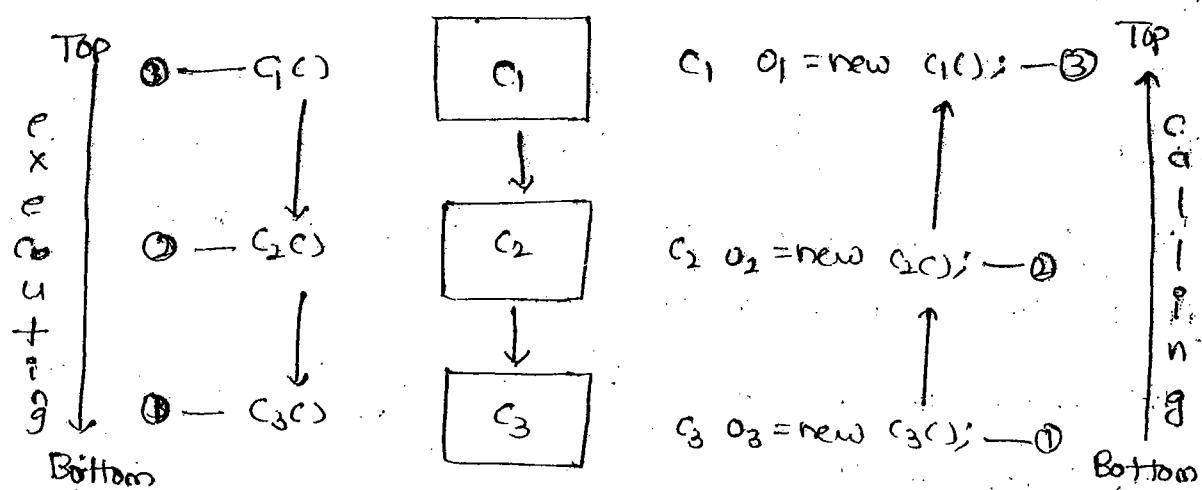
The above statement must be written in derived class overridden method.

### 'Super' at constructor level:

- When ever we develop any inheritance application we always create an object of bottom most derived class because derived class contains features of intermediate base classes and top most base class.
- When we create an object to bottom most derived class, first we get the memory space for the top most base class data members, later we get the memory space for intermediate base class data members and at last we get the memory space for bottom most derived class data members.
- In whatever the order the memory space is creating in the same order initialization will be taken place either by using System-defined default constructor or by using programmer defined constructors.

The above three points makes us to understand constructors are calling from bottom to top (increasing order) and executing from top to bottom (decreasing order)

Let us consider the following diagram



In the above diagram we create an object of C3 class because it is bottom most derived class.

The default constructor of C3 is automatically calling the default constructor of C2 and the default constructor of C2 is automatically calling the default constructor of C1.

The default constructor of C1 will be executed first, later C2 default constructor will be executed and at last C3 default constructor will be executed.

Therefore in the above diagram constructors are calling in the order of ③, ②, ① and executing in the order of ③, ②, ①.

→ Write a Java program which will justify "constructors are calling from bottom to top and executing from top to bottom".

class C1

TOP

C1(C) || —③

{  
S.O.P("C1-bc");

} || C1

class C2 extends C1

{

C2(C) || —②

{ S.O.P("C2-ibc");

}

} || C2

class C3 extends C2

{

C3(C) || —①

{ S.O.P("C3-dc");

y

} || C3

class idemo6

{

P.S. void main (String ar[])

{

C3 o3 = new (3C); // control goes to ①

}

} || idemo6

Output: C1-bc  
C2-ibc  
C3-dc

In order to establish the communication b/w base class constructors and derived class constructors, we have two functions. They are

1. `super()`

2. `super(...)`

1. `super()`:

It is used for calling Super class default constructor from the context of derived class constructors.

2. `super(...)`:

It is used for calling super class parameterized constructor from the context of derived class constructors.

Rules:

1. When ever we use either `super()` or `super(...)` in the context of derived classes, they must be used always as first statement (otherwise we get compile time error)

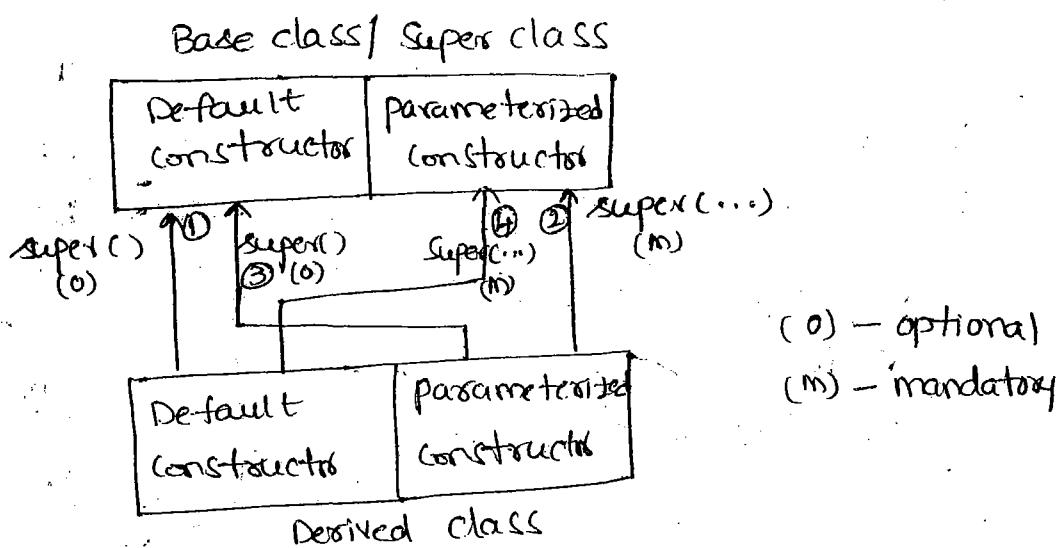
2. We can not use consecutively either `super()` or `super(...)` i.e. no two consecutive `super()` or `super(...)` in the derived class constructors.

\* No. of possibilities to apply `super()` and `super(...)`:

The no. of possibilities of using `super()` and `super(...)` in the Java programming are shown in the following diagram.

? class  
two

or



### Rule ① & ③:

When ever we want to call default constructor of base class from the context of derived class constructor we use super().

Using super() in the context of derived class constructors is optional because every base class contains single default constructor.

### Rule ② & ④:

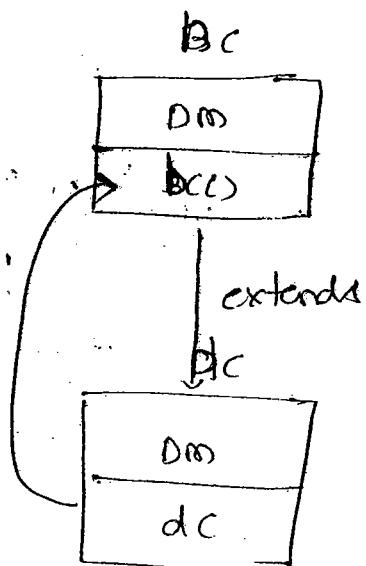
When ever we want to call base class parameterized constructor from the context of derived class constructors we must use Super(...).

Using Super(...) in the context of derived class constructors is mandatory because a base class can contain multiple parameterized constructors (known as overloaded constructors).

→ Write a Java program which illustrates the concept of Rule ① of the above diagram.

Class bc

```
{ int a;  
bc() {②  
{  
S.o.p("bc-dc");  
a=10;  
S.o.p("val of a from bc = "+a);  
}  
}
```



Class dc extends bc

```
{  
int b;  
dc() {①  
{  
super(); // optional - control goes to ②  
S.o.p("dc-dc");  
}  
S.o.p("val of b from dc = "+b);  
}  
}
```

Class TestDemo6

```
{  
P.S.V. main(String ar[]){  
d  
dc d = new dc(); // control goes to ①  
}  
}
```

outputs bc-de

val of a from bc = 10

dc-dc

val of b from dc = 20

Write a Java program which illustrate the concept of rule ② of the above diagram.

```
class bc
```

```
{
    int a;
    bc (int a) // — ②
```

```
{
    this.a = a;
    S.o.P ("base-pc");
```

```
S.o.P ("value of a - from bc = " + a);
```

```
}
```

```
} // bc
```

```
class dc extends bc
```

```
{
```

```
    int b;
```

```
    dc (int x, int y) // — ①
```

```
{ super(); // mandatory - control goes to ②
```

```
S.o.P ("Derived - pc");
```

```
b=y;
```

```
S.o.P ("value of b - from dc = " + b);
```

```
}
```

```
} // dc
```

```
class idemo7
```

```
{
```

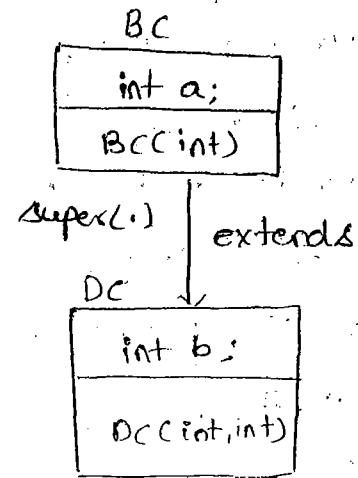
```
    p.s.v.main (String a)
```

```
{
```

```
    dc d01 = new dc (10,20); // control goes to ①
```

```
}
```

```
} // idemo7
```



```

class Idemo9
{
    p.s.v. main (String a[])
    {
        dc d= new dc(); // control goes to ①
    }
} // Idemo9

```

Output : Base\_pc

Value of a from bc = 1000

Derived - dc

Value of b from dc = 2000

In the above programs - the constructors are calling in the order of ①, ②, ③ (bottom to top) and executing in the orders of ③, ②, ① (top - bottom).

Q.E.D. : Write a Java program which illustrate the concept of this(), this(...), super() and Super(...).

```

class bc
{
    bc() // — ⑤
    {
        this(100); // control goes to ⑥
        S.O.P("X");
    }

    bc(int n) // — ⑦
    {
        S.O.P("Y");
    }
} // bc

```

class pbc extends bc

{  
    pbc() // —④ — control goes to ⑤

    {  
        S.O.P("P");

    pbc (int n) // —③

    {  
        this(); // control goes to ④

        S.O.P("Q");

} // pbc

class dc extends pbc

{  
    dc() // —①

    {  
        this(100); // control goes to ②

        S.O.P("M");

} //

dc (int n) —②

{  
    super(100); // control goes to ③

    S.O.P("N");

} //

} // dc

class idemolo

{  
    p.s.v.main(String a[])

{  
    dc d = new dc(); // control goes to ①

} // idemolo

### output

Y

X

P

①

N

M

## Polymorphism:

Polymorphism is one of the distinct facility available in object oriented programming.

Definition: The process of representing one form in multiple forms is known as polymorphism.

Polymorphism is one of the principle and brought into Java or implemented in Java by the concept of method overriding.

### Def. of method overriding:

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

or

The process of redefining the original method into various derived classes by inheriting the original method from base class for performing various operations is known as method overriding.

According to OMG group we have two types of polymorphism they are

1. Static / compile time polymorphism
2. Dynamic / Run time polymorphism.

A static polymorphism is one in which the overloaded methods binds with an object at compile time.

The basic drawback of static polymorphism is that resources are not effectively utilized. For example, if we write any C/C++ program with the concept of method overloading, then when the C/C++ program is compiled in the busy environment like client server, memory space is granted to the methods (work space) at compile time itself. If we are not running this

rogram immediately then we are not utilizing the memory  
object effectively i.e. neither our C++ programmers utilises the  
memory nor other C++ programmers utilise the memory space.

Hence in Java programming static polymorphism is not  
allowed but Java always follows dynamic or runtime polymor-  
phism.

g. Note: C++ supports static polymorphism with the concept of  
method overloading and operator overloading and it also  
supports dynamic polymorphism with the concept of  
virtual functions. In the industry C++ developers always  
follows dynamic polymorphism only even though there  
exists static polymorphism.

A dynamic polymorphism is one in which methods binds  
with an object at runtime.

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

So every Java application is recommended to develop  
with the concept of dynamic polymorphism only.

Java supports dynamic polymorphism through inheritance.

Java supports dynamic polymorphism through interface.

Java supports dynamic polymorphism through abstract class.

Java supports dynamic polymorphism through lambda expression.

Java supports dynamic polymorphism through reflection.

Java supports dynamic polymorphism through annotations.

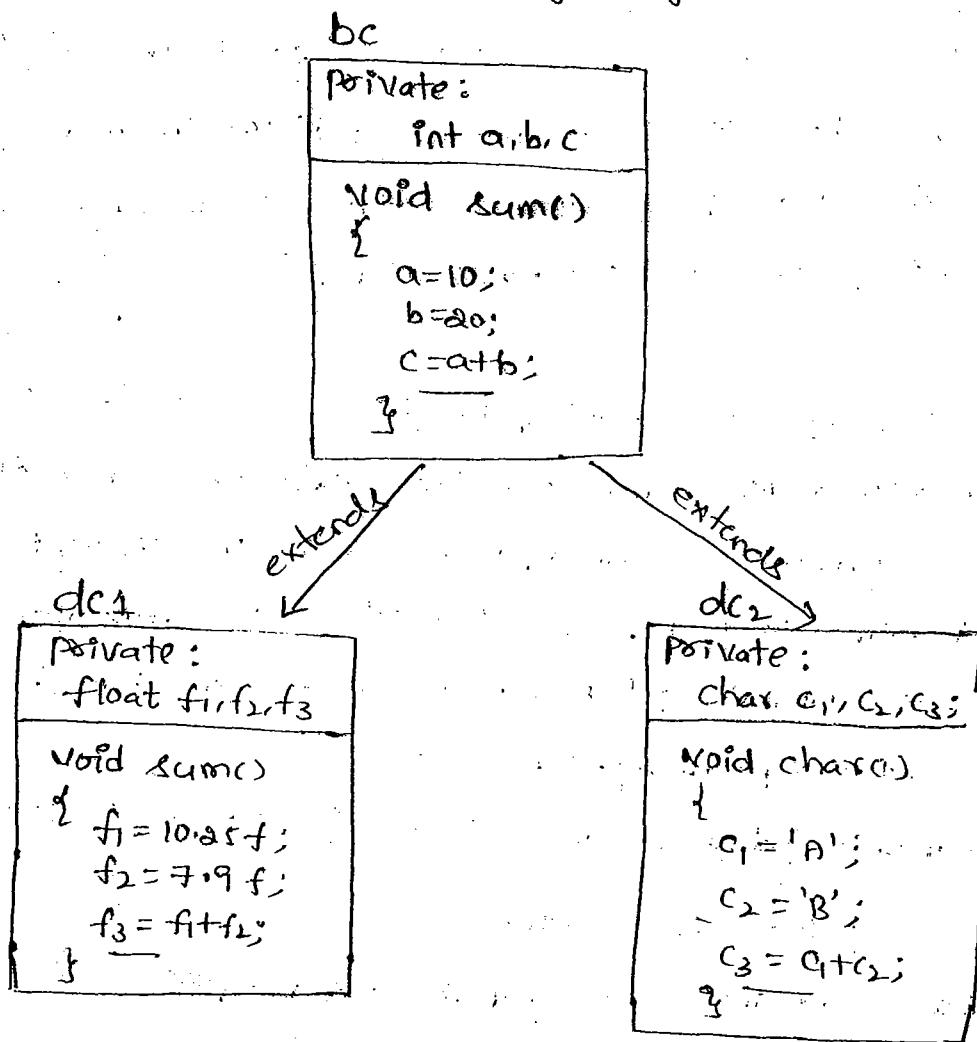
Java supports dynamic polymorphism through exception handling.

Java supports dynamic polymorphism through streams.

Java supports dynamic polymorphism through functional interfaces.

Java supports dynamic polymorphism through annotations.

Let us consider the following diagram.



In most of the real-time applications, a meaningful application will be developed by the concept of polymorphism along with method overriding and such applications are executed by the concept of dynamic binding.

#### Dynamic binding:

- \* Dynamic binding is one of the distinct principle in object oriented programming
- \* Dynamic binding is always used for executing polymorphism applications
- \* Dynamic binding always says that "don't create an object of derived classes but create an object of base class".

The advantage of dynamic binding concept is that to minimize the amount of memory space of an application. So that performance is improved.

While we are executing polymorphism applications with dynamic binding concept, JVM internally considers two points.

a. What type of object?

b. what type of address object contains?

#### \* Def of dynamic binding:

The process of binding appropriate versions (overridden methods) of the derived classes which are inherited from base class with base class object

Consider the following statements which illustrate the concept of dynamic binding (see the above diagram).

① bc bo1 = new bc();

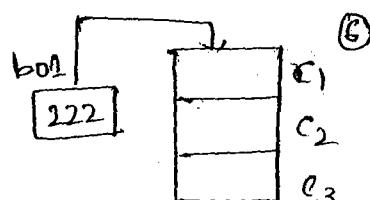
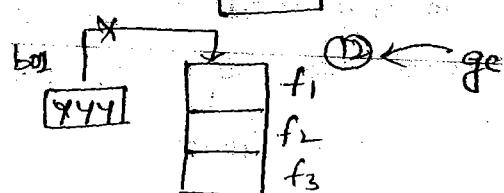
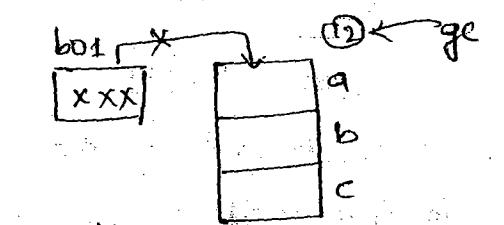
② bo1.sum();

③ bo1 = new dc1();

④ bo1.sum();

⑤ bo1 = new dc2();

⑥ bo1.sum();



In the above statements

The object bo1 contains address of bc, dc1 and dc2.

Hence the object bo1 is known as polymorphic object.

The method sum() is actually available in one form but it is represented in multiple forms. Hence

the method sum() is known as polymorphic method.

b01.sum() is actually only one statement but it gives int sum, float sum and char sum. Hence the statement b01.sum() is known as polymorphic statement.

In most of the real-time applications, applications will be developed with the concept of polymorphism, along with method overriding and such applications will be executed by the concept of dynamic binding.

### Abstract classes in Java:

We know that each and every Java program must be developed with the concept of classes.

In Java programming we have two types of classes. They are

1. Concrete classes
2. Abstract classes

\* A concrete class is one which contains fully defined methods.

\* Fully defined methods of a class are also known as implemented or concrete methods.

\* Once the class is concrete we can create object directly.

Ex: class Test

```
{  
    void f1()  
}
```

```
=====  
y=====
```

```
void f2()  
{  
    =====  
    =====
```

```
5 } // Test
```

The above class called Test contains two defined methods  $f_1()$  and  $f_2()$ . Hence the class Test is known as concrete class so that whose object can be created directly.

```
Test t1 = new Test();
```

```
t1.f1();
```

```
t1.f2();
```

\* An abstract class is one which contains some defined methods and some un-defined methods.

\* Un-defined methods of a class are also known as un implemented or abstract methods.

Def of abstract method:

An abstract method is one which does not contain any body or ~~be~~ definition but it contains only prototype / declaration.

In order to make any undefined method as abstract, that method prototype must be preceded by a keyword 'abstract'.

Syntax for abstract method:

abstract ReturnType methodName (<sup>formal</sup> list of ~~any~~ params if any).

Ex: abstract void operation( int x, int y);

```
abstract void sum();
```

Abstract methods always makes us to understand the following:

a) what a method can do?

but it never gives

b) how a method can be done?

If any class is containing abstract methods then such classes are known as abstract classes. To make that class definition as abstract, we use 'abstract' keyword before the definition of the class.

Syntax for abstract class:

abstract class <className>

{

  =====

  abstract returnType methodName(List of formal params if any);

  =====

}



Ex: abstract class Op

{

  =====

  abstract void operation(int x, int y);

  abstract void sum();

  =====

}

In the above example, the class Op contains two abstract methods. So that creating an object of Op class is not possible directly but it can be possible to create indirectly.

Op op = new Op(); // Invalid

In general once the class is abstract, whose object can not be instantiated (created) directly but it can be instantiated indirectly.

If we develop any application with respect to concrete classes, there is a possibility of increasing memory space so that application performance will be degraded.

To  
con  
Ec  
pol  
m  
sh  
→

1. To strengthen the performance of an application we use the concept of abstract classes.

2. Each and every concept of abstract class makes use of polymorphism and along with method overriding for application development and it will be executed with the help of dynamic binding concept.

→ Write a Java program which will compute sum of two integers and two floats.

Q3:

class operation

abstract class operation

```
{
    abstract void sum();
}
```

class psum extends operation

```
{
    int a,b,c;
    void sum()
    {
        a=10;
        b=20;
        c=a+b;
        System.out.println("Int sum = " + c);
    }
}
```

} H psum

class fsum extends operation

```
{
    int f1,f2,f3;
    void sum()
    {
        f1=10.75f;
        f2=10.25f;
        f3=f1+f2;
    }
}
```

```

        s.o.p ("Float sum = "+f3);
    }
}

class abcdemo1
{
    public static void main (String a[])
    {
        // Operation op = new operation; Invalid, because
        // operation is abstract
        operation op;
        op = new psum();
        op.sum();
        op = new fsum();
        op.sum();
    }
}

```

Output: Int sum=30  
float sum=20.10

In Java programming an object of abstract class cannot be instantiated directly but it can be instantiated indirectly.

\* An object of abstract class = an object of its sub class  
(or)

- An object of abstract class = an object of that class which extends an abstract class  
(or)

\* An object of sub class of abstract class is nothing but an object of abstract class.

### Points to be remember:

If we are giving any common methods for many no. of Java programmers then the common method must be placed in abstract class. In general all the methods of abstract class are called common reusable methods.

2. All the abstract classes in Java must always participate in inheritance process. In other words all the abstract classes of Java are Reusable.
3. Abstract classes should not be final because every abstract class participates in inheritance process. In general there are no final abstract classes in Java.
4. In Java programming, we can declare an object of abstract class but we can not reference an object of abstract class.

Operation op1; // valid - object declaration

op1 = new operation(); Invalid - object referencing

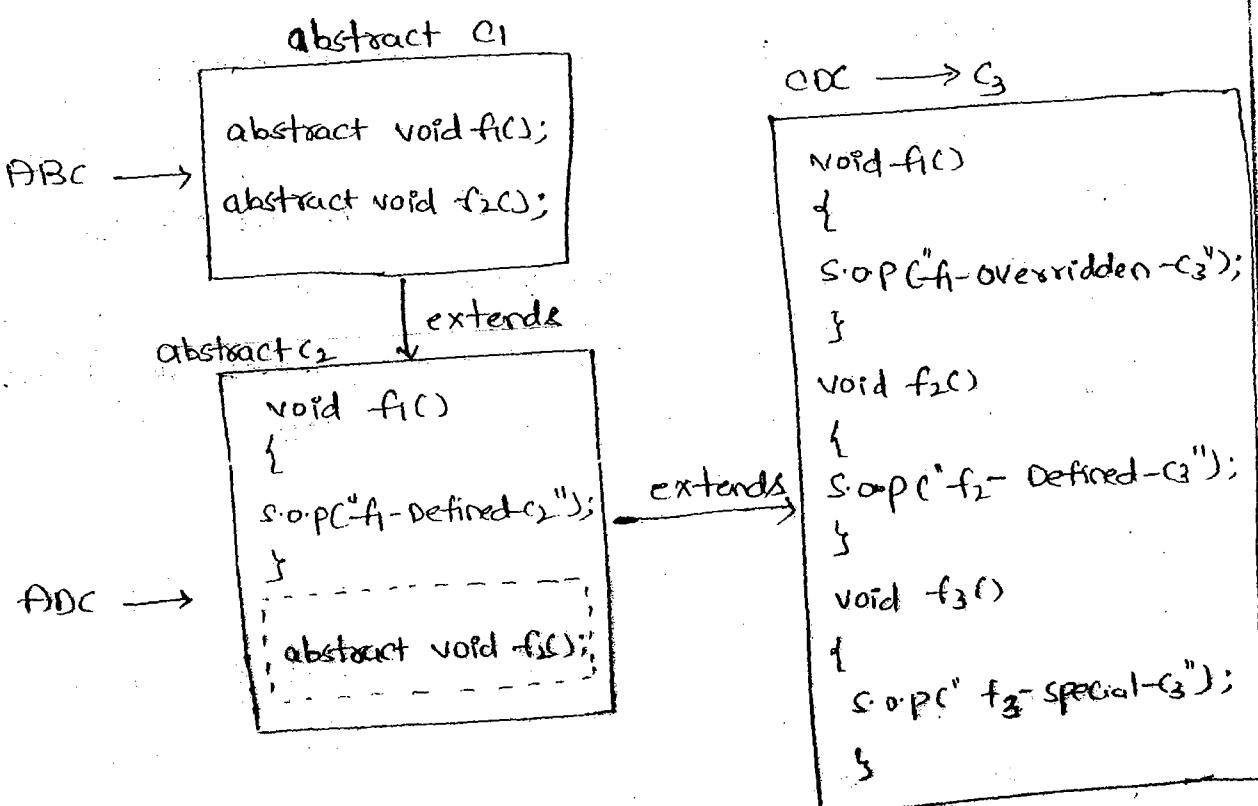
- \* 5. An object of abstract class contains the details about those methods which are available in the same class but an object of abstract class does not contain the details about those methods which are specifically defined in the derived classes.

### Abstract base classes and abstract derived classes:

- \* An abstract base class is one which contains physical representation abstract methods which are always reusable by various derived classes.

- \* An abstract derived class is one which is always containing logical representation of abstract methods which are inherited from abstract base class.

- \* With respect to both abstract base class and abstract derived class one can not instantiate an object directly but we can instantiate their objects indirectly.
- \* Both abstract base class and abstract derived class are always reusable.
- \* When a derived class inherits 'n' no. of abstract methods from the abstract base class and in the derived class not defining atleast one abstract method then the current derived class is known as abstract and whose definition must be made as abstract by using 'abstract' keyword
- \* If the derived class defines all the abstract methods which are inherited from abstract base class then the derived class is known as concrete derived class.
- Write a Java program to implement the following diagram.



**ABC** → abstract base class

**CDC** → abstract derived class

**CDC** → ~~derivative~~ concrete derived class

derived abstract class C<sub>1</sub>

can {

abstract void f1();  
abstract void f2();

like } ||C<sub>1</sub>

abstract class C<sub>2</sub> extends C<sub>1</sub>

like {

void f1();

{

s.o.p("f1-defined-C<sub>2</sub>");

}

} ||C<sub>2</sub>

class C<sub>3</sub> extends C<sub>2</sub>

{

void f1()

{

s.o.p("f1-overridden-C<sub>3</sub>");

}

void f2()

{

s.o.p("f2-defined-C<sub>3</sub>");

}

void f3()

{

s.o.p("f3-special-C<sub>3</sub>");

}

} ||C<sub>3</sub>

class abcDemo2

{

p.s.v.main(String a[])

{

s.o.p("W.A.TO C<sub>3</sub>-inheritance");

$c_3 \circ_3 = \text{new } c_3();$

$\circ_3 \cdot f_1();$

$\circ_3 \cdot f_2();$

$\circ_3 \cdot f_3();$

S.O.P C' w.r.t to  $c_2$  - Dynamic binding");

"  $c_2 \circ_2 = \text{new } c_2();$  invalid, bcoz  $c_2$  is abstract

$c_2 \circ_2 = \text{new } c_3();$

$\circ_2 \cdot f_1();$

$\circ_2 \cdot f_2();$

" "  $\circ_2 \cdot f_3();$  invalid, bcoz  $f_3()$  does not exists  
in  $c_2$

S.O.P C" w.r.t to  $c_1$  - Dynamic binding");

"  $c_1 \circ_1 = \text{new } c_1();$  invalid

"  $c_1 \circ_1 = \text{new } c_2();$  invalid

$c_1 \circ_1 = \text{new } c_3();$

$\circ_1 \cdot f_1();$

$\circ_1 \cdot f_2();$

"  $\circ_1 \cdot f_3();$  invalid, bcoz  $f_3()$  does not  
exists in  $c_1$

}

↳ " abcde $\circ_2$

## Q Consequences of abstract classes:

- Q While we are developing our application we make use of abstract classes and concrete classes.
- Q We know that concrete classes meant for specific purpose and abstract classes are meant for common reusability purpose.
- Q Abstract classes are having three types of consequences.
- Q 1. Is it possible to make a concrete class as abstract class?
- Q A) Yes, possible. Let us assume there exist a concrete class with a common method which is not containing any body. Any method which is not containing body with no block of stmts, such methods are known as null body methods. If any class is containing null body methods that class is also known as concrete class. We know that there is no use of calling null body method with respect to the concrete class object where it present so that it is highly recommended to make such concrete classes as abstract classes.

- In other words if any class is containing null body methods then such classes are highly recommended to make as abstract.
- Write a Java program which illustrate the concept of abstract concrete class (see the above consequence)

abstract class father

{

void go() // null body method  
{

}

} // Father

class son extends father

{

void go()

{ System.out.println("go only for three days"); }

```

    }
}

class abcdemo3
{
    public static void main (String a[])
    {
        // Father f = new Father(); // invalid. because father is abstract
        Father f = new Son();
        f.g();
    }
}

```

Note: Null body methods are always present in base classes and they must be overridden in derived classes.

2. We have a concrete class with collection of concrete methods with block of statements which are of no use. Calling those concrete methods which are of no use with the concrete class object is of no meaning. Hence in Java it is highly recommended to make such concrete classes as abstract.

> Ex: W. a. Java program which illustrate the above consequence.

abstract class Operations

```

{
    void sum()
    {
        System.out.println("I am -from sum() without doing any sum");
    }
}
```

} II operations

```

class isum extends Operations
{
    int a, b, c;

    void sum()
    {
        System.out.println("sum() overridden");
        a=10;
        b=20;
        c=a+b;
    }
}
```

S.OF (" int sum = " + c);

1

Hilium

## class abdemot4

1

P-S-V. main(String a[])

1

Operations a = new Operations(); // invalid, bcoz it is abstract

operation1 o1 = new ~~Fast~~ ISum();

01. sum();

1

Y II abdomen

### Output :-

sums) overridden

Int sum = 30

3. In Java programming it is highly recommended to override abstract methods only and not recommended to override the defined methods because if we try to override defined methods then whose overhead cost is more.

Method overhead cost = destruction of the existing definition + construction of the new definition in the method

In the case of abstract method overriding, we get less method overhead cost because abstract method does not contain definition in the beginning. So that abstract methods requires only construction of new method cost but not destruction cost.

→ Write a Java program which will print Fonts of the system.

```
import java.awt.GraphicsEnvironment;
```

## class fonts

{ P.S. void main (char\* arr, int n) {

```

{
    GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
    String f[] = ge.getAvailableFontFamilyNames();
    System.out.println("No. of fonts = " + f.length);
    for (int i = 0; i < f.length; i++)
        System.out.println(f[i]);
}

```

} 11 fonts

## Interfaces:

We know that abstract class is a collection of defined methods and undefined methods. As a part of our application development, if we use abstract classes then we get the following limitations.

1. abstract classes never participates in multiple inheritance.
2. In the abstract classes, it is highly recommended to override abstract methods only but not recommended to override defined methods. because we know that overriding of defined method takes more method overhead cost (destruction cost + construction cost) and overriding of abstract method requires less overhead cost (only construction cost).
3. All the features of abstract classes are common reusable features only (accessed only within the program) but not universal common reusable (must be accessed in multiple programs).  
To eliminate the above three limitations we use the concept of interfaces.

## Advantages of interfaces:

1. All the interfaces in Java participates in multiple inheritance.
2. Interfaces are containing only purely abstract method and they never allows to define the methods. So that overriding of

- abstract methods is highly recommended.
- all the features of interfaces are belongs to universal Common reusable.
- The concept of interfaces are used for developing user/programmer defined data types. Each and every interface name in Java is one of the user defined datatype.
- \* Interface names are are used for creating objects indirectly but not directly.
- \* To develop the concept of interfaces we use a keyword called 'interface'

Syntax for defining an interface:

```
interface <intfname>
{
    Variable declaration Cum initialization;
    methods declaration;
}
```

Explanation:

In the above syntax

1. interface is the keyword used for developing user/programmer defined data type
- 2.<intfname> represents a java valid variable name treated as name of the interface. All the interface names in Java are used for creating objects indirectly.

In general, in Java programming, one can not instantiate an object of an interface directly but it can be instantiated indirectly.

3. Variable declaration of interface represents data members of the interface. Each and every data member of the interface must be initialized otherwise we get compile time error (we know that initialization is required for universal data members but not for specific data members)

By default all the data members of the interface are belongs to  
public static final xxx data members

Here xxx represents datatype, variable name, variable value  
For example, if we initialize variable a with a value 10 (int a=10)  
then internally JVM is treated as public static final int a

4. Methods declaration represents the type of common methods  
we use to perform universal common operations. Declaring a  
method is nothing but choosing the undefined methods. We  
know that undefined methods of Java are known as abstract  
methods. To make this undefined methods of interface as  
abstract we need not to use or write abstract key word explicitly  
before those methods. and to make these methods as universal  
methods we need not to write public key word explicitly.  
So all the methods of Interfaces are by default belongs to  
public abstract methods

### Def of Interface:

1. An interface is a collection of public static final xxx data members and public abstract methods. or
2. An interface is a collection of universal common reusable data members and universal common reusable methods.

Ex: Define an interface I1 with the data members a and b along with 10 and 20 values respectively, choose some suitable methods for universal functionality.

I1.java

interface I1

{

```
int a=10; // public static final int a;  
int b=20; // public static final int b;  
void f1(); // public abstract f1();  
void f2(); // public abstract f2();
```

Q Compile the above program.

Q >javac i1.java

Q ensure i1.class must be generated.

Q To see the profile of an interface i1

Q >javap i1

Q Interface i1 {

    public static final int a;

    public static final int b;

    public abstract void f1();

    public abstract void f2();

}

Q points to be remember:

1. Interfaces are always containing universal common reusable feature so that they must participated in inheritance process. In other words all the interfaces must be reusable standalone interface are not there in Java).

2. Interfaces are by default abstract i.e it is not necessary for the programmer to make the interface definition as abstract explicitly.

3. Interfaces are not final because all the interfaces in Java always participates in inheritance process.

4. Interfaces should not contain constructors because of the following reasons

a. data members of the interfaces are already initialized

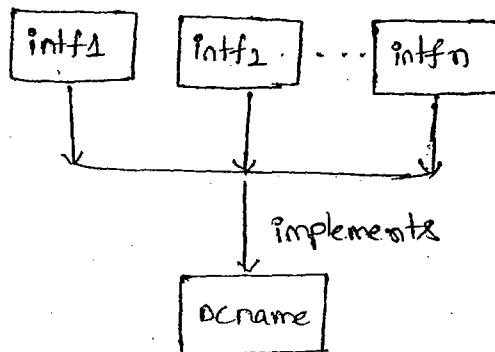
b. Interfaces are the special defined methods, defined things are not permitted in interface definition.

## Inheriting the features of interfaces into class / interface:

In order to inherit the features of interfaces to class / interface, we have three approaches.

### Syntax ① / approach ①:

```
[abstract] class <DCname> implements <intf1>,<intf2>,...<intfn>
{
    variable declaration;
    methods def/declaration;
}
```



### Explanations

In the above Syntax

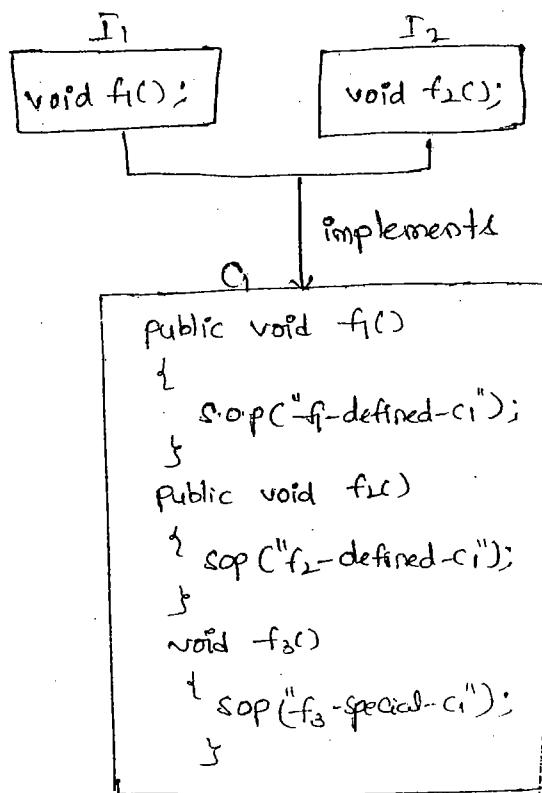
- <DCname> represents name of the derived class
- <intf1>,<intf2>,...<intfn> represents list of interfaces
- 'implements' is the keyword used for inheriting the features of either one interface or more than one interface into the derived class plus it improves the functionality of derived class.

We know that one class can extends only one class but one class can implements either one or more than one interface. because Java does not support multiple inheritance through the concept of classes but it can be supported through the concept of interfaces.

If a derived class inherits 'n' no. of abstract methods from either one interface or more than one interface and if the derived class is not defining atleast one abstract method then the current derived class is known as abstract and whose definition must be made as abstract by using 'abstract' keyword.

If the derived class defines all the abstract methods which are inherited from the interfaces then the current derived class is known as concrete derived class.

Q → Write a Java program which will implement the following diagram.



Interface I1

```

{  
    void f1(); // public abstract void f1();
}
```

Interface I2

```

{  
    void f2();
}
```

Class C1 implements I1, I2

```

{  
    public void f1()  
    {  
        System.out.println("f1-defined-C1");  
    }
}
```

```

    public void f2()  
    {  
        System.out.println("f2-defined-C1");
    }
}
```

```
void f3()
{
    sop("f3-special-e1");
}
```

```
} // q
```

```
class Idemol
```

```
{ public void main (String a[])
{
```

```
    sop("with respect to C1"); // inheritance
```

```
    C1 o1 = new C1();
```

```
    o1.f1();
```

```
    o1.f2();
```

```
    o1.f3();
```

```
    sop("w.r.t p1 dynamic binding");
```

```
// i1, io1 = new E1(); invalid bc02 E1 is abstract
```

```
i1 io1 = new C1();
```

```
io1.f1();
```

```
// io1.f2(); invalid bc02 f2() does not exists in I1 interface
```

```
// io1.f3(); invalid bc02 f3() " " " "
```

```
sop("w.r.t p2 dynamic binding");
```

```
// i2, io2 = new E2(); invalid bc02 E2 is abstract
```

```
i2 io2 = new C1();
```

```
// io2.f1(); invalid bc02 f1() does not exists in E2
```

```
io2.f2();
```

```
// io2.f3(); invalid bc02 f3() does not exists in E2
```

```
} // main
```

```
} // Idemol
```

An object of an interface can not be instantiated directly but it can be instantiated indirectly.

\* An object of an interface = an object of its subclass

- ④ \* An object of an interface = an object of that class which implements that interface.
- or
- \* An object of subclass of an interface is nothing but an object of an interface.
- - An object of interface contains the details about those methods which are available in that interface only but not it contains the details about methods available in other interfaces and the special methods defined in derived classes.
- Like abstract classes, interfaces also makes use of polymorphism and method overriding for application development and dynamic binding concept for execution.

### Approach② / Syntax② ( Interface Inheritance):

The process of getting the features of base interface(s) to the derived interface is known as interface inheritance.

interface <intf-1> extends <intf-2>, <intf-3> ... <intf-n>

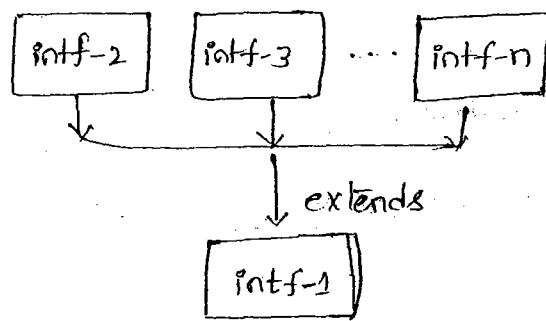
{

variable declaration cum initialization;  
methods declarations;

}

### Explanation:

In the above syntax



- \* <intf-1> represents name of the derived interface.
- \* <intf-2>, <intf-3>, ... <intf-n> represents name of the base interfaces.
- \* 'extends' is the keyword used for inheriting the features of either one interface or more than one interface into the derived interface plus it is improving the functionality of derived interface.

\* One class can extends only one base class and one interface can extends either one or more than one interface.

→ Write a Java program which will implement the following diagram.

interface  $I_1$

```
{
    void f1();
}
```

Interface  $I_2$  extends  $I_1$

```
{
    void f2();
}
```

Class  $G$  implements  $I_2$

```
{
    public void f1()
    {
        System.out.println("f1-defined-G");
    }
}
```

```
public void f2()
{
    System.out.println("f2-defined-C");
}
```

```
void f3()
{
    System.out.println("f3-special-C");
}
```

}

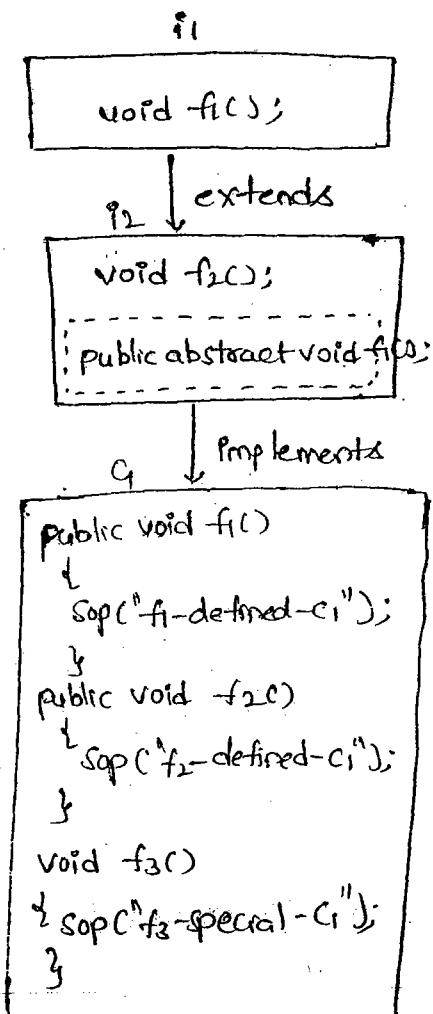
Class  $Indemo2$

```
{
    public static void main(String args)
    {
        System.out.println("W.R to inheritance C");
        C o1 = new C();
    }
}
```

```
o1.f1();
o1.f2();
o1.f3();
```

```
o1.f4();
```

```
System.out.println("W.R to I2-dynamic binding");
if o2.o3 = new I2(); invalid
```



$I_2 \circ_2 = \text{new } G();$

$\circ_2.f1();$

$\circ_2.f2();$

if  $\circ_2.f3();$  invalid, bcoz  $f_3()$  does not exists  $I_2$

// i, o<sub>1</sub> = new i<sub>1</sub>(c); Invalid bcoz o<sub>1</sub> is abstract

i, o<sub>2</sub> = new o<sub>1</sub>(c);

o<sub>2</sub>.f<sub>1</sub>();

// o<sub>3</sub>.f<sub>2</sub>(); Invalid bcoz f<sub>2</sub>() does not exists in i

// o<sub>3</sub>.f<sub>3</sub>(); " " f<sub>3</sub>() " " "

} // in demo<sub>2</sub>

### Syntax (3) / approach (3):

[Abstract] class <Dname> extends <Bname> implements <intf<sub>1</sub>>, <intf<sub>2</sub>>, ... <intf<sub>n</sub>>

{

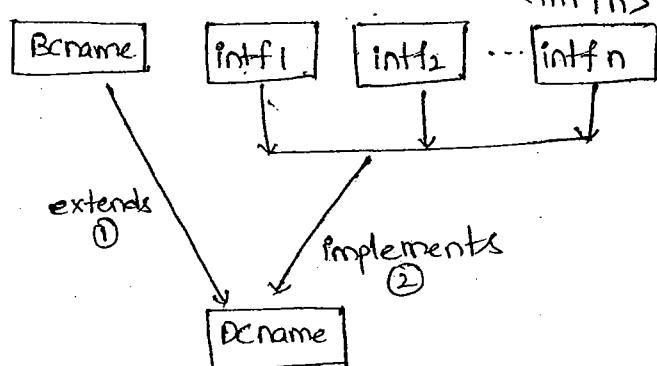
variable declaration;

methods defl declaration;

}

### Explanation:

In the above Syntax,



1. <Dname> and <Bname> represents name of the derived and base classes.

2. 'extends' and 'implements' are the keywords used for obtaining the features of classes and interfaces into the derived class.

3. When we use both extends and implements keyword in a single syntax, it is mandatory for the Java programmer to use extends keyword first and later we use implements keyword. otherwise we get compile time error.

→ Write a Java program which implements the following diagram.

interface i<sub>1</sub>,

{

void f1();

} // i<sub>1</sub>

Interface i<sub>2</sub> extends i<sub>1</sub>,

{

void f2();

} // i<sub>2</sub>

```

    "11
    void f1();
    {
        void f2();
    }
    public abstract void f3();
}

```

abstract class C<sub>1</sub>

```

    {
        void f3();
    }
}

```

abstract C<sub>2</sub> extends C<sub>1</sub>

```

    void f3()
    {
        sop("f3-Redefined-C2");
    }
    public void f1()
    {
        sop("f1-Defined-C2");
    }
    public abstract void f2();
}

```

C<sub>3</sub>

```

    public void f2()
    {
        sop("f2-defined-C3");
    }
    public void f1()
    {
        sop("f1-re-defined-C3");
    }
    void f3()
    {
        sop("f3-re-re-defined-C3");
    }
}

```

abstract class C<sub>1</sub>

```

    {
        void f3();
    }
}

```

3/1/C<sub>1</sub>

abstract C<sub>2</sub> extends C<sub>1</sub> implements i<sub>2</sub>

```

    {
        void f3()
        {
            sop("f3-Redefined-C2");
        }
        public void f1()
        {
            sop("f1-Defined-C2");
        }
    }
}

```

3/1/C<sub>2</sub>

77

```

class C3 extends C2
{
    public void f2()
    {
        System.out.println("f2-defined-C3");
    }
    public void f1()
    {
        super();
        System.out.println("A-Re-defined-C3");
    }
    void f3()
    {
        super();
        System.out.println("f3-Re-Re-defined-C3");
    }
}
class Indemo3
{
    public static void main(String ar[])
    {
        System.out.println("w.r.t C3-Inheritance");
        C3 o3 = new C3();
        o3.f1();
        o3.f2();
        o3.f3();
        System.out.println("w.r.t C2-dynamic binding");
        C2 o2 = new C3();
        o2.f1();
        o2.f2();
        o2.f3();
        System.out.println("w.r.t G-dynamic binding");
        C1 o1 = new G();
        { o1.f1();
        { o1.f2(); } } Invalid bcoz f1(), f2() does not exists in G
        o1.f3();
    }
}

```

```

Sop("w.r.t i2 - dynamic binding");
i2 i02=new C3();
i02.f1();
// i02.f2(); invalid, becoz f3() does not exists in i2
i02.f2();
Sop("w.r.t i1 -dynamic binding");
i1 i01=new C3();
i01.f1();
// i01.f2(); invalid, becoz f2() does not exists in i1
// i01.f3(); " " " f3() "
}

```

\* <sup>Book</sup>  
Dos for Dummies - Tech media

} //indemo3

## PACKAGES

Packages is one of the distinct facility in Java for effective development of applications.

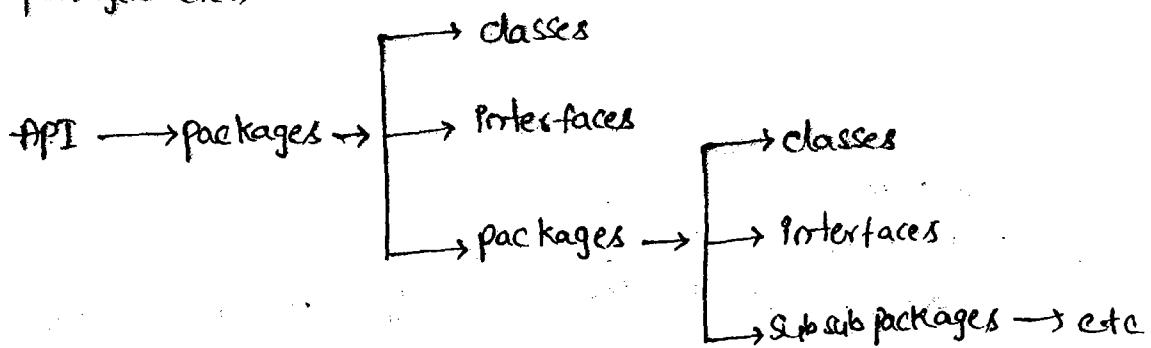
Learning C language is nothing but learning about its concepts plus learning about its library.

Similarly learning about Java is nothing but learning about its API plusoops features and its concepts.

We know that API is a collection of packages.

Def of a package:

A package is a collection of classes, interfaces and subpackages.  
A subpackage in terms contains collection of classes, interfaces and subsub packages etc..



- If any class or interface is common for many no. of Java programmers then those classes and interfaces are recommended to place within the package. In other words common classes and interfaces must be always available within the packages.

### Advantages of Packages:

- If we use package concept as a part of our java program then we get the following advantages.

1. application development time is less
2. Memory space of the application is reduced.
3. Performance of the application is improved
4. Redundancy (repetition) of the code is reduced. So that we get
  - memory cost is less and
  - consistent result
5. Execution time of the application is very less.
6. Packages concept always fulfill the slogan of Sun micro system called WORA.

- Q) What is the difference b/w data sharing using inheritance and packages?

- A) Data sharing using inheritance is possible by default within the program b/w class-to-class, interface-to-interface and interface(s)-to-class

With the concept of packages one can share the data within the program and across the programs b/w class-to-class, interface-to-interface and interface(s)-to-class.

### Types of packages in Java:

In java programming we have two types of packages. They are

1. Predefined / built-in packages
2. User / custom / secondary / programmer defined packages.

### Predefined packages:

Predefined packages are those which are developed by Sun micro system and supplied as a part of JDK to deal with universal requirement.

Some of the universal requirements are displaying the data on the console, accepting command line arguments, providing garbage collector programs etc.

### 1. User defined packages:

User defined packages are those which are developed by Java programmers and supplied as a part of their project to deal with common specific requirements.

### 2. List of predefined packages:

According to industry standards predefined packages are divided into three types. They are

1. Core packages (J2SE)
2. Advanced packages (J2EE)
3. Third party packages (other than Sun micro System)

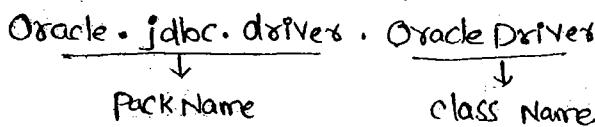
Core packages are used for developing client side applications.

All core packages starts with java.packName.  $\Rightarrow$  java.lang.\*

Advanced packages are those which are used for developing server side applications.

Advanced packages starts with javax.packName.  $\Rightarrow$  javax.servlet.\*;

Third party packages are developed by third party vendors like Oracle Corporation, IBM etc. for strengthening the J2SE and J2EE packages.



### 3. List of Core (J2SE) packages:

In J2SE we have 8 essential packages which are developed by Sun micro Systems.

All the 8 packages of J2SE are compressed into a single file called rt.jar (runtime.java) In other words the total library of J2SE is available in a file called rt.jar.

The file rt.jar is available in JDK1.5\JRE\lib folder.

JRE = JVM + Java API

### → 1. java.lang.\* :-

- This is one of the predefined package used for achieving language functionalities or preliminary basic services of the language.
- Some of the language functionalities are
  - a. displaying the data on the console
  - b. accepting the <sup>data</sup> dynamically from the keyboard or from command prompt
  - c. obtaining garbage collector
  - d. development of multi-threading applications
  - e. data conversions etc.,
- This is the package which is by default imported to every Java program.

### ○ 2. java.awt.\* (Abstract Windowing Toolkit)

This package is used for developing GUI applications (Look and feel based applications are known as GUI applications). In other words, to develop any GUI application we require various classes and interfaces which are present in java.awt.\* package.

java.awt.\* allows us to create GUI Components by using the pre-defined classes.

Creating any GUI Component is nothing but creating an object of appropriate pre-defined class.

```
Button b1 = new Button("Save"); // Save
```

### 3. java.awt.event.\*:

event is one of the subpackage of awt package. event package contains collection of classes & interfaces which will provide functionalities | life|behavior to the GUI Components.

Def of event: The change in the state of the object is known as an event.

Ex: When button is Clicked

When check box is checked, scroll bar is adjusted vertically, horizontally mouse clicked, released etc.,

In order to develop complete GUI application we need to import `java.awt.*` (Component creation) and `java.awt.event.*` (for component functionality).

#### 4. java.io.\* :-

This package is used for developing file handling operations. We know that the concept of files is used for achieving data persistency (Storing the data permanently in the form of secondary storage devices is known as data persistency).

Some of the file handling operations are

- a. opening a file in read mode
- b. opening a file in write mode
- c. inserting a record into the file
- d. deleting a record from the file
- e. modifying a record of the file etc..

file handling operations are also known as Stream handling operation.

A stream is nothing but flow of data in primary memory and secondary memory.

#### 5. java.applet.\* :

This package is used for developing distributed applications. In the earlier days of Sun Micro System, there was a concept called applets for development of distributed applications. To achieve the concept of applets, there is a predefined class called Applet which is belongs to java.applet.\* package (it contains only one class)

- i) Define an applet program ?
- ii) An applet is the Java program which runs in the context of browser / www and whose results are sharable.

#### 6. java.net.\* (Networking) :

This package is used for developing Client/Server applications or networking applications.

⇒ In networking application the programmer has to write two categories of programs. They are

- a. Set of client programs
- b. Set of server programs

In otherwords this package is containing set of classes and interfaces for developing client-server applications.

The drawback of this package is that we are able to develop only intranet applications but we are unable to develop internet application.

(this can be possible by Servlets and JSP of J2EE technology).

## Q) 7. java.util.\* (CollectionFramework) :

This is one of the predefined classes package which contains collection of classes & interface. which are used for improving the performance of Java J2EE application. Specially in client-server application development it gives guarantee of delivery of data b/w two remote machines which are available in diff place and communicating through network.

### Q) Def of Collection framework:

Collection framework is one of the standardized mechanism of Java which allows us to group or integrate multiple values either of same type or different type or both. This in a single variable. This single variable is known as collection framework variable.

Q) What are the differences b/w array and collection-framework?

#### 1) array

#### collectionFramework

\* An array is a collective name given to a group of consecutive memory locations which are all referred by similar type of elements.

\* The concept of arrays allows fixed size or static size in nature.

\* Arrays allows only homogeneous elements but not heterogeneous elements.

\* Collection framework is a mechanism of grouping similar type of elements or different type of elements or both in a single variable, this single variable is known as Collection framework.

\* The concept of Collection framework allows us to add dynamic values i.e. dynamic size in nature.

\* Collection framework allows both homogeneous and heterogeneous element

## 8. java.text.\* (Text processing)

This package is containing collection of classes & interfaces which are used for formatting date and time in the regular transactions which are occurring in the Java development. In other words this package is also used for generating reports for the day-to-day transactions in the organizations.

## User programmed | Secondary | Custom defined packages:

User defined packages are those which are developed by Java programmer and supplied as a part of their project to deal with common specific requirements.

If any class or interface is common for many no. of Java programmers then it is highly recommended to place those common classes and interfaces within the package.

Through the concept of package we can achieve the data sharing within the program and across the programs. In class-to-class, interface-to-interface and interface(s)-to-class.

Creating a User-defined package is nothing but creating it as a directory or folder within the current working machine.

### Syntax for defining a package:

```
package pack1[.pack2[...[.packn]]];
```

### Explanation:

- 'package' is the keyword used for developing user defined packages for placing the classes and interfaces for common access.
- pack1, pack2, ... packn represents each valid variable names treated as user-defined package names.
- pack2, pack3, ... packn represents sub name of the subpackages and whose specification is optional.
- pack1 represents an outer package and whose specification is mandatory.

→ Ex: package P1 ; → ① } package statements  
 package P1.P2; → ② }

Rule: Whenever we place any class or interface in a package, it is mandatory for the Java programmer to write the package statement as first statement in the Java program otherwise we get compile time error.

### Steps or Guidelines for developing user-defined packages

- When we develop any user-defined packages as a part of our Java application, the java programmer must follow the following guidelines
  1. choose an appropriate package and ensure the package statement must be the first executable statement.
  2. choose an appropriate class name or interface name and ensure whose modifier must be public.
  3. The modifier of the constructors of the class must be public (this is not the case with interface).
  4. The modifier of the methods of the class or interface must be public (this is the optional rule in the case of interface).
  5. Which ever class name or interface name belongs to a package, that class name or interface name must be given as a filename with an extension .java.
  6. At any point of time, the java programmer must type either class definition or interface definition within the single window for a package for effective identification of class names and interface name (if we write both class definition and interface definition in a package of single window then JVM gets file names identification problem).

Ex1: Develop a package tp and choose the class name called Test and select suitable constructors and methods.

```

// Test.java
package tp;
public class Test
{
    public Test()
    {
        System.out.println("Test -DC");
    }
    public void disp()
    {
        System.out.println("disp-Test");
    }
}

```

Ex: Develop a package tp and choose an interface Itest and select the suitable methods

```

// Itest.java
package tp;
public interface Itest
{
    void show(); // public abstract void show();
}

```

Syntax for Compiling package classes and interfaces:

In order to compile the classes and interfaces of a particular package, the Java programmers must follow the following Syntax

> javac -d . filename.java ↴

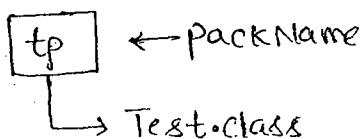
Explanation: (e) '-d' is an option or switch which gives an indication to the JVM saying that goto filename.java, take the package name and create the package name as directory/folder within the current working machine (this directory is known as current directory). provided

- no errors present in fileName.java and packageName should not be created earlier as a directory. If it is already created it is treated as current directory.

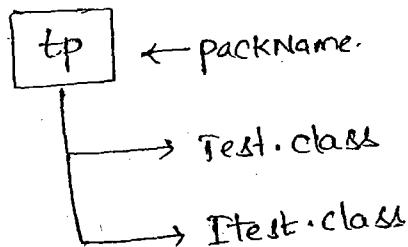
(a) Automatically fileName.java will be compiled and ensure that .class file must be generated.

dot(.) refers current directory in OF. Due to dot(.) , currently generated .class file will be automatically copied into currently created directory (package name).

Ex: >javac -d . Test.java ↴



>javac -d . ITest.java ↴



→ Write a Java program which will make use of tp.Test class  
II packdemo1.java.

```

import tp.Test;
class packdemo1
{
    public static void main (String a[])
    {
        Test t1 = new Test();
        t1.disp();
    }
}
  
```

→ Write a Java program which will makes use of tp.Itest interface

1) packdemo2.java

2) import tp.Itest;

3) class C1 implements tp.Itest

4) {  
5)     public void show()  
6)     {

7)         System.out.println("show-defined-C1");  
8)     }  
9) }

10) class packdemo2

11) {  
12)     public static void main(String a[])

13)     {  
14)         System.out.println("W.r.t C1 - inheritance");  
15)         C1 o1 = new C1();

16)         o1.show();  
17)         System.out.println("W.r.t Itest - dynamic binding");  
18)         tp.Itest io1 = new C1();  
19)         io1.show();  
20) }

21) } II packdemo2.

No. of ways to refer package classes and interfaces:

In Java programming we have two ways to refer the package classes and interfaces. They are

1. using import statement

2. using fully qualified name approach

Using import statement:

'Import' is the keyword used for referring all the classes and interfaces of a specific package or specific class or specific interface of a specific package.

### Syntax 1:

Import pack1 [·pack2 [·... [·packn]]] · \*;

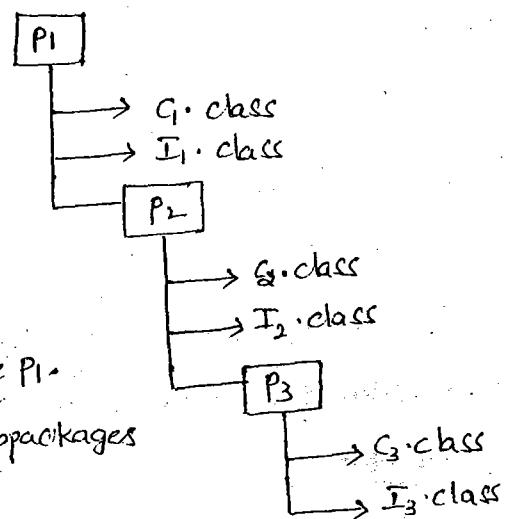
### Explanation:

→ pack1 is an outer package and pack2, pack3, ... packn represents sub packages

→ '\*' is one of the wild character, which gives an indication to JVM saying that bring all the classes and interfaces of a specific package to the current Java program.

### Ex:

- ① import P1 · \*;
- ② import P1 · P2 · \*;
- ③ import P1 · P2 · P3 · \*;



→ Statement ① represents we can refer all the classes and interfaces of package P1 but not the classes and interfaces of subpackages like P2 and P3

→ Statement ② represents all the classes and interfaces of package P2, but not the classes and interfaces of P1 and P3 packages.

### Syntax 2:

Import pack1 [·pack2 [·... [·packn]]] · className / InterfaceName ;

With this syntax we can refer either a specific class or specific interface of a specific package in the current Java program.

- ① import P1.C1;
- ② import P1 · P2 · I2;
- ③ import P1 · P2 · P3 · C3;

Statement ① represents, we can refer the class C1 of the package P1 in the current Java program but not all the classes and interfaces of P1 package.

Stmt② represents, we can refer an interface  $I_2$  of package  $P_2$  but not other classes and interfaces of package  $P_1$ .

## 2. Using FullyQualified Name approach:

FullyQualified Name approach is also known as canonical form approach. the process of referring the class name or interface name with respect to a package is known as FullyQualified Name approach. This approach is an alternative mechanism for import statement but with this approach we can refer either a class or a interface at a time but not all the classes and interfaces of a package.

Syntax:

Pack1[.Pack2[...[.Packn]]].ClassName | InterfaceName

Ex: ①  $P_1.C_1 c_1 = \text{new } P_1.Q(C_2);$

②  $P_1.P_2.I_2 i_{21} = \text{new } P_1.P_2.P_3.C_3();$

In Stmt② the features of  $I_2$  of  $P_2$  package of  $P_1$  package are inherited into the class  $C_3$  of package  $P_3$  of package  $P_2$  of package  $P_1$ .

Model 1 examples:

In model 1 examples, the classes and interfaces of a package can be used anywhere else either within the program or in different programs.

\* We know that data sharing can be possible within the program,

within the package, across the programs and across the package across the programs with the following possibilities

- Class-to-class
- Interface-to-interface
- Interface(s) to-class

Ex: Create a package mp, take a class name multiplication and suitable methods for determining multiplication table for a given number.

```

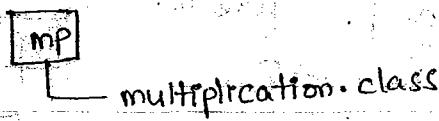
// multiplication.java
package mp;
public class multiplication
{
    int n;
    public void set(int n)
    {
        this.n=n;
    }
    public void table()
    {
        for (int i=1; i<=10; i++)
        {
            System.out.println(n+" * "+i+" = "+(n*i));
        }
    }
}

```

} // multiplication.

Compile the above program.

javac -d . multiplication.java



// packdem03

```

import mp.multiplication;
class packdem03
{
    public static void main(String ar[])
    {
        int x=Integer.parseInt(ar[0]);
        multiplication mo=new multiplication();
        mo.set(x);
        mo.table();
    }
}

```

} // packdem03.

In the above program we use multiplication.class of mp package to compute multiplication table for a given number.

Compile the above program

javac packdem03.java ↵

ensure that dot(.) class file must be generated.

Run the above program

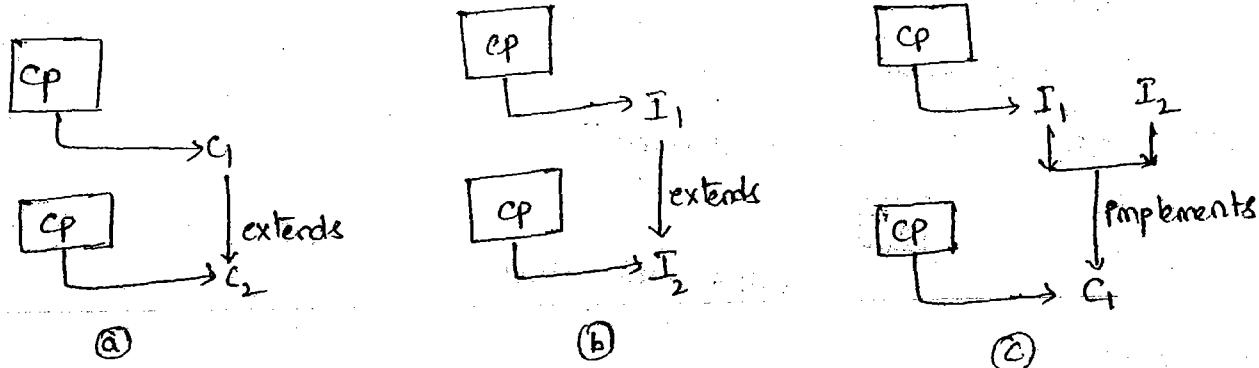
java packdem03 4 ↵

View the result

With the above example we understand that the class or interface of a single package accessed every where else in our application development.

### Model 2 Examples :

Model 2 examples makes us to understand how to access the data within the package across the programs b/w class-to-class, interface-to-interface and interface(s)-to-class.

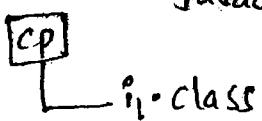


When a derived class / derived interface is inheriting the features from base class or base interface of same package, base class name or base interface name need not to be qualified by package name i.e. we refer directly when all of them belongs to same package.

### Ex:

```
// I1.java  
package cp;  
public interface I1 {  
    void f1();  
}
```

Compile the above java program  
javac -d . I1.java ↵



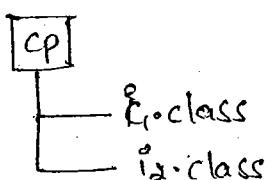
```


    // I2.java
    package CP;
    public interface I2 {
        void f2();
    }


```

Compile the above java program

javac -d . i2.java ↴



// C1.java

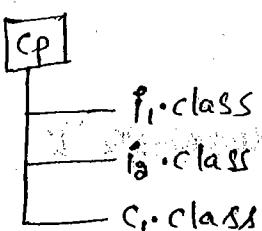
```


    package CP;
    public class C1 implements I1, I2 {
        public void f1() {
            System.out.println("f1-defined-C1");
        }
        public void f2() {
            System.out.println("f2-defined-C1");
        }
    }


```

Compile the above java program

javac -d . c1.java ↴



```

class packdemot4
{
    public main (String ar[])
    {
        System.out.println("E1 - Inheritance");
        CP.C1 o1 = new CP.C1();
        o1.f1();
        o1.f2();
        System.out.println("I2 - dynamic binding");
        CP.I2 i02 = new CP.C1();
        // i02.f1(); // invalid
        i02.f2();
        System.out.println("I1 - dynamic binding");
        CP.I1 i01 = new CP.C1();
        i01.f1();
        // i01.f2(); invalid
    }
}

```

3) II main

3) II Packdemot4

compile the above java program

javac packdemot4.java

java packdemot4

view the results

// PC1.java  
 1) package CP;  
 public class PC1  
 {  
 public void f1()  
 {  
 System.out.println("f1-defined-in-PC1");
 }
 }

javac -d . PC1.java

// PC2.java  
 package CP;  
 public class PC2 extends PC1  
 {  
 public void f2()  
 {  
 System.out.println("f2-defined-in-PC2");
 }
 }

> java -d . PC2.java

```

    " packdemo5.java
    package cp;
    class packdemo5
    {
        public void main (String a[])
        {
            System.out.println ("class-to-class within the same package");
            PC2 o1 = new PC2();
            o1.f1();
            o1.f2();
        }
    }

```

```

> javac packdemo5.java
> java packdemo5

```

(b) II pI1.java

```

package cp;
public interface PI1
{
    void f1();
}

```

```
> javac -d . PI1.java
```

II PI2.java

```

package cp;
public interface PI2 extends PI1
{
    void f2();
}

```

```
> javac -d . PI2.java
```

III: class-to-class within the same package

f1-defined-PI1

f2-defined-PI2

" packdemo6.java

```
import cp.PI1; import cp.PI2;
```

```
class PC2 implements PI2
```

```
{
    public void f1()
```

```
{
    System.out.println ("f1-defined-PI2");
}
```

```

    public void f2()
    {
        System.out.println ("f2-defined-PI2");
    }
}

```

II PC2

class packdemo6

```
{
    public void main (String a[])
    {

```

System.out.println ("Interface(s)-to-class within the same package");

```

        PC2 o1 = new PC2();

```

```

        o1.f1();

```

```

        o1.f2();
    }
}

```

```

    PI2 o1 = new PC2();

```

```

    o1.f1(); PI2 o2 = new PC2();

```

```

    o1.f2(); PI2 o2 = new PC2();
}

```

```

java packdemo6.java

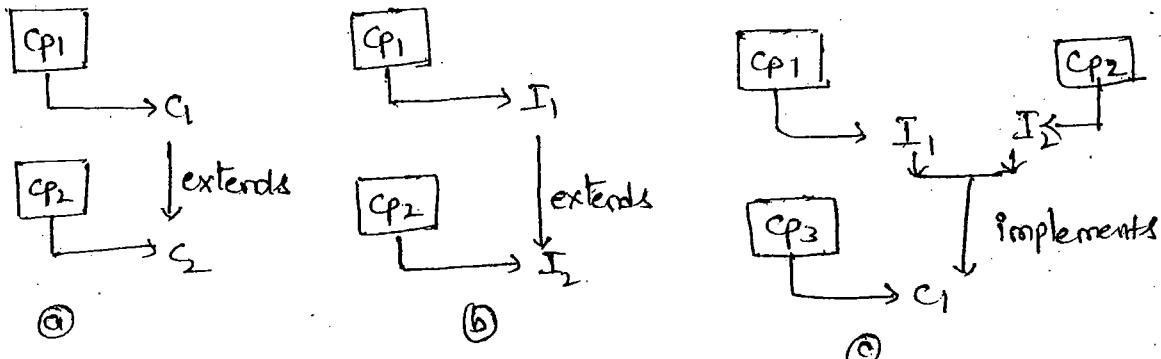
```

java packdemo6 => o1.f1();

java packdemo6 => o1.f2();

### Model 3-examples:

Model 3 examples makes us to understand how to access the data across the packages and across the programs b/w class-to-class, Interface-to-interface and Interface(s)-to-class.

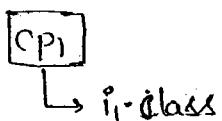


→ Write a Java program to implement diagram (c)

```
// i1.java
package cp1;
public interface i1 {
    void f1();
}
```

javac i1.java // compile the above program

> javac -d . i1.java ↴

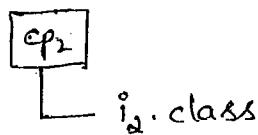


The fully qualified name of `i1` is `cp1.i1`.

```
// i2.java
package cp2;
public interface i2 {
    void f2();
}
```

compile the above java program

> javac -d . i2.java ↵



The fully qualified name of  $i_2$  is  $cp2.i2$

|| C1.java

```

package cp3;
import cp1.i1;
import cp2.i2;
public class C1 implements i1, i2
{
    public void f1()
    {
        System.out.println("f1() - defined - C1");
    }
}
  
```

public void f2()

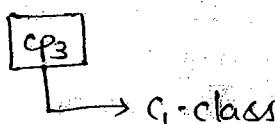
```

{
    System.out.println("f2() - defined - C1");
}
  
```

|| C1.java

Compile the above java program

> javac -d . c1.java ↵



The fully qualified name of  $C1$  is  $cp3.C1$

|| packdem07.java

This program makes use of  $cp1.i1$ ,  $cp2.i2$  and  $cp3.C1$

class packdem07

```

{
    public static void main(String args)
    {
    }
}
  
```

{

Sop("w.r.t - C<sub>1</sub> - Inheritance");cp<sub>3</sub>·C<sub>1</sub> o<sub>1</sub> = new cp<sub>3</sub>·C<sub>1</sub>( );o<sub>1</sub>·f<sub>1</sub>();o<sub>1</sub>·f<sub>2</sub>();Sop("w.r.t i<sub>1</sub> - dynamic binding");cp<sub>1</sub>·i<sub>1</sub> i<sub>01</sub> = new cp<sub>3</sub>·C<sub>1</sub>( );i<sub>01</sub>·f<sub>1</sub>();// i<sub>01</sub>·f<sub>2</sub>(); invalid, f<sub>2</sub> does not exists in cp<sub>1</sub>·i<sub>1</sub>Sop("w.r.t i<sub>2</sub> - dynamic binding");cp<sub>2</sub>·i<sub>2</sub> i<sub>02</sub> = new cp<sub>3</sub>·C<sub>1</sub>( );// i<sub>02</sub>·f<sub>1</sub>(); invalid, f<sub>1</sub> does not exists in cp<sub>2</sub>·i<sub>2</sub>i<sub>02</sub>·f<sub>2</sub>();

} // main

} // pack demo7

Note: If any class is containing main() then that type of classes are known as specific classes and are not recommended to place it in package.

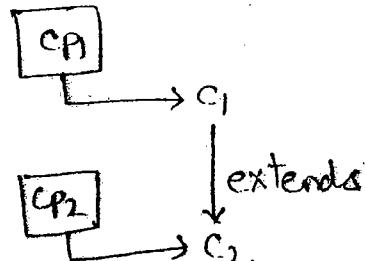
If any class is not containing main() then that class is known as common class and it is recommended to place it in a package.

(a) // C<sub>1</sub>.javaPackage cp<sub>1</sub>;Public class C<sub>1</sub>

{

public void f<sub>1</sub>(){ Sop("f<sub>1</sub>-defined-C<sub>1</sub>"); }

}

} // C<sub>1</sub>

// C2.java

```

package cp2;
public class C2 extends CP1.C1 {
    { super();
        public void f2() {
            System.out.println("f2-defined-C2");
        }
    }
}

```

// Packdemo8.java

```

import cp2.C2;
class Packdemo8 {
    public static void main(String ar[]) {
        C2 o2 = new C2();
        o2.f1();
        o2.f2();
    }
}

```

(b) // I1.java

```

package cp1;
public interface I1 {
    void f1();
}

```

// I2.java

```

package cp2;
public interface I2 extends CP1.I1 {
    void f2();
}

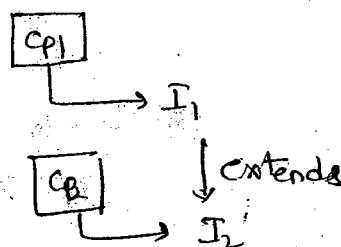
```

// packdemo9.java

```

class C1 implements cp2.I2 {
    public void f1()
}

```



```

    Sop ("f1-defined-C1");
}
public void f2()
{
    Sop ("f2-defined-C1");
}

}

class Packdem09
{
    public static void main (String ar[])
    {
        Sop ("W.R.to inheritance-C1");
        o1 = new C1();
        o1.f1();
        o1.f2();
        Sop ("W.R.to -dynamic binding");
        i1 = new C1();
        i1.f1();
        // i1.f2();
        Sop ("W.R.to -dynamic binding");
        i2 = new C2();
        // i2.f1(); Invalid
        i2.f2();
    }
}

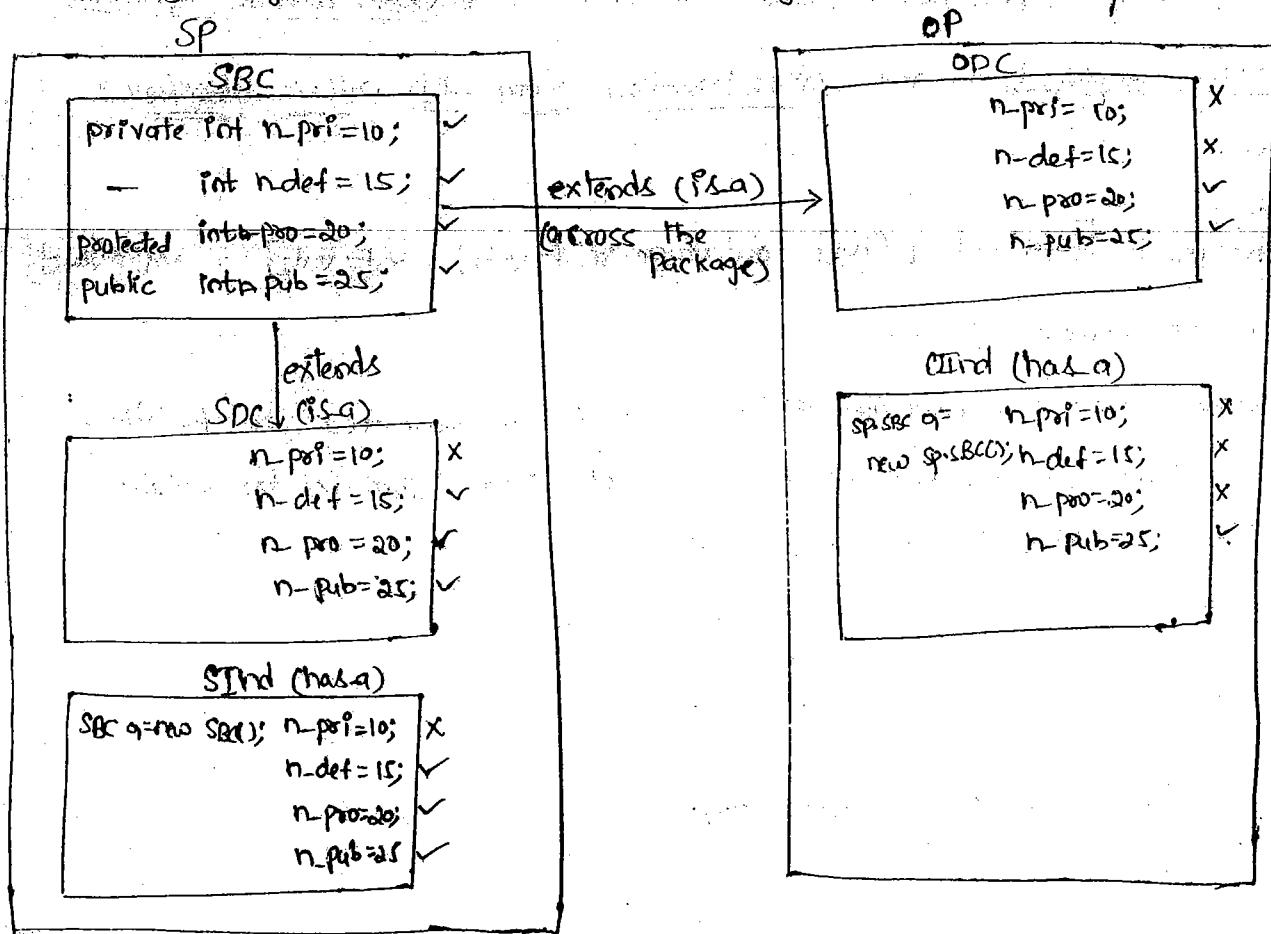

```

## ACCESS SPECIFIERS

- In Java programming access specifiers are always applied before the data members.
- In Java programming we have four types of access specifiers. They are
  1. private
  2. default (not a keyword)
  3. protected and
  4. public
- Access specifiers always makes us to understand how to access the data of a class either within the package or across the package between class-to-class, interface-to-interface and portface(s)-to-class.
- In another words access specifiers always gives the scope of the data or the availability of the data within the package and across the packages.

### Rules for access specifiers:

- The following diagram gives the rules and regulations of access specifiers.



## Access protection matrix :

Type of classes of access specified \ Type of classes	Same package Baseclass	Same package Derived class	Same package Independent class	Other package Derived class	Other package Ind class
private	✓	✗	✗	✗	✗
default	✓	✓	✓	✗	✗
protected	✓	✓	✓	✓	✗
public	✓	✓	✓	✓	✓

### Note :

- \* private access specifier is also known as private access specifier.
- \* default access specifier is also known as package access specifier.
- \* protected access specifier is also known as Inherited access specifier.
- \* public access specifier is also known as universal access specifier.

## Variable declaration and initialization along with access specifier :

### Syntax

[Access specifier] [static] [final] datatype v1 [=val1], v2 [=val2] ... vn [=valn];

In the above syntax,

The access specifier can be either private or protected or public.  
If we don't write these three key words then JVM is by default treated as default access specifier.

Ex:- public static final float PI = 3.1415f;

protected static int x = 20;

int y = 30;

private final int PIN = 4042;

→ Syntax for defining a method along with access specifier:

[Access specifier] [static] [final] returnType methodName (list of formal params if any)  
 {  
 Block of Stmt(s);  
 }

Ex: public static void main (String args [])

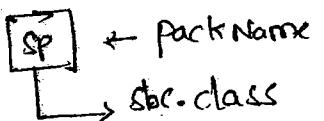
{  
 ==  
 ; };

→ Write a Java program which illustrate the concept of access specifier rules.

```
|| Sbc.java
Package sp;
public class Sbc
{
    private int n_pri=10;
    int n_def=20;
    protected int n_pro=30;
    public int n_pub=40;
    public Sbc()
    {
        System.out.println("Value of n_pri=" + n_pri);
        System.out.println("Value of n_def=" + n_def);
        System.out.println("Value of n_pro=" + n_pro);
        System.out.println("Value of n_pub=" + n_pub);
    }
}
|| bc
|| output;
|| Sbc s1=new Sbc(); -- 10 20 30 40
```

Compile the above program

```
javac -d . sbe.java ↴
```



|| sdc.java

```
package sp;
```

public class Sdc extends Sbc // Is-a relationship within the package.

```
{
```

```
    public Sdc()
```

```
    {
        // Sop("value of n-priv=" + n-priv); not accessible executing
        Sop("value of n-def=" + n-def);
        Sop("value of n-prot=" + n-prot);
        Sop("value of n-pub=" + n-pub);
    }
```

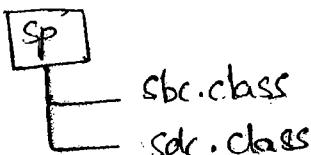
```
}/Sdc
```

|| output:

```
// Sdc s1=new Sdc(); ---- 10 20 30 40 20 30 40
```

compile the above java program

```
javac -d . sdc.java ↴
```

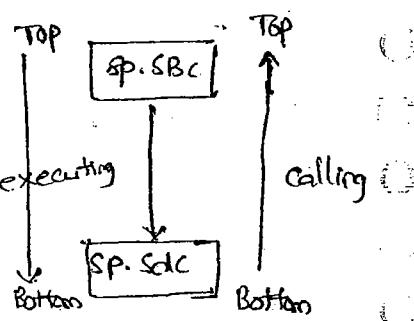


|| sind.java

```
package sp;
```

```
public class Sind
```

```
{
    Sbe s1=new Sbc(); // has-a relationship with in the same pack
```



```

public Sind()
{
    // Sop("Val of n-pri = " + n-pri); not accessible
    Sop("Val of n-def = " + n-def);
    Sop("Val of n-pro = " + n-pro);
    Sop("Val of n-pub = " + n-pub);
}

```

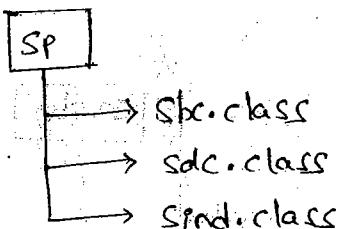
} || Sind

// output:

11. 10 20 Sind s<sub>3</sub> = new Sind(); 11 10 20 30 40 20 30 40

compile the above program

javac -d . Sind.java



// packdemo0

This program makes use of Sp.Sbc, Sp.Sdc and Sp.Sind.

class packdemo0

{

```

    public main(String a[])
    {
        Sop("w.r.t- Sbc");
        Sp.Sbc s1 = new Sp.Sbc(); // 10 20 30 40
        Sop("w.r.t- Sdc");
        Sp.Sdc s2 = new Sp.Sdc(); // 10 20 30 40 20 30 40
        Sop("w.r.t- Sind");
        Sp.Sind s3 = new Sp.Sind(); // 10 20 30 40 20 30 40
    }
}

```

Compile the above program

> javac packdemolo.java ↵

ensure the .class file must be generated.

Run the Java program

> java packdemolo ↵

View the result

11 odc.java

package op;

public class odc extends sp.sbc { It is a relationship across the package

{

    public odc()

{

    // Sop("val of n-pri = " + n-pri); } not accessible

    // Sop("val of n-def = " + n-def);

    Sop("value of n-pub= " + n-pub);

    Sop("value of npub=" + n-pub);

}

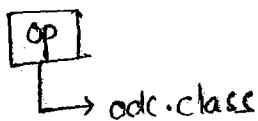
} 11 odc

Output:

11 odc o1 = new odc(); --- 10 20 30 40 30 40

Compile the above program

> javac -d . odc.java ↵



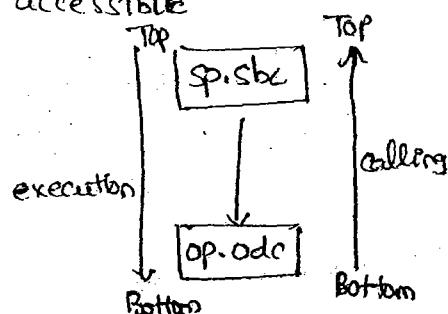
11 aind.java

Package op;

public class aind

{

    sp.sbc s1 = new sp.sbc(); It has-a relationship across the package



```

public oind()
{
    //Op("val of n-pri= "+n-pri);
    //Op("val of n-def= "+n-def);
    //Op("val of n-pro= "+n-pro);
    Op("val of n-pub= "+n-pub);
}

```

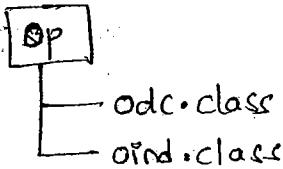
} || oind

|| output:

|| Oind a=new oind(); 10 20 30 40 40

compile the above program

javac -d . oind.java



|| packdem01.java

import op.opcode;  
|| This program makes use of opcode and op.oind

import op.opcode;

import op.oind;

class packdem01

{

Op("n.a.to odc");

odc q=new odc(); // 10 20 30 40 30 40

Op("n.a.to oind");

oind q2=new oind(); // 10 20 30 40 40

}

} Packdem01

Compile the above program

> javac packdem01.java ↴

Ensure that .class file must be generated

run the program

> java packdem01 ↴

view the output

(Q) Can we execute a program which is available in the package with a class and main()?

A) Yes, it is possible to execute

Syntax: Compile the program

↳ javac -d . filename.java ↴

javac -d . packdem01.java ↴

Orithya

packdem01.class

└ main()

↳ Run the program

↳ java packname.className which contains main() ↴

The above syntax can be applicable on all windows operating systems and Unix / Linux operating systems

Syntax2:

java PackName/className which contains main() ↴

This syntax works on windows operating systems only but not on Unix and Linux

Syntax3:

java packname\*className which contain main() ↴

This syntax works on Unix and Linux OS only but not on windows operating systems.

## EXCEPTION HANDLING

- Exception handling is one of the distinct facility in Java programming to make Java as robusted language.
- In any programming language, when ever we write a program, we get two types of errors. They are
  - 1. Compile time errors
  - 2. Runtime errors

Compile time errors are those which are listed or resulted when the programmer is not following syntaxes or grammar of the language at the time of developing the application. In other words if the programmer is having poor knowledge about the language then we get compile time errors.

Runtime errors are those which are resulted or listed when the normal user enters invalid input during execution of the program.

In general runtime errors of Java language are known as exceptions.

In any SW development, If the developer uses C, C++, Pascal, Cobol etc for developing the project then a normal user enters an invalid input to the project, there is a possibility that the entire project will be closed without giving any suitable message. So that the normal user understand that the projects are not developed properly even though the normal user enters <sup>invalid</sup> ~~valid~~ input. This makes us to understand the languages like C, C++, Pascal, Cobol etc., are not strong programming languages because they are not providing a concept called exception handling.

In other words to program in information technology if the normal user enter invalid input then the normal user receives system error message or technical error message which is not able to understand by ordinary user. In the SW industry it is not recommended to display system error messages but highly recommended to display user friendly

Error messages by using the concept exception handling.

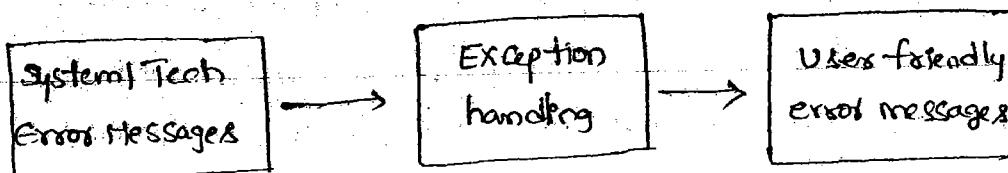
The languages/ technologies like Java and .Net and whose applications are so strong because these are containing a distinct concept called exception handling.

Specially in Java development, every SW engineer will use the concept of exception handling to built robust application by converting system error messages to user friendly error messages.

In Java execution environment a system error message is nothing but exception.

- Whenever we enter invalid input, we get runtime errors. We know that runtime errors are known as exceptions.
- Whenever an exception occurs in any program, program execution is abnormally terminated, control comes out of the program and displays system error messages by default. Which are not recommended.
- Definition of exception handling

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



"Each and every action / task / operation must be performed in Java with respect to an object"

In Java programming we have two types of exceptions. They are

- a. pre-defined / builtin exceptions.
- b. user / programmer / custom / secondary defined exceptions.

Pre-defined exceptions are those which are developed by sun micro systems supplied as a part of JDK to deal with universal problems.

Some of the universal problems are

- a. division by zero problems
- b. invalid bounds of the array
- c. invalid numbers formats etc...

User defined exceptions are those which are developed by Java programme and supplied as a part of the project to deal with common specific problems.

Some of the common specific problems are

- a. Invalid salary of an employee
- b. Invalid age of the human beings
- c. Invalid number of password characters etc.

Predefined exceptions are classified into two types. They are

- 1. Asynchronous exceptions
- 2. Synchronous exceptions.

Asynchronous exceptions are those which deals with hardware problems + external problems.

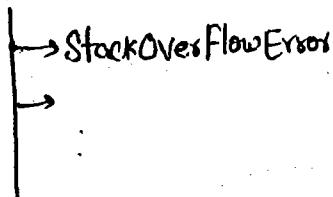
Some of the hardware and external problems are

- a. memory problems } hardware problems
- b. keyboard / mouse / mother board failures }
- c. power failures } external problem.

To handle all the asynchronous exceptions, we have a superclass called java.lang.Error.

As on today, a real-time programmer can not deal with asynchronous exceptions because asynchronous exceptions are not fully developed by SUN MICRO system and they are in R&D of SUN microsystem.

java.lang.Error



R&D → Research & Development

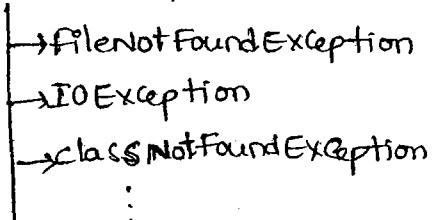
Synchronous exceptions are those which deals with programmatic run time errors. Synchronous exceptions are classified into two types.

a. Checked Exceptions

b. Unchecked exceptions

Checked exceptions are those which are the subclasses of java.lang.Exception.

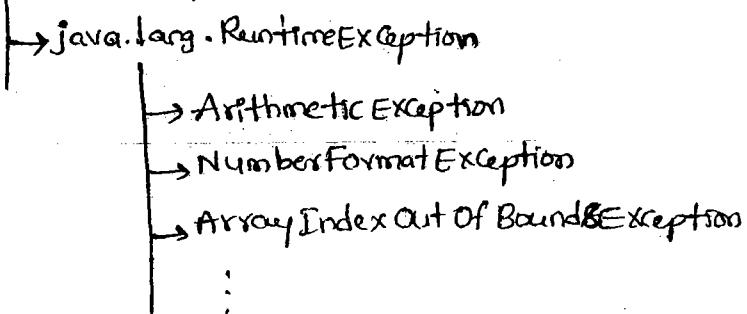
Ex: java.lang.Exception



(Any kind of checking process becomes fail during harddisk checking is known as checked exception.)

Unchecked exceptions are those which are the subclasses of java.lang.RuntimeException.

Ex: java.lang.Exception



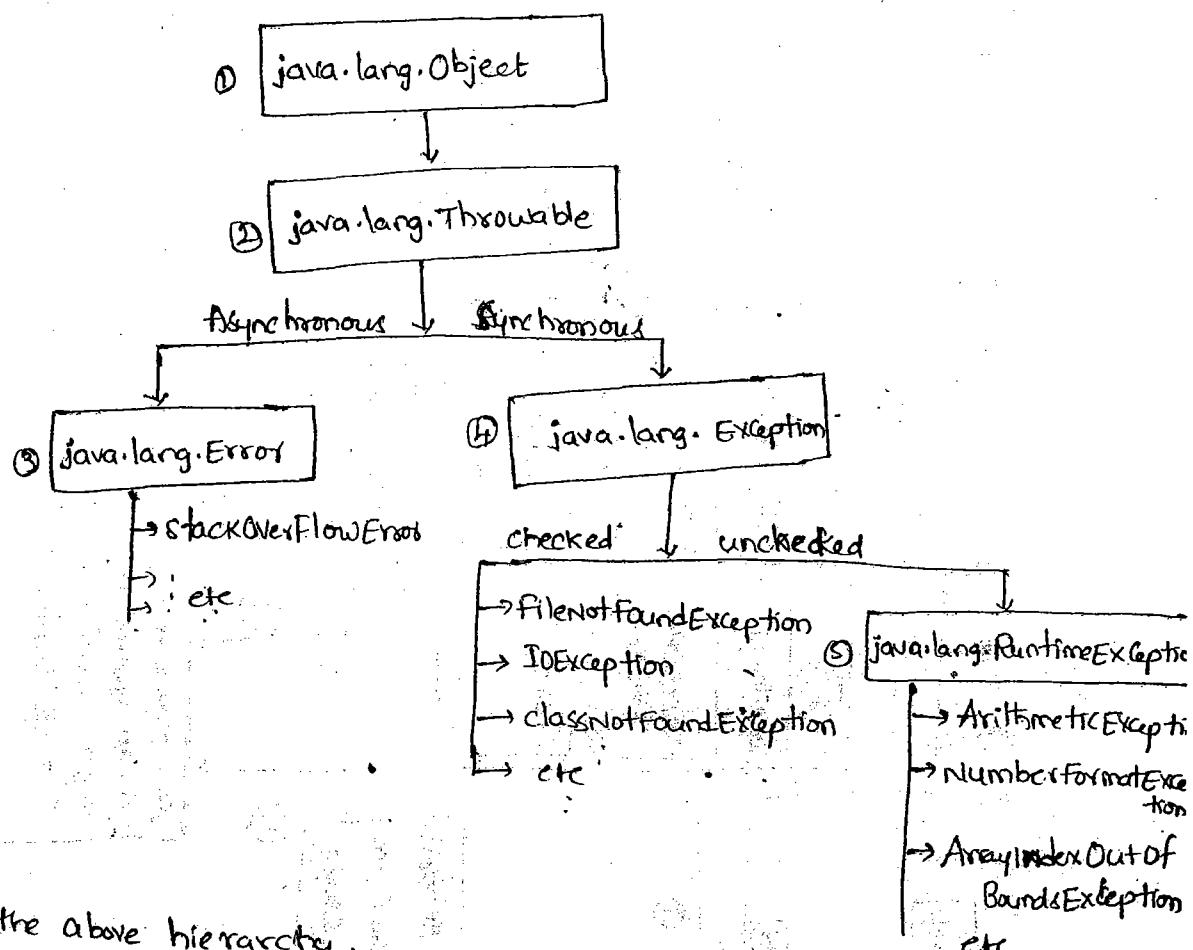
(Any kind of checking process becoming fail during main memory checking, that type of exceptions are known as unchecked exception.)

Note: To deal with almost all the exceptions in Java we have three categories of classes. They are

1. `java.lang.Error` (Asynchronous exceptions)
2. `java.lang.Exception` (Synchronous checked)
3. `java.lang.RuntimeException` (Synchronous unchecked)

## Exception Handling Hierarchy chart:

The following diagram gives various classes used for dealing with various exceptions occurring in the Java program.



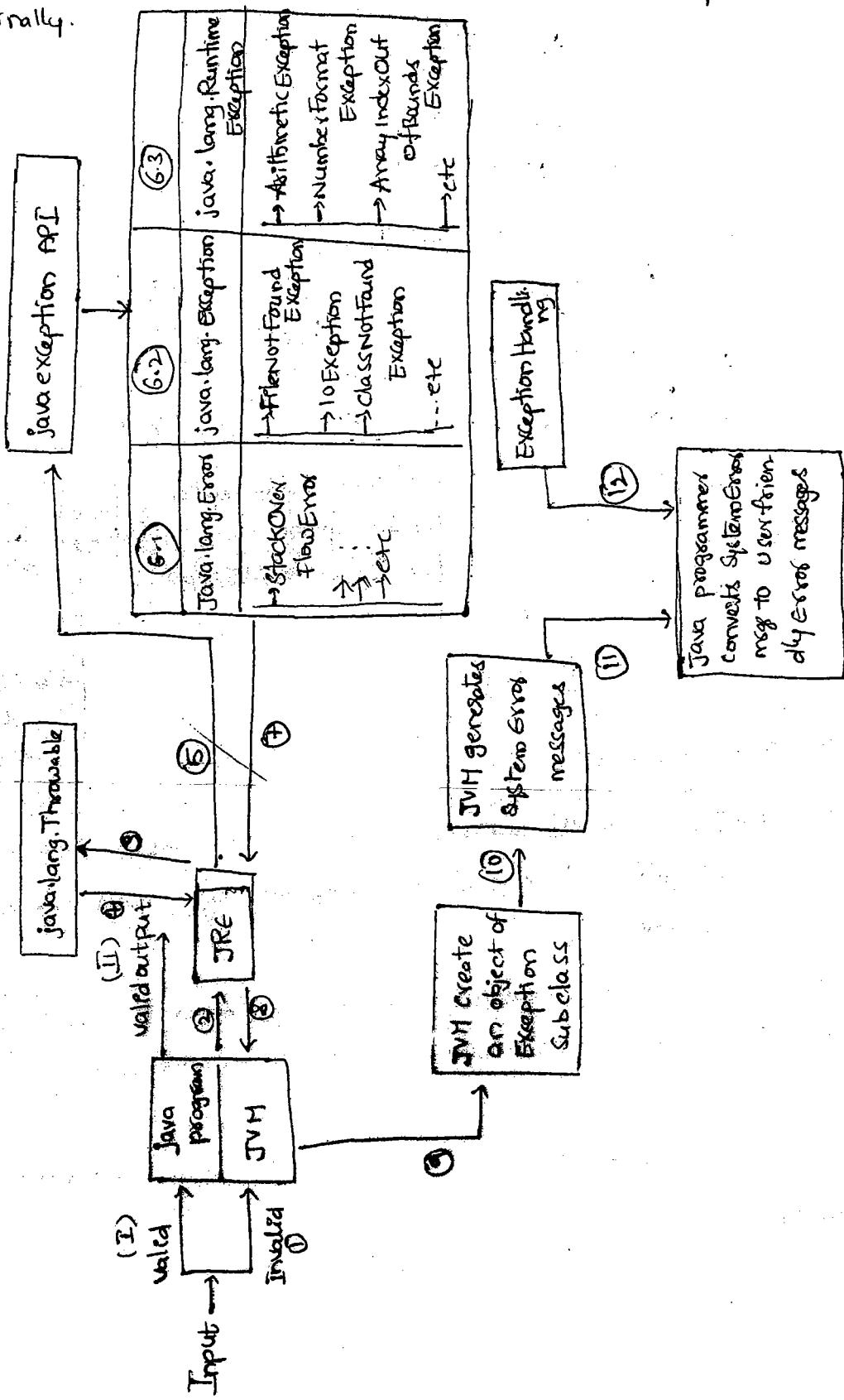
In the above hierarchy,

- ① java.lang.Object is the super class for all the classes in Java
- ② java.lang.Throwable is one of the predefined super class for all the exceptions in Java.
- ③ java.lang.Error is the super class for all asynchronous exceptions
- ④ java.lang.Exception is the super class for all synchronous exceptions (Specially for checked)
- ⑤ java.lang.RuntimeException is the super class for all synchronous unchecked exceptions.

### Internal flow of exception occurrence :

Whenever we write a Java program, there is a possibility of occurring various exceptions when the user enters invalid input.

The following diagram makes us to understand how the exceptions are occurring internally.



### Steps

- (I) End user enters valid input to the Java program.
- (II) Java program processed by JVM effectively and gives valid output
- (1) Normal user enters invalid input
- (2) Java program cannot be processed by JVM and it contacts to JRE for obtaining Exception subclass.
- (3) JRE further contacts to java.lang.Throwable for obtaining the type of exception is which is occurred in the java program.
- (4) java.lang.Throwable is the predefined superclass for all the exceptions in java and it decides the type of exception occurred in java program and gives that message to JRE.
- (5) JRE further contacts to java exception API for obtaining an appropriate exception subclass.
- (6.1), (6.2), (6.3) Java exception API executes either (6.1) (asynchronous) or (6.2) (synchronous-checked) or (6.3) (synchronous unchecked).
- (7) Java exception API gives an appropriate exception subclass of either (6.1) or (6.2) or (6.3) to JRE.
- (8) JRE gives an appropriate exception subclass to JVM
- (9) JVM creates an object of appropriate exception subclass

Ex: For division of two numbers program, if the normal user enters 10 and 0 as an input then JVM creates an object of ArithmaticException programmatically

```
ArithmaticException ae = new ArithmaticException(" /by zero");
          ↓   ↓
        Object Runtime           ↓
                           nature of the message
                           (SystemError message)
```

- (10) JVM generates system error message after the appropriate exception subclass object is created.

Ex: for the above exception

System.out.println("exception in thread main java.lang.ArithmaticException: /by zero")

Generating system error messages are not recommended in the industry.

⑩, ⑪ As a Java program we convert system error messages into user friendly error message by making use of the concept of Exception handling.

Ex: For the above system error message, as a Java programmer we give the following user-friendly error by using exception handling.

Don't enter zero for Denominator → User-friendly error message.

### Technical definition of exception:

An exception is an object occurs at run-time which describes the nature of the message. The nature of the message can be either system error message or user-friendly error message.

Ex: Arithmetic exception ae = new ArithmeticException (" / by zero ");  
↓      ↓      ↓  
Object   Runtime      Nature of the message

### Handling the Exceptions:

When ever we write a Java program, there is a possibility of getting more no. of exceptions. We know that when ever we get the exceptions, by default we use to get system error messages which are not recommended in the software industry.

It is highly recommended to the Java programmers to convert system error messages into user-friendly error messages by making use of the keywords presents in exception handling Keywords Concepts.

Exception handling concept contains 5 keywords. They are

1. try
2. catch
3. finally
4. throws
5. throw

## Syntax:

try

{

Block of stmt(s), which causes problems at runtime

}

Catch (type of exception1 obj1)

{

Block of stmt(s), provides user friendly error messages

}

Catch (type of exception2 obj2)

{

Block of stmt(s), provides user friendly error messages

}

⋮

⋮

Catch (type of exception objn)

{

Block of stmt(s), provides user friendly error messages

}

finally

{

Block of stmt(s), which will execute compulsorily

}

## Explanation:

### try block:

1. It is one of the block in which we write the block of statements which causes problems at runtime.
2. Whenever an exception occurs in Java, program execution will be abnormally terminated and control goes to appropriate catch block for displaying user friendly error message.

3. After executing catch block, control never goes to try block to execute rest of the statements in try block even we use return catch statement in the catch block.
4. one try block can contain at least one catch block that is it is highly recommended for the java programmer to write multiple catch blocks to display / generate multiple user friendly error messages.
5. each and every try block must be followed by one catch block. That is no intermediate statements are allowed b/w try and catch blocks.
6. one try block can contains another try block i.e nested / inner try blocks can be possible.

### Catch block:

1. This is one of the predefined block in which we write the block of statements ,which will provide user-friendly error messages by suppressing system error messages.
2. catch block will execute automatically when ever an exception occurs in try block.
3. At any point of time one catch block will execute out of multiple catch blocks if an exception occurs.
4. In the catch block
  - a. Declaration of appropriate exception class object done by Java programmer.  
Ex: ArithmeticException ae;
  - b. Referencing of the appropriate exception class object must be done by JVM.  
Ex: new ArithmeticException('by zero');
5. one catch block can contain a try block.

## finally block :

1. This is one of the predefined block in which we write the block of statements which will relinquish (release) the resources (file, Data base) which are obtained in try block.
2. finally block will execute compulsorily irrespective of whether the exception has taken place or not
3. Writing finally block is optional
4. Within the finally block one can write try and catch blocks  
Case 1:  
If (no exception occurs) then
  - \* complete try block will execute
  - \* finally block will execute (if we write)

## Catch :

If (exception occurs) then

- \* a part of try block will execute
- \* one appropriate catch block will execute
- \* finally block will execute (if we write)

## 'throws' Keyword :

'throws' is one of the keyword which gives an indication to the calling function to keep the called function under try and catch blocks for generating user-friendly error messages.

In other words 'throws' keyword will establish the bridge between Called function and Calling function.

'throws' keyword always describes the type of exceptions occurring as a part of method body,

'throws' keyword must be always used as a part of method heading.

In SW development, if we develop any common methods for group of Java programmers and if the common method is having various exceptions in its body then it is the responsibility of the developer to express those

exceptions as a part of method heading by using 'throws' keyword. So that which our programmers want to use the common method, they understands that exception related methods must be always kept under try and catch blocks for generating user friendly error messages.

### Syntax for 'throws' keyword

Returntype methodName (list of formal params if any) throws exceptionType<sub>1</sub>,  
..... exceptionType<sub>n</sub>

```
{  
    Block of start();  
}
```

In the above syntax, exceptionType<sub>1</sub>, exceptionType<sub>2</sub>..... exceptionType<sub>n</sub> represents either checked exception or un-checked exceptions or both.

Ex: Let us consider the following

```
java.lang.Integer  
public static int parseInt (String) throws NumberFormatException
```

We understand that the above method is not a normal method but it is one of the exception method because it contains 'throws' keyword and they must be always kept under try and catch blocks.

→ Write a Java program which illustrate the concept of 'throws' keyword for predefined functions (See the above example)

```
class Ex1  
{  
    public static void main (String a[])  
    {  
        try  
        {  
            String s = a[0];  
            int x = Integer.parseInt (s);  
        }  
    }  
}
```

```
System.out.println("val of x = " + x);  
}  
catch (NumberFormatException nfe)  
{  
    System.out.println("Enter numeric int value...");  
}
```

In the above example,

- (1) NumberFormatException raised in parseInt() of Integer class and it is handled in main() of Ex1 class.
  - (2) Hence in general in which our functions exceptions are raising, those functions are called common functions or calling functions and in which our functions we are handling the exceptions, those functions are known as specific functions or calling functions.

→ Write a program which illustrate the concept of try and catch exception

### **Keywords**

## Media.java

## Class div 1

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

ist abr.:

1

```
int a,b,c;
```

float c;

```
a = Integer.parseInt(args[0]);
```

```
b=Integer.parseInt(args[1]);
```

$$C = q/h$$

```
System.out.println("Div = " + c);
```

1

3

```
// div2.java.  
class div2  
{  
    public static void main (String args[])  
    {  
        try  
        {  
            int a = Integer.parseInt (args[0]);  
            int b = Integer.parseInt (args[1]);  
            float c = a/b;  
            System.out.println ("Div = " + c);  
        }  
        catch (ArithmaticException ae)  
        {  
            System.out.println ("Don't enter zero for denominator...");  
        }  
        catch (NumberFormatException nfe)  
        {  
            System.out.println ("Plz enter numerical value");  
        }  
        catch (ArrayIndexOutOfBoundsException nob)  
        {  
            System.out.println ("Plz enter two values");  
        }  
        finally  
        {  
            System.out.println ("I am from finally");  
        }  
    } // main  
} // div2
```

→ Write a Java program which illustrate 'throws' keys for user defined functions.

## // Ex.java

Package ep;

```
public class Ex2
```

O Public void division(String s<sub>1</sub>, String s<sub>2</sub>) throws ArithmeticException, NumberFormatException  
O {

```
int x = Integer.parseInt(s1);
```

```
int y = Integer.parseInt(s2);
```

$$\text{int } z = xly;$$

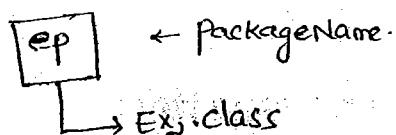
Top C<sup>"</sup> Division in Ex<sub>2</sub> = " + 2).

1

3 || Ex,

11 Compile the above program

```
> javac -d . Ex1.java
```



## 11 Ex3.java

Import Ep. Ex.;

## Class Ex<sub>3</sub>

3

PSV main(String a[])

1

try

۱۷

String s<sub>i</sub> = a[0];

String  $s_2 = a[i];$

$$Ex_2 \circ_2 = new \ Ex_2(j)$$

62. division( $s_1, s_2$ );

```

        catch(NumberFormatException nfe)
        {
            System.out.println("Plz enter numerical int values");
        }

        catch(ArithmeticException ae)
        {
            System.out.println("Don't enter zero for denominator");
        }

        catch(ArrayIndexOutOfBoundsException ab)
        {
            System.out.println("Plz enter two values");
        }
    }

} // main
}

```

### 3) EX3

In the above program exceptions are raised in division(-,-) of cp. Ex2 class and they are handled in main() of Ex3 class and generating user friendly error messages.

#### No. of ways to find the details about unknown exceptions:

When ever we write any Java program, there is a possibility of having many no. of exceptions and these exceptions may not be known to the Java programmer.

SUN Micro System has provided three ways to the Java programmer to find details about unknown exceptions. They are

1. An object of java.lang.Exception class
2. By using printStackTrace()
3. By using getMessage()

#### By using an object of java.lang.Exception:

We know that java.lang.Exception is the super class for all the synchronous exceptions (both for checked and unchecked), it can contain details about

- all the specific exceptions which are occurring in the Java program (dynamic binding) occurring +

- An object of `java.lang.Exception` always displays the following details
    - a. name of the unknown exception
    - b. nature of the message.

Ex: —

```
tay  
{  
    =  
    int c = 100/0;  
    =  
}  
catch (Exception e)
```

System.out.println(e); // java.lang.ArithmeticException : / by zero

↓  
name of the exception

### Nature of message

2. By using printStackTrace():

`printStackTrace()` is one of the predefined method available in `java.lang.Throwable` class and this method further inherited into both `java.lang.Error` and `java.lang.Exception`.

`printStackTrace()` will display the following details when the exception has taken place.

- a. Name of the unknown exception
  - b. Nature of the message.
  - c. Line number where the exception has taken place.

The prototype of printStackTrace() is given below

java.lang.Throwable  
public void printStackTrace()

The above method should not be used as a part of `System.out.println()` and `System.out.println()` because its return type is `void`.

Ex:

```
try {
    1.-
    2.-
    3.-
    4. int c = 10/0;
    5.-.
    6.-
}
```

} catch(Exception e)

{

e.printStackTrace(); // invalid

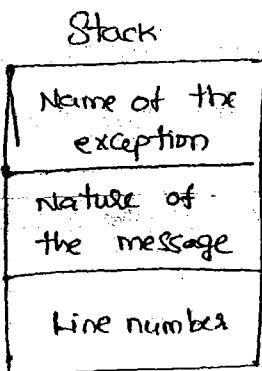
e.printStackTrace(); // java.lang.ArithmaticException : / by zero at line no 4

↓  
Name of the exception

↓  
Nature  
of msg

↓  
Line no

Note: Whenever an exception occurs in Java, the details about the exceptions such as name of the exception, nature of the message and line number will be pushed into the stack by abnormally terminating Java program execution.



### 3: By using getMessage():

getMessage() is one of the predefined method available in java.lang.Throwable and it is further inherited into both java.lang.Exception and java.lang.Error.

By using getMessage() we can obtain only nature of the message when the exception occurs in Java.

The prototype of the getMessage() is given below

```
java.lang.Throwable
    ↓
public String getMessage()
```

Ex:

```
=
try {
    =
    int c = 10/0;
    =
} catch (Exception e) {
    {
        s.e.println(e.getMessage()); // by zero.
    }
}
```

Note: By using this method Java programmers will verify whether the nature of the message is able to understandable by normal user or not. If it is not able to understandable the java programmers must convert the technical nature of the message into a plain user friendly error messages.

#### User / programmer / custom / secondary generated exceptions:

User defined exceptions are those which are developed by Java programmers and supplied as a part of their project to deal with common specific problems.

Some of the common specific problems are.

- Negative age of human beings
- Negative salary of an employee
- Invalid no. of password chars etc..

We know that technically an exception is an object occurs at runtime which will describe the nature of the message. The nature of the message can either a system error message or user friendly error message.

## Steps for developing user defined exceptions :

1. choose an appropriate package and ensure the package statement must be the first executable statement and this package is used for placing user defined exception subclasses for common specific purpose.
2. choose an appropriate user defined class and ensure whose modifier must be public and this class is going to be called as user defined exception subclass.
3. What is the ~~ever~~ the class is selected in the step ② it must extends either `java.lang.RuntimeException` or `java.lang.Exception`.
4. Each and every user defined exception subclass must have a parameterized constructor by taking `String` as a parameter. `String` parameter always represents nature of the message.
5. Each and every user defined exception subclass parameterized constructor must call parameterized constructor of either `java.lang.Exception` or `java.lang.RuntimeException` by using `super(msg)`, here `msg` represents `String` parameter and contains nature of the message. ( If the java programmer is not using the specific exceptions then the specific exception details must be handled by either `java.lang.RuntimeException` or `java.lang.Exception`.)
6. Whichever user defined exception subclass belongs to a package, that class name must be given as a file name with an extension `.java`.

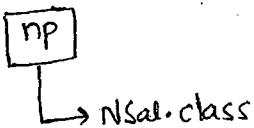
Exhibit a user defined exception subclass to deal with negative salary of an employee

`/* NSal.java */`

```
package np; ①
②
public class NSal extends javaException
{
    ④
    public NSal(String s)
    {
        super(s); ⑤
    }
}
```

→ Compile the above program

> javac -d . NSal.java ↴



→ Ex2: Define or write user defined exception subclass to deal with positive salary of an employee

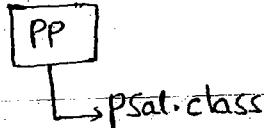
// PSal.java

```

package PP;
public class PSal extends RuntimeException
{
    public PSal (String s)
    {
        super(s);
    }
}
  
```

Compile the above program

> java -d . PSal.java ↴



→ Write a user defined generalized class which will check whether the salary is +ve or -ve.

// Emp.java

```

package org.Sathyas;
public class Emp
{
    public void decideSal(String s) throws NP.NSal, PP.PSal, NumberFormatException
    {
        int Sal = Integer.parseInt(s);
        if (Sal <= 0)
            NP.NSal na = new NP.NSal ("-'ve Salary");
        throw(na);
    }
}
  
```

```
    }  
    else  
    {  
        pp.PSal pa = new pp.PSal("OK, good salary");  
        throw(pa);  
    }  
}
```

```
} // decideSal()
```

```
} // Emp
```

```
> javac -d . Emp.java
```

→ Insite a java program → to check the salary of an individual employee.

```
// SathyasUser.java
```

```
import org.Sathyas.Emp;  
import np.NSal;  
import pp.PSal;  
Class SathyasUser  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            String Sal = args[0];  
            Emp eo = new Emp();  
            eo.decideSal(Sal);  
        } catch (NSal na)  
        {  
            S.e.println("Don't enter -ve salary");  
        }  
        catch (PSal pa)  
        {  
            S.e.println("Good salary, try for best");  
        }  
        catch (NumberFormatException nfe)  
        {  
            S.e.println("Enter only numerical Salary");  
        }  
    }  
}
```

Catch (ArrayIndexOutOfBoundsException a b)

```

    {
        s.e.p("Enter the salary from the console prompt");
    }
    finally
    {
        s.o.p("I am from finally block");
    }
}

```

} || main

} || class

In the above application development

NSal.java and PSal.java are comes under development of user defined exception subclasses and treated as stage-1.

Emp.java is the generalized user defined class which makes use of stage 1 classes and this comes under stage 2.

SathyasUser.java is the user defined specific class which makes use of stage 2 class and this comes under stage 3.

### throw keyword:

'throw' is the keyword which is used for hitting or generating the exceptions which are created as a part of method body.

In other words 'throw' is the keyword which will establish the bridge between method body and method heading.

'throw' keyword must be used as a part of method body.

Whenever we use 'throw' keyword as a part of method body, it is mandatory for the Java programmer to use 'throws' keyword as a part of method heading otherwise we get compiletime error.

### Syntax:

1. Classname objname = new Classname (msg);

    throw (objname);

```

2. return-type methodname( list of formal params if any ) throws <classname>
   <classname>---<classname>()
{
    if (Testcond1)
    {
        <classname1> Obj1 = new <classname-1>(msg)
        throw (obj1)
    }
    :
    if (Testcond n)
    {
        <classname-n> Objn = new <classname-n>( msg )
        throw (objn)
    }
}

```

In the above syntax `<classname1>, <classname2>, ... <classname>` represents name of the user defined exception subclasses.  
`msg` represents nature of the message which is occurred because of the exception.

(Q) What are the differences b/w throw and throws keywords

### throw

① throw is a keyword which is always used for hitting or rising the exception which are occurred as a part of method body.

② throws keyword must be always used with in the method body.

③ Whenever we use throw keyword as a part of method body, it is mandatory for the java programmer to

### throws

① throws is a keyword which always gives an indication to the calling function to keep the called function under try and catch blocks.

② throws keyword must be always used as a part of method heading.

③ When ever we use throws keyword as a part of method heading, it is optional to the java programmer to

- to use throws key word as a part of method heading.
- use throw key word as a part of method body.

## Applets Programming

- o Is one of the distinct facility available in Java programming for development of distributed applications.

- o According to industry standards, as a java programmer we can write two types of applications. They are.

1. Standalone applications
2. Distributed applications

A standalone application is one which runs in the context of local disk and whose results are not sharable.

Every standalone application by default contains main() for execution of the java program and System.out.println() to display the result of standalone application on the console. Hence main() is one of the lifecycle method of standalone applications.

A distributed application is one which runs in the context of browser/www and whose results are sharable.

All the distributed applications contains lifecycle or loopback methods for its execution and some predefined method to display its result on the browser.

### Def. of Lifecycle methods :

LifeCycle methods are those which are automatically called by either by browser or by server.

In real industry all the distributed applications resides in Server.

Based on the place where distributed are running, the programs of distributed applications are classified into two types. They are

1. static web resource programs / client side programs

2. Dynamic web resource programs / Server side programs.

A static web resource program is one which resides in the context of server and running in the context of browser.

Ex: Html, Dhtml, javascript, VB script etc., are treated as static web resource technologies.

A dynamic web resource program is one which are always residing in the context of server and also running in the context of server.

Ex: applets, Servlets, Jsp etc..

In our real world applications static web resource programs will carry the data from client and hand over to the dynamic webresource programs and dynamic web resource programs will validate client data and gives appropriate response to the client.

In the initial days of SunMicroSystem, we have a concept called applets for development of distributed applications.

To develop the distributed applications by using applets concept, we have a package called java.applet.\* and this package contains a predefined class called Applet.

In other words in order to deal with applet programming we must import a package called java.applet.\*

Def. of applet:

An applet is the java program which runs in the context of www/browsers and whose results are sharable across the globe.

Since applet applications are running in the context of browser/www and it contains life cycle or loop back methods.

## Life Cycle or Loopback method in Applet class:

Life cycle methods of applet are divided into 4 types. They are.

1. public void init()
2. public void start()
3. public void stop()
4. public void destroy()

### public void init():

→ This is one of the predefined life cycle method present in Applet class.

→ This method called by the browser only once when the application loaded in the browser or we make a first request to distributed application.  
 → In this method we write the block of statements which will perform one time activity such as opening a file either in read mode or write mode, obtaining database connection, initialization of parameters etc..

→ In general init() is always containing the block of statements which are used for obtaining the resources (files, databases, etc.).

Note: init() by default the null body method.

### Public void start():

→ This is one of the predefined life cycle method present in Applet class.

→ When we make first request, init() will be executed. After executing init() start() method also will be executed. From second request to further subsequent requests, start() will be executed by the browser / server. Hence start() will be executed each and every time as and when we make a request.

→ In start() we write the block of statements which will perform repeated operations such as reading the records from the file, reading the records from the DB, modifying initialized parameters etc.,

→ In general start() always contains block of statements which will provide business logic.

### public void stop():

It is also one of the predefined life cycle method present in applet class. This method will be called by the browser/server when the window is minimized where the applet application is running.

→ In stop() we write the block of statement which will perform the operations like closing the database connections, files extra temporarily.

→ In general in stop() we write the block of statements which will relinquish (release/close) the resources (files, DBS) temporarily.

### public void destroy():

→ It is one of the predefined life cycle method present in applet class.

→ This method will be called by the browser/server when the browser window is terminated or closed where the applet application is running.

→ In this method we write the block of statements which will release the database connections, closing of the files permanently.

→ In general, in destroy() we write the block of statements which will relinquish the resources permanently.

Hence all the above life cycle methods defined in the Applet class with null body. In order to develop our own applet program, we must choose our own class, it must extends java.applet.Applet class and over ride the required life cycle methods.

Ex: import java.applet.\*;

public class MyApp extends Applet

{  
---  
---}

// over ride the life cycle methods of Applet

}  
---

→ Which our class is extending `java.applet.Applet` class, whose modifier must be public otherwise we get runtime error.

### ○ Displaying the result of an applet on the browser:

To display the result of an applet we use the following pre-defined method

```
java.applet.Applet
    ↓
    public void paint(Graphics){ }
```

`paint()` is automatically called by the browser after execution of `start()` because the result of an applet is ready after execution of `start()`. When the control comes to `paint()`, JVM will automatically instantiate an object of `Graphics` class which is present in `java.awt.*` package.

In `Graphics` class we have the following pre-defined method, which is displaying the result of our applet on the browser.

```
java.awt.Graphics
    ↓
    public void drawString(String, int, int)
```

Here,

the parameter `String` represents result of an applet that is result of an applet must be always displayed in the form of `String`.

The first int parameter represents the x-coordinate of the display area of the browser window.

The second int parameter represents the y-coordinate of the display area of the browser window.

On overall the result of the applet (`String` data) must be always displayed in the intersection point of `x` and `y` coordinates of the display area of the browser.

→ Write an applet program which will display a message "Hello Applet".

// MyApp.java

```
import java.applet.Applet;
import java.awt.Graphics;
```

```
public class MyApp extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("HelloApplet", 100, 100);  
    }  
}
```

Compile the above program

> javac MyApp.java +

Ensure MyApp.class must be generated.

No. of ways to run the applet program:

applet is one of the concept which can be developed in all high level languages, middle level languages and low level languages. In java programming applet concept developed by using a predefined class called java.applet.Applet class.

applet programs can be executed in two ways. They are

1. By writing HTML program
2. By using appletViewers tool

By writing HTML program:

applets are written in Java language. So that java programming related logic is not able to understand by browser softwares which are available in the real world. In general browser softwares never understands the logic of high level, middle level and low level languages but all the real world browsers understands only markup languages like SGML, HTML, DHTML etc.,

- In order to run our applet program in the context of browser, we must write HTML program.
- In realworld applications HTML programs are developed by web programmers / web authors by making use of tags
- SUN Micro System has developed a tag for the web programmers to use
- In HTML program for running the applets and whose syntax is given below.

```
<applet code = "Name of the class"
        height = Numerical Integer value for height
        width = Numerical Integer value for width>
```

</applet>

In the above syntax,

<applet>.....</applet> is the tag.

Code, height, width are called attributes

The attribute code represents name of the class which extends java.applet.Applet class. height and width attributes represents a numerical integer value both for height and width in terms of pixels for reserving a portion of the area within the display area of the browser.

The above tag must be always written within the <Body>.....</Body> tag of HTML program.

Ex: <applet code = "MyApp"

height = 500  
width = 500>

</applet>

→ Write a HTML program for running the above applet program.

<html>

<head>  
<title> Applet program </title>

```
</head>
<Body>
    <applet code = "MyApp" height = 500 width = 500>
    </applet>
</Body>
</html>
```

Save the above program on some file name with an extension either with .html or .htm.

Open the browser Software,

- › Select the HTML program and load
- › View the result of an applet.

By using appletviewer tool :

Most of the real world browsers may not be providing their support for running the applets because of the drawbacks of applet programming.1

We know that Java is one of the independent programming language and Java language and their applications should not depend on any browser, Sun Micro System has developed a tool called appletviewer.

Using appletviewer tool, we can run our applet program without depending on any browser.

To run the applet programs by using applet viewer, one must write the `<applets>...</applets>` tag within the Java program by keeping it into commented statements for ignoring at compile time.

```
/* <applet code = "MyApp"
           height = 500
           width = 500>
    </applet> */
```

→ Syntax for appletviewer tool:

appletviewer filename.java

Ex: appletviewer MyApp.java

→ Write an applet program which illustrate the concept of life cycle methods of applet.

// LifeApp.java

```
import java.applet.Applet;
import java.awt.*;
/* <applet code = "LifeApp" height = 500 width = 500>
 * </applet> */
```

public class LifeApp extends Applet

```
{  
    String s = " " ;  
    public void init()  
    {  
        s = s + "init";  
    }
```

```
    public void start()  
    {  
        s = s + "start";  
    }
```

```
    public void stop()  
    {  
        s = s + "stop";  
    }
```

```
    public void destroy()  
    {  
        s = s + "destroy";  
    }
```

public void paint(Graphics g)

```
{  
    font f = new Font("Arial", Font.BOLD, 20);  
    g.setFont(f);  
    g.drawString(s);  
}
```

Compile the above program

javac Lifefapp.java ↴

Ensure Lifefapp.class file must be generated.

Run the applet program

> appletviewer Lifefapp.java ↴

Note: In order to see destroy method message, do the following

choose applet → Restart

(Restart = stop destroy init start)

### Windows Programming or AWT (Abstract Windowing Toolkit)

The In the olden days, in information technology programmes use to develop only ~~CUI~~ <sup>EUI based</sup> (Character User Interface). ~~CUI~~ application are nothing but command oriented applications. These applications may not be giving a clarity to the end user or normal user.

In later days we have an approach called GUI application development which gives a clarity about the application to the normal user.

In information technology we have two types of technologies for development of GUI applications. They are

a. Developer-2000 (Oracle Corporation)

VB (Microsoft)

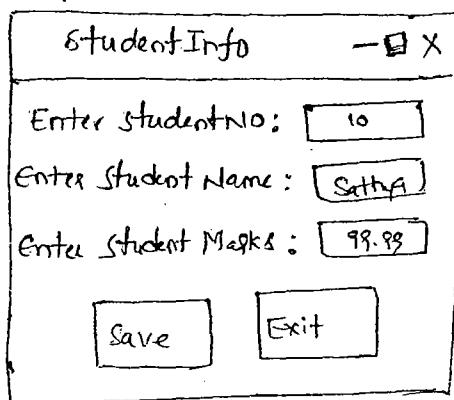
ASP (Microsoft)

If we develop any GUI applications with the above technologies then by default they run on only Microsoft providing operating systems only but not on non-microsoft providing OS. Hence these type of technologies are known as platform dependent front end technologies.

b. awt/applets/swings (java based)

- If we develop any GUI applications with the above concepts then those GUI applications will run on every OS without considering their providers
- Hence these concepts are called platform independent front end technologies
- The basic aim of AWT / Applet / Swing is to develop GUI frontend / window User interfaces applications.
- A GUI application is one which will have "look and feel"

Ex:



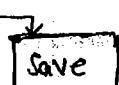
- To develop any GUI application we require GUI components / controls / widgets.
- Some of the GUI components are label, text field, button, checkbox, scrollbars etc..
- In Java programming, all the GUI components are given in the form of predefined classes which are present in the predefined package called java.awt.\*

Some of the predefined classes for the GUI components are Label, TextField, Button, Checkbox, etc.

→ Creating a GUI component is nothing but creating an object of appropriate predefined class.

Ex: Create a button with a caption Save

Button b1 = new Button("Save");



Here b1 is called reference object name, Save is called caption / name of the button / label of the button.

In order to provide functionality or life or behavior to the GUI components we require predefined classes and interfaces which are present in java.awt.event.\* package.

→ To develop the complete GUI application we import java.awt.\* and (for component creation) and java.awt.event.\* (for component functionality).

### Types of GUI Components:

In Java windows programming GUI components are classified into two types. They are

1. Auxiliary components
2. Logical components.

Log&F

-Auxiliary components are those which are always providing "touch & feel" approach (or) auxiliary components are the hardware devices which are required to interact with GUI applications.

Auxiliary components in Java does not contain any predefined classes.

Ex: Mouse, keyboard.

Logical components are those which are always providing "look & feel approach" or Logical components are the software components which are required for building GUI applications.

All the logical components in Java will have predefined classes.

Ex: Label, Button, Checkbox, Scrollbar

→ Logical components are classified into two types they are.

- a. passive / inactive / non-interactive components
- b. active / interactive components

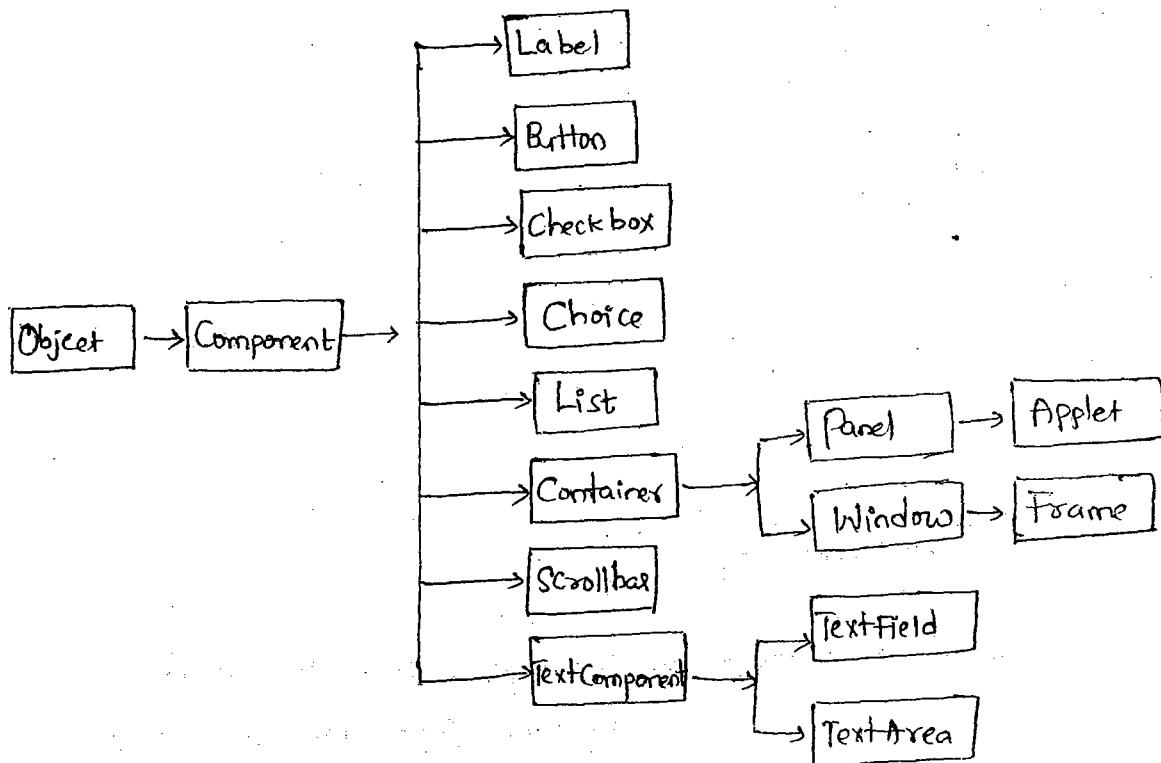
Passive components are those which does not have any interaction with the normal user / with mouse device

Ex: Label component

Active Components are those which will have an interaction with normal user with mouse device.

○ Ex: button, checkbox, scrollbar etc... (except label)

AWT Hierarchy chart :



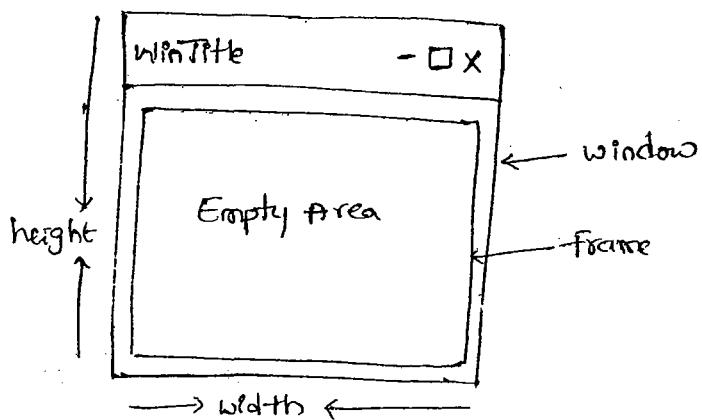
○ In the above hierarchy chart,

1. `java.lang.Object` is the Super class for all the classes in java
2. `java.awt.Component` is the Super class for all the specific components in windows programming. Component class is one of the abstract class and whose object creation is not possible. but it contains the general properties regarding specific components such as
  - changing the color of the components
  - changing the dimensions of the component etc...
3. `java.awt.Container` is one predefined class and whose object is used for adding or grouping or integrating the specific components as a single unit.

4. java.awt.Window is one of the predefined class and whose object is mandatory for every GUI application.

5. java.awt.Frame is also one of the predefined class and whose object must be embedded within the Window Component

Technically an empty area within the Window is treated as an object of frame.



If we want to develop any standalone GUI application then we must choose an appropriate class and it must extends java.awt.Frame class

Ex : Class X extends Frame

```
{  
= =  
y =
```

If we want to develop any distributed GUI application then we must choose an appropriate class and it must extends java.applet.Applet class

Ex : Class Y extends Applet

```
public class Y extends Applet  
{  
= =  
y =
```

## Steps or guidelines for developing GUI application:

1. Create a window by creating an object of Window class
2. Create a frame by creating an object of Frame class and add it to window.
3. Create a container by creating an object of Container class and add it to frame.
4. Create the specific components which are required for building the GUI application and add them to container.

→ Write a Java program which will create a window and an empty area within the window.

1) Fdemo.java

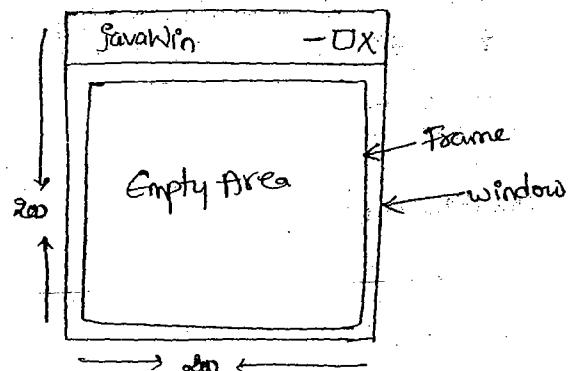
```
import java.awt.Frame;
class Myf extends Frame
{
    Myf()
    {
        setTitle ("javawin");
        setSize (200,200);
        setVisible (true);
    }
}
```

```
} //Myf
class Fdemo
{
    public static void main (String a[])
    {
        Myf mo = new Myf();
    }
}
```

```
} // Fdemo
```

In the above program

1. `public void setTitle (String):` It is one of the predefined method available in



java.awt.Window class and it is used for setting the title of the window.

1. public void setSize(int h, int w): It is also one of the pre-defined method present in java.awt.Window class and it is used for setting height and width of the window in terms of pixels.

3. public void setVisible(boolean): It is one of the pre-defined method present in java.awt.Component class and this method is used for making the created components to be visible at the time of running the application by passing true for boolean.

### Layout Managers:

The mechanism of organizing the components logically in the container is known as layout managers.

In other words layout managers are used for arranging or organizing the components logically within the container.

In Java windows programming, layout managers are given in the form of predefined classes which are present in java.awt.\* package.

In windows programming of Java contains four layout managers. They are

1. BorderLayout
2. FlowLayout
3. GridLayout
4. GridBagLayout

If we want to set any layout within the container for arranging the components then we must follow the following two steps.

1. Create an object of appropriate layout manager class

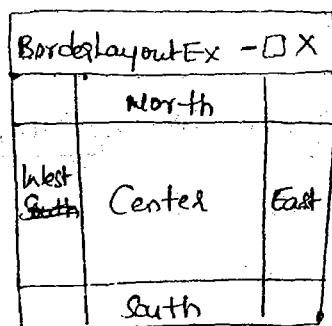
Ex: `BorderLayout bl = new BorderLayout();`

2. Set the layout by calling a predefined method called setLayout() by passing an appropriate object of layout manager classes

Ex: `setLayout(bl);`

## → 1. Border Layout:

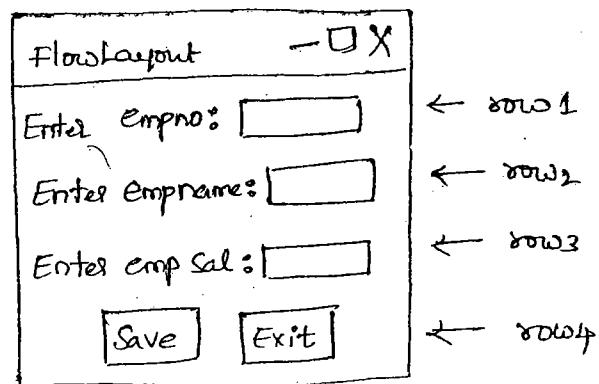
- \* It is one of the predefined layout manager.
- \* BorderLayout organizes the components logically in the container in the form of region wise.
- \* The qualified regions in the BorderLayout are
  - North,
  - South,
  - East and
  - West and
  - Center (default)
- \* The diagrammatic representation of BorderLayout is shown below.



- \* BorderLayout is the default layout manager in all standalone GUI or frame applications.
- \* Creating a BorderLayout is nothing but creating an object of BorderLayout class and calling setLayout() method by passing an object of BorderLayout class.
- Ex: `BorderLayout Bl = new BorderLayout();  
setLayout(Bl);`
- Note: setLayout() is the predefined method which is present in `java.awt.Component` class.

## 2. FlowLayout:

- \* It is one of the predefined layout manager.
- \* This layout organizes the components in the container in the form of row wise logically.
- \* The diagrammatic representation of FlowLayout is given below



\* FlowLayout is the default layout manager in distributed GUI or applet applications

\* Creating a FlowLayout is nothing but creating an object of FlowLayout class and calling setLayout() method by passing an object of FlowLayout class.

Ex: FlowLayout fl = new FlowLayout();  
setLayout (fl);

## 3. GridLayout:

- \* It is one of the predefined layout manager.
- \* GridLayout organizes the components in the form of rows and columns logically in the container.
- \* The diagrammatic representation of GridLayout is shown below

GridLayout - □ X			
1	2	3	4
5	6	7	8
9	0	+	-
*	/	%	=

\* This layout is able to maintain all the components in same size or in equal size and it is unable to maintain the space b/w vertical and horizontal components.

\* Creating a GridLayout is nothing but creating an object of GridLayout class by specifying no. of rows and columns and call setLayout(-) method by passing an object of GridLayout class.

Ex: GridLayout gl = new GridLayout(4, 4);  
             ↓          →  
             rows       cols  
     getLayout(gl);

#### 4. GridBagLayout:

\* It is one of the predefined revised layout manager from GridLayout.

\* This layout also organizes the components in the form of rows and columns logically in the container.

\* The diagrammatic representation of GridBagLayout is shown below.

GridBagLayout - □ X			
1	2	3	4
5	6	7	8
9	0	%	
+/-	*	*/=	

\* This layout is able to maintain different components either in samesize or in different size or in both the sizes. And it is able to maintain the space b/w vertical and horizontal components.

- \* Creating GridBagLayout is nothing but creating an object of GridBagLayout class by specifying no of rows and columns and call setLayout() method by passing an object of GridBagLayout class.

Ex: GridBagLayout gbl = new GridBagLayout(4, 4);  
setLayout(gbl);

### Label Component :

1. Label Component is one of the passive component.
2. Creating a Label Component is nothing but creating an object of Label class.

Ex: Label l1 = new Label("username");

Here l1 is called reference

username is called caption or label name

3. The Label Component always makes us to understand the functionality of active components.

In other words Label component always describes the functionality of active components.

4. The value of the Label object is always textual data or string data.

### Label class :

#### Data members:

```
public static final int LEFT (=0)
public static final int CENTER (=1)
public static final int RIGHT (=2)
```

The above three data members are known as label alignment modifiers.

and they are used for aligning the label in the window.

The default alignment modifier is LEFT. All the alignment modifiers values must be accessed by class name.

(2) How do you print the values of alignment modifiers?

```
import java.awt.Label;
class labmodi
{
    public void main(String args[])
    {
        System.out.println("value of left = " + Label.LEFT);
        System.out.println("value of center = " + Label.CENTER);
        System.out.println("value of right = " + Label.RIGHT);
    }
}
```

↳ class

```
> javac labmodi.java
> java labmodi
```

### Constructors

- ① Label()
- ② Label(String)
- ③ Label(String, int)

### Instance methods

- ④ public void setText(String)
- ⑤ public String getText()
- ⑥ public void setAlignment(int)
- ⑦ public int getAlignment()

\* Constructor ① is used for creating an object of Label class without specifying its caption.

Ex: Label l<sub>1</sub> = new Label();

\* Constructor ② is used for creating an object of Label class by specifying its caption and without specifying any alignment modifier.

Ex: Label l<sub>1</sub> = new Label("username");

When we use constructor ① and ② for creation of labels, which are by default aligned at left side.

\* Constructor ③ is used for creating an object of Label class by specifying option and its alignment modifier

Ex: Label l<sub>1</sub> = new Label ("username", Label.RIGHT);

\* Methods ④ and ⑤ are used for setting the caption to the Label component and getting the caption from the Label component.

Ex: Label l<sub>2</sub> = new Label();  
l<sub>2</sub>.setText ("password");  
String lab = l<sub>2</sub>.getText();  
Sop (lab); // password.

\* Methods ⑥ and ⑦ are used for setting the label alignment modifier to the Label component and getting the Label alignment modifier from the Label component.

Ex: Label l<sub>2</sub> = new Label ("password");  
int x = l<sub>2</sub>.getAlignment();  
Sop (x); // 0 - LEFT  
l<sub>2</sub>.setAlignment (Label.CENTER);  
int x = l<sub>2</sub>.getAlignment();  
Sop (x); // 1 - CENTER

(1) Define mutators and inspectors?

a) Mutators are the methods which are used for changing the value of the object. It

The general notation of mutator method is

public void setXXX (parameter)

Inspectors are the methods which are used for obtaining the value of the object. and whose general notation is

public ReturnType getXXX()

Note: In Java programming if any class is containing set of `setXxx()` methods and set of `getXxx()` methods then that class is known as Java Bean class (which is a reusable component or class).

### EVENT DELEGATION MODEL

The basic aim of event delegation model is to provide life / functionality behavior to the active components.

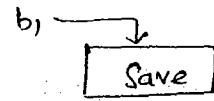
In general all the active components of windows programming participates in event delegation model. And no passive components participates in event delegation model because active components are interactable and passive components are non interactable.

Event delegation model contains four phases. They are

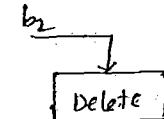
#### I phase:

a. Each and every GUI component can be qualified with caption and reference

Button b<sub>1</sub> = new Button("Save");



Button b<sub>2</sub> = new Button("Delete");



Here b<sub>1</sub>, b<sub>2</sub> are called references and

Save, delete are called captions.

b. A GUI Component can be processed either by using reference & by using caption.

\* c. Whenever we interact with any active component, automatically JVM creates an object with respect to a predefined class and whose generalized notation is

`XXXEvent` class and it records the reference and

caption. In general an object of `XXXEvent` records & stores the caption and reference of the active component.

XXXEvent obj

reference	Caption
-----------	---------

- Q. The following table gives GUI component name and its corresponding XXXEvent class name.

GUI Component name	XXXEvent
Label	—
Button	ActionEvent
checkbox	ItemEvent
<u>TextField</u>	TextEvent
TextArea	
Scrollbar	AdjustmentEvent
Window	WindowEvent

All above pre-defined event classes are present in a package called java.awt.event.\* package.

[phase :

In order to provide a functionality to each and every GUI Component we must write a piece of code in the form of block of statements. We know that block of stmts must be written always in methods.

SUN Micro System has given some pre designed methods for writing the block of statements. In Java programming pre-designed methods are known as underlined or abstract methods. To get more advantages, SUN Micro System placed the abstract methods in interface.

In Windows programming interfaces are known as listeners.

Hence each and every GUI Component must take the abstract method from the specific listener and whose generalized notation is

### xxxListener

The following table gives GUI Component name and their corresponding xxxListener.

GUI Component Name	xxxListener
Label	—
Button	ActionListener
Checkbox	ItemListener
TextField	TextListener
TextArea	—
Scrollbar	AdjustmentListener
Window	WindowListener

All the above predefined Listener belongs to a package called java.awt.event.\*.

### III phase:

Identify the abstract method present in xxxListener.

The following table gives xxxListener and corresponding abstract method.

<u>xxxListener</u>	<u>Abstract Method</u>
ActionListener	public abstract void actionPerformed(ActionEvent)
ItemListener	public abstract void itemStateChanged(ChangeEvent)
TextListener	public abstract void textValueChanged(TextEvent)
AdjustmentListener	public abstract void adjustmentValueChanged(AdjustmentEvent)

### IV phase:

To provide the functionality to each and every GUI component, as a java programmer we must register with appropriate listeners. Once the component functionality is over, automatically JVM will unregister from its listener.

In general each and every GUI Component will have the following generalized event registration and unregistration methods.

public void addxxxListener (xxxListener) // Registration

public void removexxxListener (xxxListener) // Un registration.

The following table gives GUI Component name and corresponding registration and unregistration methods.

<u>GUI Component Name</u>	<u>Registration Method</u>	<u>Unregistration Method</u>
Label	—	—
Button	public void addActionListener (ActionListener)	public void removeActionListener (ActionListener)
checkbox	public void addItemListener (ItemListener)	public void removeItemListener (ItemListener)
extfield	public void addTextListener (TextListener)	public void removeTextListener (TextListener)
extarea	—	—
Scrollbar	public void addAdjustmentListener (AdjustmentListener)	public void removeAdjustmentListener (AdjustmentListener)

The summary of event delegation model is given in the following table for all the GUI Components.

## GUI component

### Abstract method

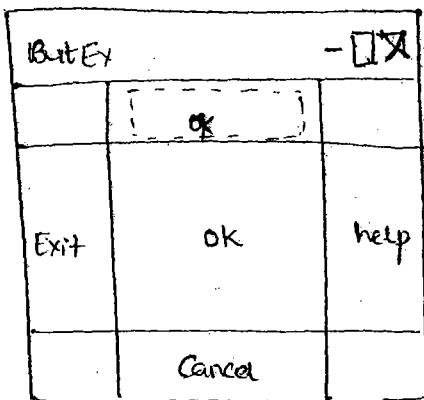
### Registration

### on Registration.

## label

button	ActionEvent	ActionListener	public void abstract void actionPerformed (ActionEvent) { Action Listener } " "	public void addactionListener (Action Listener) " " public void removeactionListener (Action Listener) " "
list multicheck)	"	"	"	"
checkbox	ItemEvent	ItemListener	public abstract void itemStateChanged (ItemEvent) { Item Listener } " "	public void additemStateChanged (Item Listener) " " public void removeitemStateChanged (Item Listener) " "
choice	"	"	"	"
list (single check)	"	"	"	"
textfield <u>textArea</u>	TextEvent	TextListener	public abstract void textValueChanged (TextEvent) { Text Listener } " "	public void addTextListener (Text Listener) " " public void removeTextListener (Text Listener) " "
scrollbar	AdjustmentEvent	AdjustmentListener	public abstract adjustmentValueChanged (AdjustmentEvent) { Adjustment Listener } " "	public void addadjustmentListener (Adjustment Listener) " " public void removeadjustmentListener (Adjustment Listener) " "

→ Write a java program for developing the following screen.



The above screen can be developed in two ways.

- a. By using frames
- b. By using applets

By using frames :

```
1. ButDemo.java
import java.awt.*;
import java.awt.event.*;
class But extends Frame implements ActionListener
{
    Button b1,b2,b3,b4;
    Label l1;
    but()
    {
        setTitle("ButEx");
        setSize(200,200);
        // BorderLayout bl=new BorderLayout();
        // setLayout(bl); x)
    }
    //Create the components
    b1=new Button("OK");
    b2=new Button("CANCEL");
    b3=new Button("HELP");
    b4=new Button("EXIT");
}
```

①      ②      ③      ④      ⑤      ⑥

l<sub>1</sub> = new Label(); }  
 l<sub>1</sub>.setAlignment(Label.CENTER); }

⑩ add the components to container.

add(b<sub>1</sub>, "North"); }  
 add(b<sub>2</sub>, "South"); }  
 add(b<sub>3</sub>, "East"); }  
 add(b<sub>4</sub>, "West"); }  
 add(l<sub>1</sub>); }

⑪ registration

⑫ { b<sub>1</sub>.addActionListener(this); /\* 'this' is an object of current  
 class which implements  
 ActionListener i.e. object of curr  
 ent class is nothing but obj  
 b<sub>2</sub>.addActionListener(this); \*/  
 b<sub>3</sub>.addActionListener(this);  
 b<sub>4</sub>.addActionListener(this);  
 // make the components to be visible of ActionListener \*/  
} // but<sub>4</sub> setVisible(true); } ⑬

public void actionPerformed(ActionEvent ae) ⑭ → ⑮

if (ae.getSource() == b<sub>1</sub>)  
{ String cap = b<sub>1</sub>.getLabel();  
l<sub>1</sub>.setText(cap);  
}

if (ae.getSource() == b<sub>2</sub>)  
{  
String cap = b<sub>2</sub>.getLabel();  
l<sub>1</sub>.setText(cap);  
}

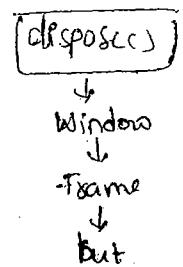
if (ae.getSource() == b<sub>3</sub>)  
{  
String cap = b<sub>3</sub>.getLabel();  
l<sub>1</sub>.setText(cap);  
}

```

if (ae.getSource() == b4)
{
    String cap = budgetLabel();
    l1.setText(cap); // or
    // System.exit(); or
    dispose();
}
}

class ButDemo
{
    public static void main (String[] args)
    {
        But bb = new But();
    }
}

```



### Steps or Guidelines for developing standalone or frame applications:

In order to develop any standalone GUI application, the Java programmers must follow the following steps.

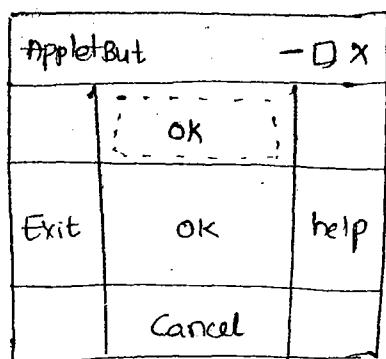
1. Import `java.awt.*` (for component creation), `java.awt.event.*` (for component functionality) and other packages if required.
2. Choose the user-defined class
3. Whatever the class is selected in Step ②, it must extends `Frame` and implements appropriate listeners if required
4. Identify the type of GUI components required for building GUI application
5. Set the title, set the size of the window and layout (if required)
6. Create the GUI components which are identified in step ④

- ⑦ Whatever the comp are created in step ⑥, add them to container.
- ⑧ Registers the events of GUI active components with appropriate listeners.
- ⑨ Make the components to be visible.
- ⑩ Define the appropriate abstract method which is inherited from appropriate Listener.

### Steps/Guidelines for developing distributed or applet applications:

- If we want to develop any distributed GUI or applet applications, as a Java programmer we need to follow the following standard steps.
- 1. Import `java.awt.*` (for component creation), `java.awt.event.*` (for component functionality), `java.applet.Applet` (for lifeCycle methods) and other packages if required.
- 2. Choose an appropriate user defined class and ensure whose modifier must be public.
- 3. Whatever the class is selected in step ②, it must extends `java.applet.Applet` class and implements appropriate listeners if required.
- 4. Identify the type of GUI components which are required for building GUI applications.
- 5. Override `public void init()` method to do the following
  - 5.1 Set the layout if required
  - 5.2 Create the GUI components which are identified in step ④
  - 5.3 Add the created components to the container  
(These are the one time activities so put in `init()`)
- 6. Override `public void start()` method to do the following
  - 6.1 Register the events of GUI active component with appropriate listeners (through the generated connection)

7. Make the created components to be visible (this is optional in the case of applet, because applet related components are by default visible)
8. Define / override the appropriate abstract method which is inherited from appropriate Listener.
- Write an applet program for developing the following screen.



### 1) ButApp.java

```

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
}
<applet code = "ButApp" height=200 width=200>
</applet>
}
public class ButApp extends Applet implements ActionListener
{
    Button b1, b2, b3, b4, b5;
    Label l1;
}
public void init() {
    BorderLayout bl = new BorderLayout();
    setLayout(bl);
}
}
  
```

①      ②      ③      ④      ⑤      ⑥.1

// create components

```
b1 = new Button("OK");
b2 = new Button("CANCEL");
b3 = new Button("HELP");
b4 = new Button("EXIT");
l1 = new Label();
l1.setAlignment(Label.CENTER);
```

(5.2)

// add the comp to container

```
add(b1, "North");
add(b2, "South");
add(b3, "East");
add(b4, "West");
add(l1);
```

(5.3)

} // init()

public void start() // -- ⑥

{

// registration

```
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
```

(6.1)

// setVisible(true); // -- ⑦

} // start

public void actionPerformed(ActionEvent ae) // -- ⑧

{

if (ae.getSource() == b<sub>1</sub>)

```
{ String cap = b1.getLabel();
l1.setText(cap); }
```

```

if (ae.getSource() == b2)
{
    String cap = b2.getLabel();
    l1.setText(cap);
}

if (ae.getSource() == b3)
{
    String cap = b3.getLabel();
    l1.setText(cap);
}

if (ae.getSource() == b4)
{
    String cap = b4.getLabel();
    l1.setText(cap);
}

} // actionPerformed
}

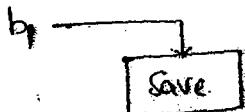
} // class

```

### Button Component:

- Button is one of the active component . So that it participates in Event delegation model.
- Creating a button is nothing but creating an object of Button class.

Button b1 = new Button("Save");



Here b1 is called reference

Save is called caption.

- \* The basic purpose of Button Component is that to perform "atomic unit of operations" such as save, delete, update, exiting etc operations.

## Button class :

### Constructor :

- ① Button ()
- ② Button (String)

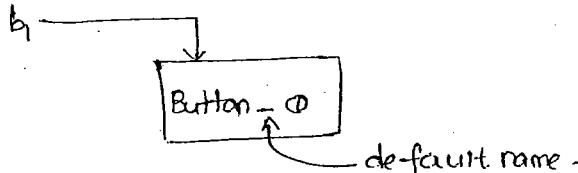
### Instance methods

- ③ public void setLabel (String)
- ④ public String getLabel () ;
- ⑤ public void addActionListener (ActionListener)
- ⑥ public void removeActionListener (ActionListener)

\* Constructor ① is used for creating an object of Button class without specifying any caption and the default captions start with Button\_Ø, Button\_1, ... Button\_(n+2);

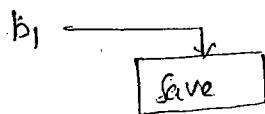
### Ex :

Button b<sub>1</sub> = new Button ();



\* Constructor ② is used for creating an object of Button class by specifying its caption.

Ex : Button b<sub>1</sub> = new Button ("Save");



\* Methods ③ and ④ are used for setting the label to the Button object and getting the label from the Button.

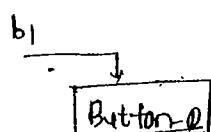
Ex : Button b<sub>1</sub> = new Button ();

String s<sub>1</sub> = b<sub>1</sub>.getLabel () ;

Sop (s<sub>1</sub>) ; // Button\_Ø

b<sub>1</sub>.setLabel ("Save");

String s<sub>2</sub> = b<sub>1</sub>.getLabel () ;



`sep(s); //Save`

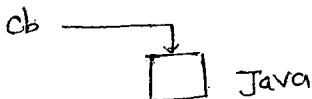
\* Methods ⑤ and ⑥ are used for registering/unregistering the events of Button with ActionListener interface.

### Checkbox Component :

Checkbox is one of the active component and hence it participates in event delegation model.

Creating a checkbox is nothing but creating an object of checkbox class.

Ex: `checkbox cb = new Checkbox("Java");`



Here, we know that cb is called reference and Java is called label/caption.

Graphically a checkbox may be defined as square symbol (□) plus label.

The Component Checkbox contains two types of states. They are

1. Unchecked state (false)
2. Checked state (true)

The default state of the checkbox is unchecked.

The hidden feature →

The real time implementation of checkbox component is that to achieve the functionality of checkbox.

### Checkbox class :

Constructors

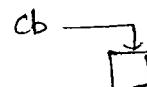
- ① Checkbox()
- ② Checkbox(String)
- ③ Checkbox(String, boolean)

Instance methods

- ④ public void setLabel(String)
- ⑤ public String getLabel()
- ⑥ public void setState(boolean)
- ⑦ public boolean getState()
- ⑧ public void addItemListener(ItemListener)
- ⑨ public void removeItemListener(ItemListener)

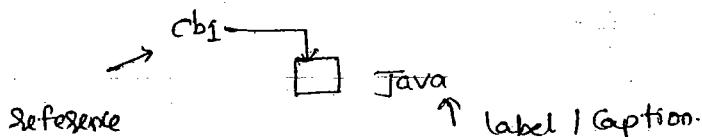
\* Constructor ① is used for creating an object of Checkbox without specifying caption and state.

Ex: Checkbox cb = new Checkbox();



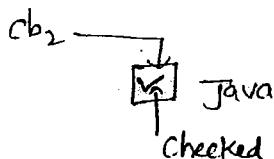
\* Constructor ② is used for creating an object of Checkbox class by specifying the caption and without specifying its state.

Ex: Checkbox cb1 = new Checkbox("Java");



\* Constructor ③ is used for creating an object of Checkbox class by specifying the caption and state.

Ex: Checkbox cb2 = new Checkbox("Java", true);



\* Methods ④ and ⑤ are used for setting the label to the checkbox and getting the label from checkbox.

Ex:

```

checkbox cb1 = new checkbox();
cb1.setLabel("Java");
String sp = cb1.getLabel();
System.out.println(sp); // Java.

```

\* Methods ⑥ and ⑦ are used for setting and getting the state of the checkbox.

Ex: Checkbox cb<sub>2</sub> = new checkbox ("Java");

cb<sub>2</sub> →  
 Java

```
boolean state = cb2.getState();
```

```
System.out.println(state); // false
```

```
cb2.setState(true);
```

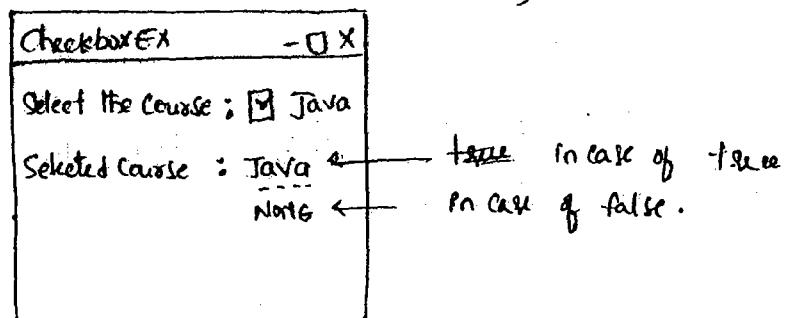
cb<sub>2</sub> →  
 Java.

```
state = cb2.getState();
```

```
System.out.println(state); // true
```

\* Methods ⑧ and ⑨ are used for registering/unregistering the events of checkbox with ItemListeners. That is as and when the state of the checkbox is changing, automatically itemStateChanged() method of ItemListener will be called. (This is called registration)

→ Write a Java program to implement the following screen



The above screen can be implemented by

- a. using frames
- b. using applets

a. By using frames :

//CbDemo.java

```
import java.awt.*;
import java.awt.event.*;
```

Class cbdemo extends Frame

{

Label l1, l2;

checkbox cb1;

abc)

{

setTitle ("checkboxEx");

setSize (250,250);

FlowLayout fl = new FlowLayout();

setLayout (fl);

//Create the components

l1 = new Label ("Select the course");

l2 = new Label ();

cb1 = new Checkbox ("Java");

//Add the Components

add(l1);

add(cb1);

add(l2);

//Registration

Hyd ho = new Hyd(); // here Hyd is inner class which is implementing  
'ItemListener'

```

cb1.addItemListener(ho); // here 'ho' is object of inner class
setVisible(true);

} // cbc

class Hyd implements ItemListener { inner class def ...
{
    public void itemStateChanged(ItemEvent ie)
    {
        if (ie.getSource() == cb1)
        {
            if (cb1.getState()) // inner the case of true
            {
                l2.setText("Selected Course : " + cb1.getLabel());
            }
            else // in the case of false
            {
                l2.setText("Selected Course : None");
            }
        }
    } // isc
} // Hyd - inner class

} // cb - outer class

class cbDemo
{
    public static void main(String[] args)
    {
        cb co = new cb();
    } // main
} // cbDemo

```

— Rewrite the above program by using applets.

○ // cbdemoapplet.java

import java.awt.\*;

import java.awt.event.\*;

import java.applet.Applet;

public class cbdemoapplet extends Applet

{ Label l1, l2;

Checkbox cb1;

Public void init()

{ Create the comp

l1 = new Label("Select the course :");

l2 = new Label();

cb1 = new Checkbox("Java");

Add the comp

add(l1);

add(cb1);

add(l2);

} init()

public void start()

{ Registration

Hyd h0 = new Hyd();

cb1.addItemListener(h0);

setVisible(true);

}

class Hyd implements ItemListener

{ public void itemStateChanged(ItemEvent ie)

{ if (ie.getSource() == cb1)

{ if (cb1.getState())

```
{ l2.setText("Selected Course : " + cb1.getSelectedItem());  
}  
else  
{  
    l2.setText("Selected Course : None");  
}  
}  
if
```

} || isCC

y || Hyd

g || Ch outer class

```
<applet code="cbDemoApplet" height=400 width=400 >  
</applet>
```

```
>javac cbDemoApplet.java  
>appletviewer cbDemoApplet.java
```

Q) Define inner class and containesship?

A) If the definition of one class (say Y) is written within the definition of another class (say X) then the one class (Y) is known as inner/nested class.

The process of defining one class definition within the definition of another class is known as containesship or composition.

```

class X
{
    class Y
    {
        }
    }
}
} // Y-inne
} // X-outer

```

In the above example the class X is called outer class and the class Y is called inner class.

The process of writing def within the X class def is known as containment / composition.

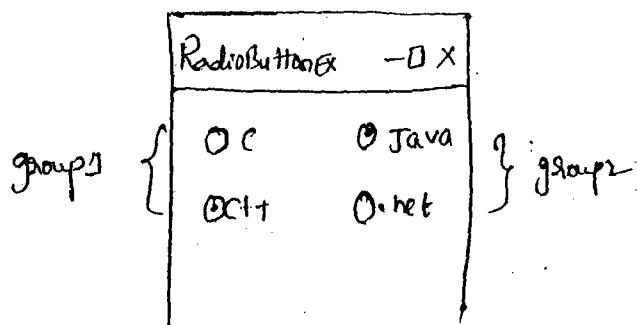
### RadioButton:

In windows programming of Java (awt) we are not having a direct class called RadioButton. But we have a procedure to ~~separate~~ achieve RadioButton.

In windows programming of Java the component checkboxes allows us to select multiple whereas the component RadioButton allows us to select one from the group

Obtaining a RadioButton is nothing but converting checkboxes into RadioButton components.

Ex: Consider the following screen



Steps or procedure for converting check boxes into RadioButtons:

Step 1: Decide the no.of checkboxes and identify their groups.

Ex: Checkbox cb<sub>1</sub> = new Checkbox("C");  
Checkbox cb<sub>2</sub> = new Checkbox("C++");  
Checkbox cb<sub>3</sub> = new Checkbox("Java");  
Checkbox cb<sub>4</sub> = new Checkbox(".net");

Here, we identified cb<sub>1</sub> and cb<sub>2</sub> is first group, cb<sub>3</sub> and cb<sub>4</sub> is second group.

Step 2: Based on the no.of groups we identified, we create that many no.of objects of CheckboxGroup class. CheckboxGroup is the predefined class which allows us to select only one checkbox from multiple checkboxes in that particular group. and appearance of the component changes from  →

Ex: CheckboxGroup cbg<sub>1</sub> = new CheckboxGroup();  
CheckboxGroup cbg<sub>2</sub> = new CheckboxGroup();

Step 3: Add the identified check box objects (cb<sub>1</sub>, cb<sub>2</sub> → cbg<sub>1</sub>) & (cb<sub>3</sub>, cb<sub>4</sub> → cbg<sub>2</sub>) to the appropriate CheckboxGroup class object.

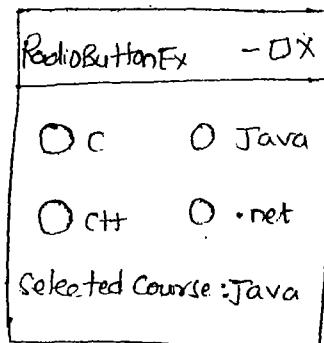
To add the checkbox object to CheckboxGroup class object we use the following method which is present in Checkbox class.

java.awt.Checkbox  
↓  
public void setcheckboxGroup(CheckboxGroup)

Ex: cb<sub>1</sub>.setcheckboxGroup(cbg<sub>1</sub>);  
cb<sub>2</sub>.setcheckboxGroup(cbg<sub>1</sub>);  
cb<sub>3</sub>.setcheckboxGroup(cbg<sub>2</sub>);  
cb<sub>4</sub>.setcheckboxGroup(cbg<sub>2</sub>);

→ Note: Event delegation model of checkbox to be applied as it is even for RadioButtons also.

→ Write a Java program to implement the following screen.



// RdDemo.java

```

import java.awt.*;
import java.awt.event.*;

class RB extends Frame
{
    Label l1;
    Checkbox cb1, cb2, cb3, cb4;
    CheckboxGroup cbg1, cbg2;

    RB()
    {
        setTitle ("RadioButtonEx");
        setSize (300, 300);
        flowLayout fl = new FlowLayout();
        setLayout (fl);

        // Create the Comp
        l1 = new Label ();
        l1.setText ("Selected Course : ");

        cb1 = new Checkbox ("C");
        cb2 = new Checkbox ("C++");
        cb3 = new Checkbox ("JAVA");
        cb4 = new Checkbox (".NET");
      
```

// Step-2

```
cbg1 = new CheckboxGroup();
```

```
cbg2 = new CheckboxGroup();
```

// Step-3

```
cb1.setCheckboxGroup(cbg1);
```

```
cb2.setCheckboxGroup(cbg1);
```

```
cb3.setCheckboxGroup(cbg2);
```

```
cb4.setCheckboxGroup(cbg2);
```

// add the comp to container

```
add(cb1);
```

```
add(cb2);
```

```
add(cb3);
```

```
add(cb4);
```

```
add(l1);
```

// Registration

```
Hyd ho = new Hyd()
```

```
cb1.addItemListener(ho);
```

```
cb2.addItemListener(ho);
```

```
cb3.addItemListener(ho);
```

```
cb4.addItemListener(ho);
```

```
setVisible(true);
```

} // abc

class Hyd implements ItemListener

```
{
```

```
public void itemStateChanged(ItemEvent ie)
```

```
{
```

```
    Checkbox cb = (Checkbox) ie.getItemSelectable(); // obj type casting
```

```
* Object obj = ie.getItemSelectable();
```

```
    Checkbox cb = (Checkbox) ie.getItemSelectable(); // obj
```

```

        if (cb.getState())
    {
        l1.setText("Selected course: " + cb.getLabel());
    }
}

} // MISCs

} // Hyd_inner-class

} // cb_outer-class

class RdDemo
{
    public static void main(String[] args)
    {
        xb xo = new xb();
    } // main
} // RdDemo

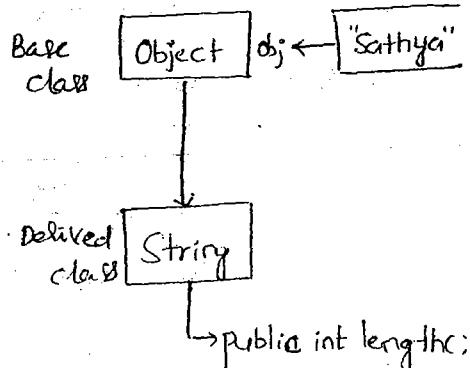
```

(Q) Define object type casting and explain its necessity?

Def: The process of getting Base class object

into derived class object is known as  
Object type casting

Necessity:



- ① The result of the Java program is available within the object of `java.lang.Object`. (Base class)
- ② To process the result of object of `java.lang.Object`, we are not having suitable methods. (in base class)
- ③ To process the result of object of `java.lang.Object` (base class result) we have the suitable methods present in its derived class
- ④ We know that by using an object of `java.lang.Object` we can't access

the special methods of derived class.

⑤ Finally to process the result of java object of `java.lang.Object` with respect to the derived class special methods we must use the concept of object type casting.

Syntax:

`Subclassname subobj = (Subclassname) SupObj;`

Ex: `int noc = obj.length();` // invalid

`String s = (String) obj;` // object type casting

`int noc = s.length();`

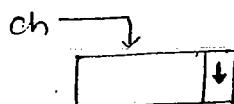
`sop(noc);` // 6

Choice:

\* Choice is one of the active component so that it participates in event delegation model.

: Creating a choice is nothing but creating an object of choice class

Ex: `Choice ch = new Choice();`



The component choice contains only reference but not caption

\* The component choice allows us to add the data in the form of string.

Constructors

`Choice()`

Instance methods

② `public void add(String)` → (1.4 version) { same }

③ `public void addItem(String)` - (1.5 version) { same }

④ `public void add(String, int)` { same }

⑤ `public void addItem(String, int)` { same }

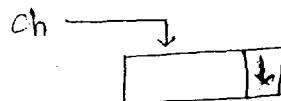
⑥ `public String remove(int)`

- ⑦ public void remove(String)
- ⑧ public void removeAll()
- \* ⑨ public String getSelectedItem()
- ⑩ public int getSelectedIndex()
- ⑪ public void addItemListener(ItemListener)
- ⑫ public void removeItemListener(ItemListener)

→ Internally all the elements of choice are organized by following Zero based Indexing concept (arrays)

→ Constructors ① used for creating an object of choice component

Ex: Choice ch = new Choice();

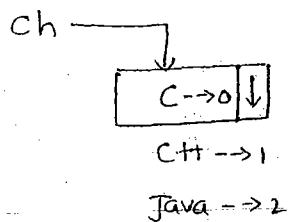


→ Methods ② and ③ are used for adding the elements to the choice component only at the end.

Ex: ch.add("c");

ch.addItem("C++");

ch.add("Java");

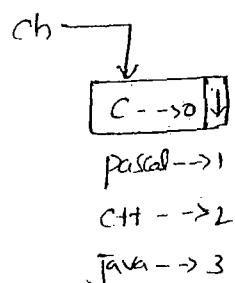


→ Methods ④ and ⑤ are used for adding the elements to the choice component at a specific position.

Ex: ch.add("pascal", 1);

(or)

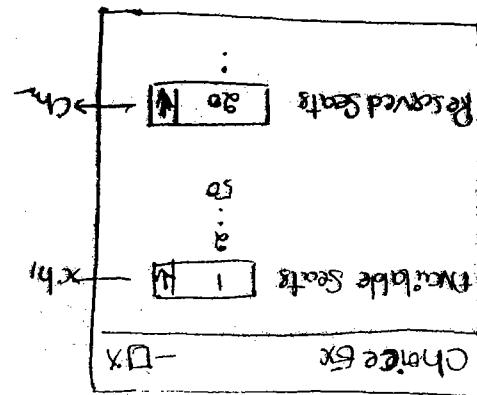
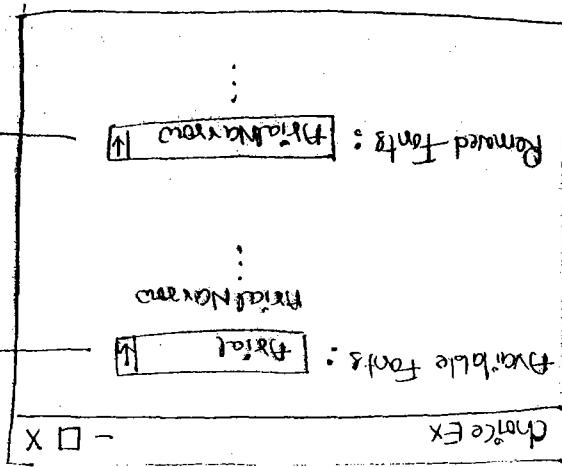
ch.addItem("pascal", 1);



→ Methods ⑥ and ⑦ are used for removing the elements of the choice either based on position or based on content.

Ex: str. String s = ch.remove(1); // Based on position

(or)  
ch.remove("pascal"); // Based on content



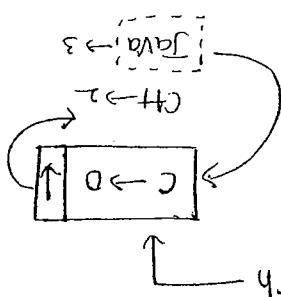
+ Write a Java program to implement the following screen.

Methods (1) and (2) are used for Segregating | un segregating the items of Choice component as it is selected, automatically `ItemState Changed(→)` of ItemListerner will be called.

Ex: `if pos = ch.getSelectedIndex();`

`if pos = ch.getSelectedIndex();`

Method (1) used for obtaining an index of item which is selected at run time from the Choice component.



Ex: `if pos = ch.getSelectedIndex();`

`String s = ch.getSelectedIndex();`

From the Choice component at run time.

Method (2) is used for selecting an item or activating an item

Ex: `ch.setSelectedIndex();`

Method (3) is used for removing all the elements of Choice component.

(ie first occurrence).

The cursor must be click sensitive and removes only one occurrence.

= By using applet:

```

import java.awt.*;  

import java.awt.event.*;  

import java.applet.Applet;  

/* <applet code="Chapp" height=250 width=250>  

</applet> */  
  

public class Chapp extends Applet  

{  

    Label l1, l2;  

    Choice ch1, ch2;  

    public void init()  

    {  

        // Setting the layout is optional bcoz applet by default FlowLayout.  

        l1=new Label("Available Fonts");  

        l2=new Label("Removed Fonts");  

        ch1=new Choice();  

        ch2=new Choice();  

        // Code for getting the Fonts  

        GraphicsEnvironment ge=GraphicsEnvironment.getLocalGraphicsEnvironment();  

        String f[]={};  

        f=ge.getAvailableFontFamilyNames();  

        // add the fonts to ch1  

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

        {  

            ch1.add(f[i]);  

        }  

        // add the comp to container  

        add(l1);  

        add(ch1);  

        add(l2);  

        add(ch2);  

    }
}

```

adding no to string

for (int i=1; i<=lw; i++)

String s=string.valueOf(i);

ch1.add(s);

```

public void start()
{
    Sathyap s0 = new Sathyap(); // here Sathyap is inner class which is
                                // implementing "ItemListener"
    ch1.addItemListener(s0); // registration
}

// start()

class Sathyap implements ItemListener // inner cls def
{
    public void itemStateChanged(ItemEvent ie)
    {
        if (ie.getSource() == ch1)
        {
            String s = ch1.getSelectedItem();
            ch2.add(s);
            ch1.remove(s);
        }
    }
}

```

} // Sathyap - inner

} // Chapp - outer

~~Sathyap~~

-> ch1

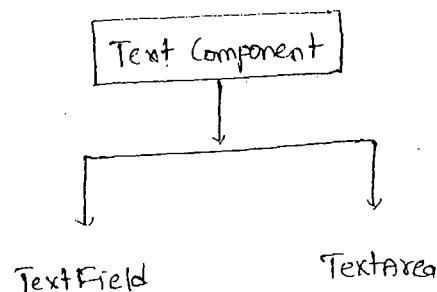
### Text Component:

In Windows programming other than the Text component like Button, Choice, Checkbox etc.,.. allows us to Select & interact to perform some actions.

Text component is the one which allows us to take the User input either in the form of single line of text or in the form of multiple lines of text.

Text Component is one of the predefined abstract class and whose object can't be created directly.

- Text component class provides general properties <sup>to</sup> regarding their subclasses TextField and Textarea.



### Methods in Text Component:

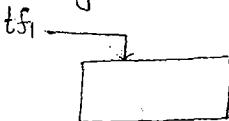
- ① public void setText (String)
- ② public String getText()

The above methods are used for setting the text and getting the text of TextField and both the methods are inherited into TextField and Textarea classes.

### TextField Component:

- TextField is one of the active component and hence it participates in Event delegation model.
- Creating a TextField is nothing but creating an object of TextField Class

Ex: TextField tf = new TextField();



Even for the TextField it contains only reference but no caption.

- Whatever the data we enter into Text Field which is by default belongs to String data.
- The functionality of TextField Component is that to accept one line of Text as an input from the application user.
- TextField Component is divided into two types
  - a. Non-Echo TextField
  - b. Echo TextField

\* A non-echo TextField is one whose data is able to be understandable by every application user.

Ex: UserName:   $\leftarrow$  tf<sub>1</sub>

Here tf<sub>1</sub> is called non-echo textfield.

\* An echo TextField is one whose content is not able to be understandable by other application users.

Ex: password:   $\leftarrow$  tf<sub>2</sub>

Here tf<sub>2</sub> is known as echo textfield and '\*' is known as echo character.

→ Whenever we create a TextField which is by default non-echo. One can convert programmatically from non-echo text field to echo text field.

### TextField class :

#### Constructors :

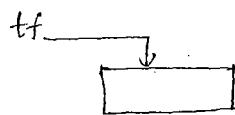
- ① TextField()
- ② TextField (int size)
- ③ TextField (String)

#### Instance methods

- ④ public void setSize (int)
- ⑤ public int getSize ()
- ⑥ public void setText (String)
- ⑦ public String getText ()
- ⑧ public void setEchochar (char)
- ⑨ public char getEchochar ()
- ⑩ public boolean setEchochar (Set)
- ⑪ public void addTextListener (TextListener)
- ⑫ public void removeTextListener (TextListener)

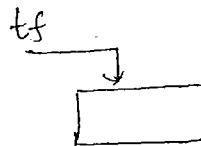
\* Constructor ① is used for creating an obj of TextField class with the default size

Ex: `TextField tf = new TextField();`



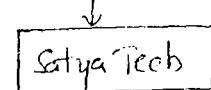
\* Constructor ② is used for creating an object of TextField class by specifying the length of the TextField

Ex: `TextField tf = new TextField(40);`



\* Constructor ③ is used for creating an obj of TextField by placing the predecided text. Some of the predecided text data is like Company names, System date, addresses etc.

Ex: `TextField tf1 = new TextField("Satya Tech");`



\* Methods ④ and ⑤ are used for setting and getting the size of the TextField.

Ex: `TextField tf1 = new TextField();`

`int size = tf1.getSize();`

`System.out.println(size); // 10 (by default)`

`tf1.setSize(40);`

`size = tf1.getSize();`

`System.out.println(size); // 40 (by program)`

\* Methods ⑥ and ⑦ are used for setting and getting the echo characters of the TextField component.

In other words method ⑥ is used for converting non-echo textField into echo textField.

\* Method ⑩ returns true provided text field belongs to echo otherwise it returns false (in the case of non-echo).

```

Ex: TextField tf1 = new TextField();
TextField tf2 = new TextField();
tf1.setEchoChar('$');
char ch = tf1.getEchoChar();
System.out.println(ch); // $ (Echo character)

boolean b = tf1.isEchoCharSet();
System.out.println(b); // true

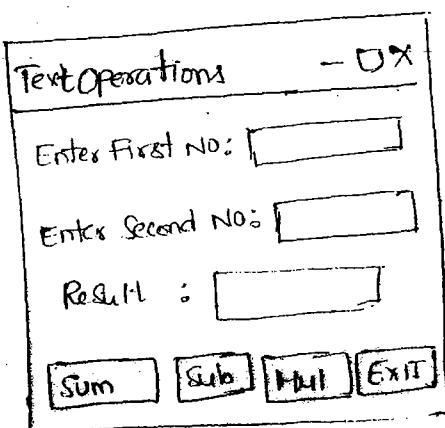
b = tf2.isEchoCharSet();
System.out.println(b); // false

```

\* Methods ⑪ and ⑫ are used for registering / unregistering the events of TextField with Text Listener.

In otherwords as and when the value of the TextField is changing, automatically `textValueChanged()` method of Text Listener must be called.

→ Write a Java program to implement the following



11

```

import java.awt.*;
import java.awt.event.*;

```

public class to extends Frame

Label l<sub>1</sub>, l<sub>2</sub>, l<sub>3</sub>;

TextField tf<sub>1</sub>, tf<sub>2</sub>, tf<sub>3</sub>;

Button b<sub>1</sub>, b<sub>2</sub>, b<sub>3</sub>, b<sub>4</sub>;

to

{

setTitle ("TextOperations");

setSize (300, 300);

FlowLayout fl = new FlowLayout();

setLayout (fl);

// Create the comp

l<sub>1</sub> = new Label ("Enter First No.");

l<sub>2</sub> = new Label ("Enter Second No.");

l<sub>3</sub> = new Label ("Result");

tf<sub>1</sub> = new TextField ();

tf<sub>2</sub> = new TextField ();

tf<sub>3</sub> = new TextField ();

b<sub>1</sub> = new Button ("Sum");

b<sub>2</sub> = new Button ("Sub");

b<sub>3</sub> = new Button ("Mul");

b<sub>4</sub> = new Button ("EXIT");

Add the comp to container.

add (l<sub>1</sub>);

add (tf<sub>1</sub>);

add (l<sub>2</sub>);

add (tf<sub>2</sub>);

add (l<sub>3</sub>); add (tf<sub>3</sub>);

```
add(b1); add(b2); add(b3); add(b4);
```

```
// Registration Hycd ho=new Hycd();
```

```
    ho  
b1.addActionListener(ho);
```

```
    ho  
b2.addActionListener(ho);
```

```
    ho  
b3.addActionListener(ho);
```

```
    ho  
b4.addActionListener(ho);
```

```
// tf1.addTextListener(ho);  
// tf2.addTextListener(ho);
```

```
setVisible(true);
```

```
} Hycd()
```

Class Hycd implements ActionListener, TextListener -

```
{
```

```
    public void actionPerformed(ActionEvent ae)
```

```
    { if (ae.getSource() == b1)
```

```
        String s1 = tf1.getText();
```

```
        String s2 = tf2.getText();
```

```
        int x = Integer.parseInt(String.valueOf(tf1)); Integer.parseInt(s1);
```

```
        int y = Integer.parseInt(String.valueOf(tf2)); Integer.parseInt(s2);
```

```
        int z = x + y; String n = String.valueOf(z);
```

```
        tf3.setText(n);
```

```
}
```

```
    if (ae.getSource() == b2)
```

```
    { String s1 = tf1.getText();
```

```
        String s2 = tf2.getText();
```

```
        int x = Integer.parseInt(String.valueOf(tf1)); Integer.parseInt(s1);
```

```
        int y = Integer.parseInt(String.valueOf(tf2)); Integer.parseInt(s2);
```

```

    int z = x-y; String n = String.valueOf(z);
    tf3.setText(n);

}

if (ae.getSource() == b3)
{
    String s1 = tf1.getText();
    String s2 = tf2.getText();
    int x = Integer.parseInt(s1);
    int y = Integer.parseInt(s2);
    int z = x*y; String n = String.valueOf(z);
    tf3.setText(n);
}

if (ae.getSource() == b4)
{
    System.exit(0); // used to close the window at run time.
}

} // Main
}

```

} // class-outter.

```

class JTextFieldDemo
{

```

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

```
    {
        To o = new To();
    }
}
```

} // main

} // JTextFieldDemo.

## Textarea Component:

\* Textarea is one of the active component so that it participates in event delegation model.

Creating a Textarea is nothing but creating an object of Textarea class

```
Textarea ta = new TextArea();
```

\* Graphically Textarea is a collection of rows and columns.

\* When we create an object of TextArea, by default vertical scrollbar will be added and it is in disabled state and it will be enabled when the no. of rows are exceeding than the specified.

\* When the no. of columns are exceeding then horizontal scrollbar will be appended and enabled. Both vertical and horizontal scrollbars are used for to see or view the hidden no. of rows and columns of the Textarea. i.e. a Java programmer need not to write any special program for scrollbar for appending to the Textarea. In the current versions of Java scrollbar related component will be appended to the other GUI components where ever we require.

\* What ever the data we enter into Textarea, which is by default treated as string data

\* The functionality of TextArea Component is to accept the input from the user either in the form of single line of text or in the form of multiple lines of text or both.

## Textarea class:

### Data Members

① public static final int SCROLLBARS\_BOTH (=0)

② public static final int SCROLLBARS\_NONE (=1)

③ public static final int SCROLLBARS\_VERTICAL\_ONLY (=2)

④ public static final int SCROLLBARS\_HORIZONTAL\_ONLY (=3)

- The above data members are known as scrollbar visibility modifiers
- and they must be accessed with respect to name of the class ie TextArea
- The default scrollbar visibility modifier is SCROLLBARS\_VERTICAL\_ONLY

### Constructors:

① TextArea()

④ public void setRows (int)

② TextArea (int, int)

⑤ public void setCols (int)

③ TextArea (int, int, int)

⑥ public int getRows()

⑦ public int getCols()

⑧ public void setScrollbarVisibility (int)

⑨ public int getScrollbarVisibility ()

\* ⑩ public void append (String)

{ ⑪ public void setText (String)

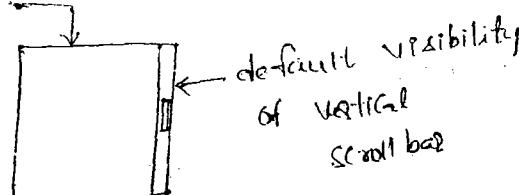
⑫ public String getText ()

⑬ public void addTextListener (TextListener)

⑭ public void removeTextListener (TextListener)

\* Constructor ① is used for creating an object of TextArea without specifying no. of rows and no. of cols and scrollbar visibility.

Eg: TextArea ta = new TextArea();



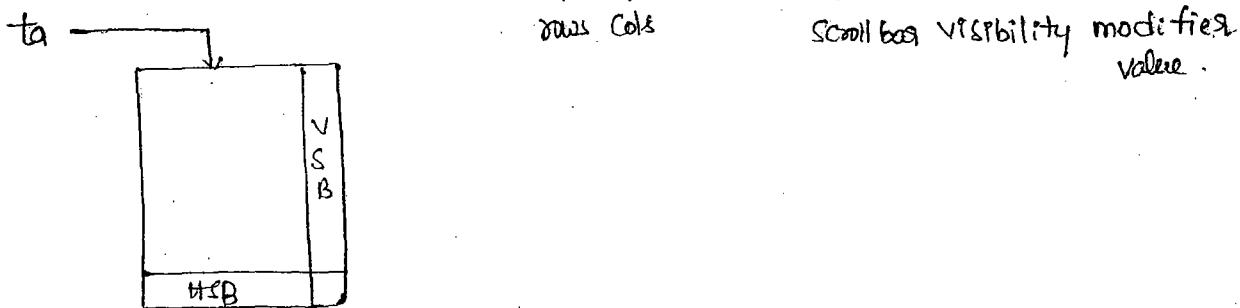
\* Constructor ② is used for creating an obj of TextArea by specifying no. of rows and cols and without specifying scrollbar visibility

Eg: TextArea ta = new TextArea(10, 100);

↑ ↑  
rows cols

\* Constructor ③ is used for creating an obj TextArea by specifying no. of rows, no. of cols and scrollbar visibility modifier value.

Eg: `TextArea ta = new TextArea(10, 100, TextArea.SCROLLBARS_BOTH);`



\* Methods ④ and ⑤ are used for setting the no. of rows and no. of cols.  
Methods ⑥ and ⑦ are used for obtaining the no. of rows and cols.

Eg: `TextArea ta = new TextArea();`

`ta.setRows(10);`

`ta.setColumns(100);`

`int rs = ta.getRows();`

`int cs = ta.getColumns();`

`System.out.println("No. of rows = " + rs); // No. of rows = 10`

`System.out.println("No. of cols = " + cs); // No. of cols = 100`

Methods ⑧ and ⑨ are used for setting and getting the scrollbar visibility modifier value to the textarea component

Eg: `TextArea ta = new TextArea();`

`int SBV = ta.getScrollbarVisibility();`

`System.out.println(SBV); // 3 (default value of SCROLLBARS_VERTICAL_ONLY)`

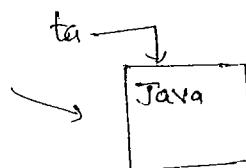
`ta.setScrollbarVisibility(TextArea.SCROLLBARS_BOTH);`

`SBV = ta.getScrollbarVisibility();`

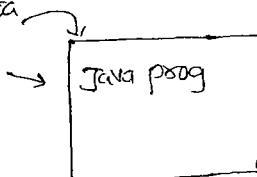
`System.out.println(SBV); // 0 (value of SCROLLBARS_BOTH)`

→ \* Method ⑩ is used for adding the data at the end <sup>to</sup> of the existing data of Textarea whereas method ⑪ is used for adding the data to the Textarea component by replacing the existing data.

Eg: `Textarea ta = new Textarea();  
ta.setText("Java");`

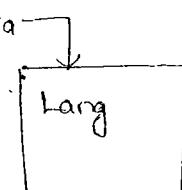


`ta.append ("prog");`



`String s = ta.getText();`

`System.out.println(s);`



`ta.setText ("Lang");`

`s = ta.getText();`

`System.out.println(s);`

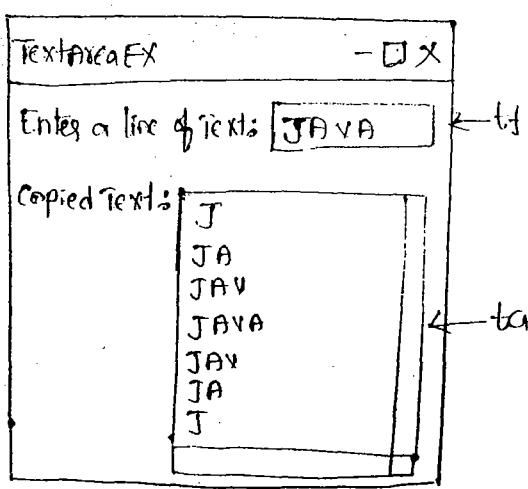
→ Methods ⑬ and ⑭ are used for registering / un-registering the events of Textarea with TextListener.

In otherwords as and when the value of the Textarea is changing

automatically textValueChanged (-) of TextListener will be called.

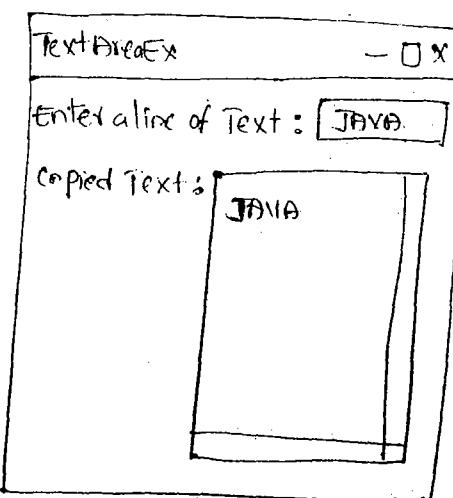
→ Write a Java program to implement the following screens.

①



HINT: use append(-)  
append(-)

②



HINT: use setText()

# '1. TextAreaEx.java'

```
import java.awt.*;
import java.awt.event.*;

public class ta extends frame
{
    Label l1, l2;
    TextField tf;
    TextArea ta;
    ta()
    {
        setTitle ("TextAreaEx");
        setSize (300, 300);
        flowLayout fl = new FlowLayout();
        setLayout (fl);
        l1 = new Label ("Enter a line of Text");
        l2 = new Label ("Copied Text");
        tf = new TextField (10);
        ta = new TextArea (2);
        ta.setRows (10);
        ta.setColumns (100);
        ta.setScrollbarVisibility (TextArea.SCROLLBARS_BOTH);
        add(l1);
        add(tf);
        add(l2);
        add(ta);
        Hyst h0 = new Hyst();
        tf.addTextListener (h0);
        setVisible(true);
    }
}
```

```

class Hyd implements TextListener
{
    public void textValueChanged (TextEvent te)
    {
        if (te.getSource() == tf)
        {
            String s = tf.getText();
            ta.append (s + "\n"); // ta.setText(s);
        } // if
    } // tvc()
}

```

} // Hyd - inner

} // outer class

```
class TextAreaEx
```

```
{
    public static void main (String[] args)
    {
        ta obj = new ta ();
    }
}
```

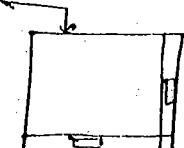
} // main

} // TextAreaEx.

### List Component:

- \* List component is one of the active component so that it participates in event delegation model.
- \* Creating a list is nothing but creating an object of List class.

Ex: List l = new List();



Here List component will have only reference but there is no caption

- The basic functionality of List Component is to select multiple items but by default the component List allows us to select single item and one can convert the List component from single item selection to multiple item selection programmatically.
- List component allows us to add the data in the form of String. List component organizes the data on the basis of 0-based indexing concept (like arrays).

List class: All the methods of choice are available as it is in List class

#### Constructor

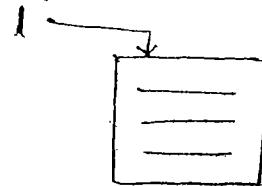
- ① List()
- ② List(int vsize)
- ③ List(int vsize, boolean)

#### Instance methods

- ④ public String[] getSelectedItems()
- ⑤ public int[] getSelectedIndexes()
- ⑥ public void setMultipleMode(boolean)
- ⑦ public boolean setMultipleModeIsSet()
- ⑧ public void addItemSelectedListener(ItemListener)
- ⑨ public void removeItemSelectedListener(ItemListener)
- ⑩ public void addActionListener(ActionListener)
- ⑪ public void removeActionListener(ActionListener)

- Constructor ① is used for creating an object of List class without specifying any visible size. By default 3 items will be visible.

Ex: List l = new List();



Constructor ② is used for creating an object of List class by specifying no. of items to be visible. If we specify the value for vsize then at the time of running vsize-1 elements will be visible.

Ex: List l<sub>1</sub> = new List(6);

Here 5 items will be visible among multiple items

- \* Constructors ① and ② are by default allowing us to select only one item from multiple items i.e. single item selection but the functionality of list component is multiple item selection.
- \* Constructor ③ is used for creating an obj of list by specifying visible size and passing the value true for the boolean to convert a list component from single item selection to multiple item selection.

Ex: List l<sub>1</sub> = new List(6, true);

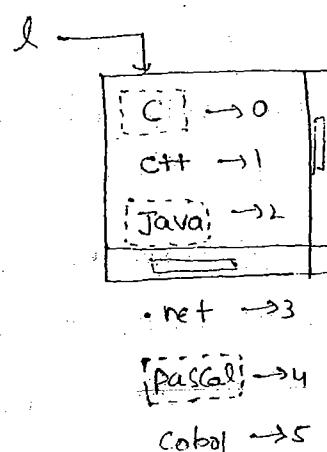
↓            ↓  
visible      multiple item  
Selection

- \* Method ④ is used for obtaining multiple items or single item from the list component.

Ex: String s[ ] = l<sub>1</sub>.getSelectedItems();

C	0
Java	1
Pascal	2

```
for (int i=0; i<s.length; i++)
    System.out.println(s[i]);
```



The above code is recommended to write for single item selection and multiple items selection.

- \* Method ⑤ is used for selecting indexes of those items which are selected at runtime.

Ex: int s[ ] = l<sub>1</sub>.getSelectedIndexes();

0	0
2	1
4	2

```
for (int i=0; i<s.length; i++)
    System.out.println(i);
```

0 2 4 are the indexes for C, Java, Pascal respectively

\* Method ⑥ is used for converting the List component from single item selection to multiple items selection. by passing true for the boolean

\* Method ⑦ is returns true provided the comp List allows us to select multiple items otherwise it returns false

Ex: List l<sub>1</sub> = new List();

List l<sub>2</sub> = new List();

l<sub>1</sub>.setMultipleMode(true);

boolean b = l<sub>1</sub>.setMultipleModeIsSet();

Sop(b); // true

b = l<sub>2</sub>.setMultipleModeIsSet();

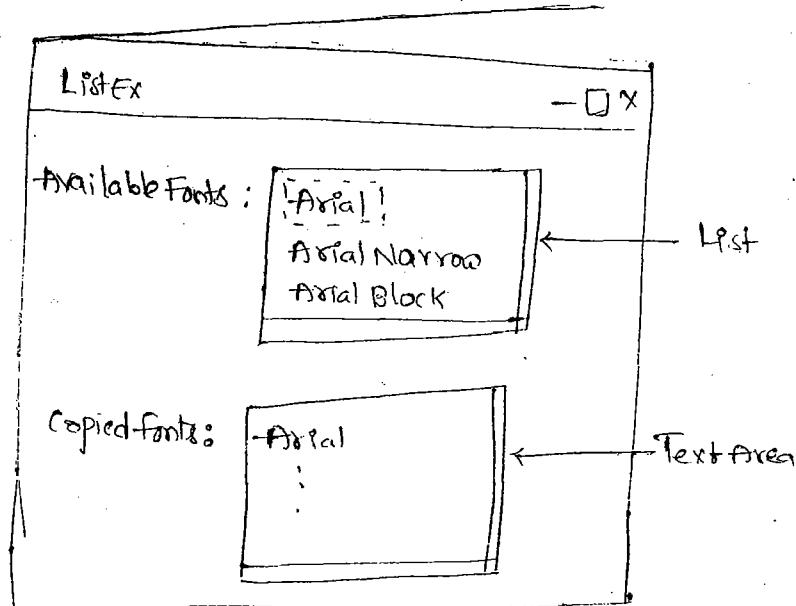
Sop(b); // false

\* Methods ⑧ and ⑨ are used for registering/unregistering the events of List component with ItemListener provided items of the List are selected with the single click.

\* Methods ⑩ and ⑪ are used for registering/unregistering the events of list component with ActionListener provided the items of the List are selected with the double click

Type of Selection Type of Click	Single Item Selection	Multiple Item Selection
Single Click	ItemListener boolean = false	ItemListener boolean = true
Double Click	ActionListener boolean = false	ActionListener boolean = true

→ Write a Java program to implement the following screen.



By using applet:

```
1 // ListEx.java
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.applet.Applet
6
7 public class ListEx extends Applet
8 {
9
10    /*<applet code="ListEx" height=300 width=300>
11     </applet> */
12
13    public class ListEx extends Applet
14    {
15        Label l1, l2;
16        List li;
17        TextArea ta;
18
19        public void init()
20        {
21            l1 = new Label("Available Fonts:");
22        }
23    }
24 }
```

```

l2 = new Label ("Copied Fonts");
l2 = new List(); - - - > l = setMultipleMode (true);
ta = new Textarea(); Graphics&Environment ge = Graphics&Environment.getGlobal
add (l1);
add (l2);
add (ta);
String f [] = getAvailableFontFamilyNames();
for (int i=0; i< f.length; i++)
    l.add (f[i]);
}

```

} // inner

public void start()

```

{
    Hyd ho = new Hyd();
    l.addItemListener (ho);
    setVisible (true);
}

```

class Hyd implements ItemListener

```

{
    public void itemStateChanged (ItemEvent ie)
}

```

if (ie.getSource () == l)

```

{
    String s = l.getSelectedIndex();
    ta.append (s);
}

```

} // if

} // inner

} // Hyd\_inner

} // class-outer

if (ie.getSource () == l)

```

{
    String s [] = l.getSelectedIndex();
    for (int i=0; i< s.length; i++)
        ta.add (s[i]);
}

```

} // if

} // inner

}

> javac ListEx.java <

> appletviewer ListEx <

## Adapter classes

- Adapter classes is one of the new concept introduced in -for saving the application development time.

### Importance of adapter classes:

Let us assume -there exists an interface with 'n' no.of abstract methods. A derived class programmer is interested in deriving inheriting the features of interfaces and interested in defining only few abstract methods and not interested in defining other abstract methods. This results in -the derived class is containing some defined methods and some undefined methods. Hence -the derived class is known as abstract and whose definition must be made as abstract by using 'abstract' keyword. Once the derived class is abstract, the derived class programmer can't create an object. So that the derived class programmer is unable to call the interested methods.

In order to call the interested methods ,the derived class programmer must define uninterested methods also by wasting lot of time. Finally -the derived class becomes concrete so that we can call the interested methods.

This approach leads to waste of time because the derived class programmes is defining unnecessarily un interested methods also. To eliminate this problem we must use the concept of adapter classes. The advantage of adapter classes is that to define only interested methods by leaving un interested methods, which results in saving of the application development time. So that productivity of the Company will be increased.

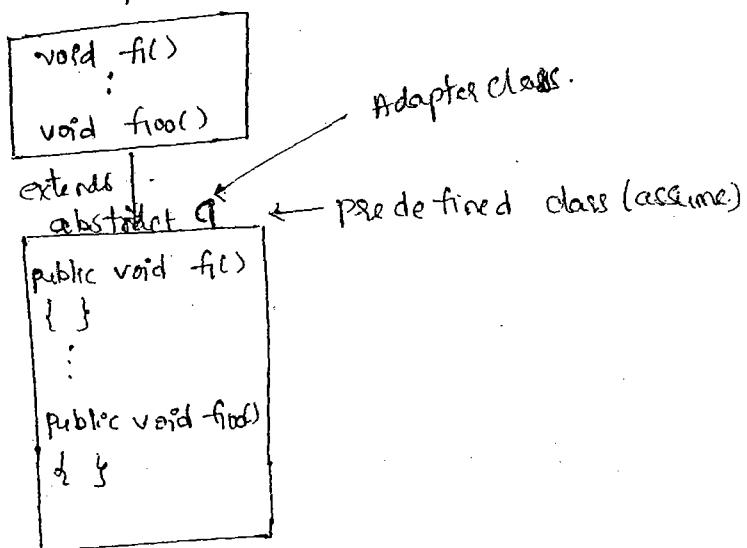
## Def of adapter class :

"An adapter class is one which contains null body methods for those abstract methods which are inheriting from predefined interfaces".

In Java programming all the predefined adapter classes are abstract classes bcoz they contains null body methods.

Ex:

$I_1 \leftarrow$  predefined interface (assume)



In windows programming of Java all the adapter classes are present in a predefined package called java.awt.event.\*.

If any predefined interface is containing more than one abstract method then for that interface there exists a predefined adapter class.

Specifically in windows programming if any XXXListener is containing more than one abstract method then there exists a predefined adapter class and whose generalized notation is XXXAdapter. The following table gives XXXListener and the corresponding XXX-Adapter.

XXXListener	XXXAdapter
ActionListener	—
ItemListener	—
WindowListener	WindowAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter

## Windows component event delegation model :

Component name = Window

Event name = WindowEvent

Listener name = WindowListener

Adapter class = WindowAdapter

Registration method = public void addWindowListener(WindowListener)

Unregistration method = public void removeWindowListener(WindowListener)

Abstract methods = 7 abstract methods

(See profile of WindowListener)

> JavaP java.awt.event.WindowListener ↴

The above registration and unregistration methods are present in

Window component.

→ Write a Java program to close the window & to terminate the window of frame application.

FileDemo.java

```
import java.awt.Frame;
import java.awt.event.*;
class Myf extends Frame
{
    Myf()
    {
        setTitle("Java in");
        setSize(400, 400);
        setVisible(true);
        Window w = new Window();
        addWindowListener(w);
        setVisible(true);
    }
}
```

```

class Hyd extends WindowAdapter
{
    public void windowClosing (WindowEvent we)
    {
        System.exit(0);
    }
}

```

(windowClosing) inner class code  
[after registration and before  
setVisible.]

### Class Edemo

```

{
    public main (String[] args)
    {
        Hyd obj = new Myfc();
    }
}

```

### Mouse handling Operations:

In windows programming of Java we have two types of components. They are

1. Logical components
2. Auxiliary components

Auxiliary components are those which are having "touch and feel" approach. In other words any hardware component used for interacting with GUI application is known as auxiliary component.

In windows programming of Java auxiliary components does not contain predefined classes

Ex: Mouse

Keyboard

- Operations of mouse are classified into two types. They are
  - 1. Basic operations of mouse
  - 2. Advanced operations of mouse.

### Basic operations of mouse:

Basic operations of mouse are classified into five types. They are

1. mouse clicked
2. mouse released
3. mouse pressed
4. mouse entered
5. mouse exited

To carry out the above operations, we have a predefined interface called java.awt.event.MouseListener.

Methods in java.awt.event.MouseListener interface:

1. public abstract void mouseClicked(MouseEvent)
2. public abstract void mouseReleased(MouseEvent)
3. public abstract void mousePressed(MouseEvent)
4. public abstract void mouseEntered(MouseEvent)
5. public abstract void mouseExited(MouseEvent)

Since MouseListener is containing more than one abstract method, there exist a corresponding predefined adapter class known as java.awt.event

### MouseAdapter

### Advanced operations of Mouse:

Advanced operations of mouse are classified into two types. They are

1. Mouse moved
2. Mouse dragged

To perform the above operations, we have a predefined interface called `java.awt.event.MouseMotionListener`

#### Methods in MouseMotionListener:

1. `public abstract void mouseMoved(MouseEvent)`
2. `public abstract void mouseDragged(MouseEvent)`

Since `MouseMotionListener` is containing more than one abstract method and hence there exist a predefined adapter class known as `java.awt.event.MouseMotionAdapter`.

#### Event delegation model for mouse Component:

Component Name : mouse

Event Name : MouseEvent

Listeners : MouseListener

                  MouseMotionListener

Abstract methods : `MouseListener` contains 5 AM

`MouseMotionListener` contains 2 AM

Reg. methods : `public void addMouseListener(MouseListener)`

`public void addMouseMotionListener(MouseMotionListener)`

UnReg. methods : `public void removeMouseListener(MouseListener)`

`public void removeMouseMotionListener(MouseMotionListener)`

Adapters classes : `MouseAdapter`

`MouseMotionAdapter`

The above registration and unregistration methods present in a class called `java.awt.Component` class.

→ Write a Java program which illustrate the concept of mouse handling operations.

1) mouseop.java

```

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

/* <applet code = "mouseop" width=300 height=300 >
</applet> */

public class mouseop extends Applet
{
    String op;
    int x,y;
    public void init()
    {
        setBackground(Color.cyan);
        setForeground(Color.red);
        x=100;
        y=100;
        op=null;
    } // init()

    public void start()
    {
        // Registration of basic op of mouse & mo: [new m1()]
        addMouseListener(new m1()); // m1 is inner class
        addMouseMotionListener(new mm1()); // mm1 is inner class
    } // start()

    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,60);
        g.setFont(f);
    }
}

```

```
g.drawString (op + " (" + x + ", " + y + ')', x, y);  
} // paint
```

```
class ml extends MouseAdapter
```

```
{
```

```
public void mouseEntered (MouseEvent me)
```

```
{
```

```
showStatus ("Mouse entered in the window");
```

```
}
```

```
public void mouseExited (MouseEvent me)
```

```
{
```

```
showStatus ("Mouse Exited from the window");
```

```
}
```

```
public void mousePressed (MouseEvent me)
```

```
{
```

```
op = "Mouse pressed";
```

```
x = me.getX();
```

```
y = me.getY();
```

```
repaint();
```

```
}
```

```
public void mouseReleased (MouseEvent me)
```

```
{
```

```
op = "Mouse Released";
```

```
x = me.getX();
```

```
y = me.getY();
```

```
repaint();
```

```
public void mouseClicked (MouseEvent me)
```

```
{
```

```
op = "Mouse Clicked";
```

```
x = me.getX();
```

```
y = me.getY();
```

```
repaint();
```

```
}
```

```
} // ml - Paint class
```

```

class mm1 extends MouseMotionAdapter
{
    public void mouseMoved (MouseEvent me)
    {
        op = "Mouse Moved";
        x= me.getx();
        y= me.gety();
        repaint();
    }

    public void mouseDragged (MouseEvent me)
    {
        op = "Mouse Dragged";
        x= me.getx();
        y= me.gety();
        repaint();
    }
}

// main - inner class

// mouse op - outer class

```

In the above program,

- ① Public void setBackground (int) } see the two predefined methods present  
Public void setForeground (int) }
- ② Public void .showStatus (String) It is the one of the predefined method  
present in Applet class and this method is used for changing the  
status bar msg of the applet
- ③ Public void int .getx() } all the two predefined methods present in  
Public int gety() Mouse Event class.

used for obtaining x and y coordinates of the mouse pointer.

④ public void repaint() it is one of the predefined method present in Applet class used for calling the paint().

## \* Collection Framework (java.util.\*)

Collection framework is one of the distinct facility available in Java programming.

Collection framework is one of the additional service or add-on service for strengthening the performance of Java and J2EE applications.

Def of Collection framework :

Collection framework is ~~an~~ one of the standardised mechanism of Java which allows us to integrate or group different type of elements or similar type of elements or both of them in a single variable.

This single variable is known as collection framework variable.

In order to deal with collection framework we need to use collection of predefined classes and predefined interfaces which are present in java.util.\* package. In afterwards to write the collection framework applications we must import a package called java.util.\*.

Q) What are the differences b/w arrays and collection framework?

### Array

An array is a collective name given to a group of consecutive memory locations used technique of Java which allows us to group different type of elements & similar type of elements or both in a single variable. This single variable is known as collection framework variable.

- ② The concept of arrays allows only homogeneous elements
- ③ The concept of arrays containing fixed size in nature.
- ② The concept of Collection frame work allows both homogeneous and heterogeneous elements
- ③ The concept of Collection frame work contains dynamic size in nature.

### Goals / aims / properties / characteristics of Collection framework:

- If we use collection frame work concept as a part of our application, we get the following goals / advantages.
- 1. Collection frame work provides high performance to the Java J2EE applications.
- 2. Collection frame work provides adaptability (the process of adding the content of one collection frame work variable to the another collection frame work variable either at the end or at the specific position).
- 3. Collection frame work provides extensibility.
- 4. Collection frame work is one of the algorithmic oriented.
  - 4.1. Collection frame work provides readily available sorting techniques
  - 4.2. Collection frame work provides readily available searching techniques
  - 4.3. Collection frame work provides readily available preliminary data structures concepts.

### Types of Collection frame works:

Collection frame works are classified into two types. They are

- \* new collection frame work
- \* legacy collection frame work

New Collection frame works: In the initial days of sun Micro System the concept of collection frame work was known as Data structures. The

Concept of DS was fulfilling few demands of the industry and few demands were not fulfilled. Hence SUN Micro systems has performed reengineering operation on DS concept and released to the industry on the name of new collection-frame work. In very short new collection-frame work is one of the renamed form of data structures.

The existing DS concept was renamed as legacy collection-frame work <sup>(old)</sup>. New collection-frame work is classified into two types. They are

1. 1D | Single collection-frame work
2. 2D | double collection-frame works | maps.

### 1D collection-frame work:

1D collection-frame work is one in which the data is organizing not in the form of (key, value) pair but it is organizing either in the form of row or in the form of column.

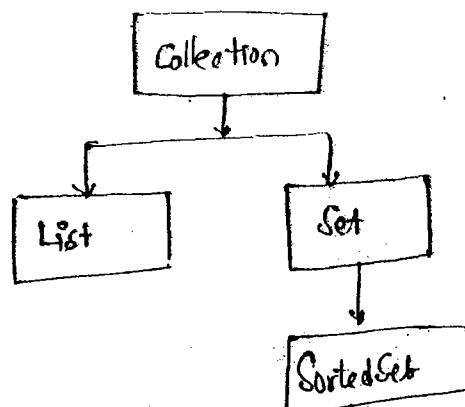
In order to deal with 1D collection-frame work related programming we need to learn about 1D collection-frame work interfaces and 1D collection-frame work class.

### 1D collection-frame work interfaces:

1D collection-frame work interfaces are divided into four types. They are

- (i) java.util.Collection
- ii) java.util.List
- iii) java.util.Set
- iv) java.util.SortedSet

### Hierarchy chart:



## Collection

### List

- The interface collection is available on the top of hierarchy of all the collection framework interfaces
- obj of collection interface allows us to place duplicate values
- An obj of collection interface always displays the data in sequential order.

- The interface List is the sub interface of collection
- An object of List interface allows use to organize the duplicate values
- An obj of List always displays the data in sequential order.

- Collection interface obj allows us to get or extract the data only in forward direction but not in backward direction

- Collection interface obj allows us to get or extract the data only in forward direction but not in backward direction or in random retrieval

in all

### Set

- The interface Set is the sub interface of collection
- An object of Set interface does not allow duplicate

- An obj of Set interface does not allow duplicate

- An obj of Set always displays the data in random order.

- An obj of Set always displays the data in sequential order.

- An obj of List interface allows us to add the data either at the end or at the specific position.

- An obj of Set interface allows us to add the data only at the end.

- An obj of Sorted Set interface allows us to add the data only in forward direction but not in backward direction.

- not in backward direction

### Sorted Set

- The interface Sorted Set is the sub interface of collection
- An object of Sorted Set does not allow duplicate

- An obj of Sorted Set does not allow duplicate

- An obj of Sorted Set always displays the data in sorted order.

- An obj of Sorted Set always displays the data in sorted order.

- An obj of Sorted Set allows us to add the data only in forward direction but not in backward direction.

- not in backward direction

- 145

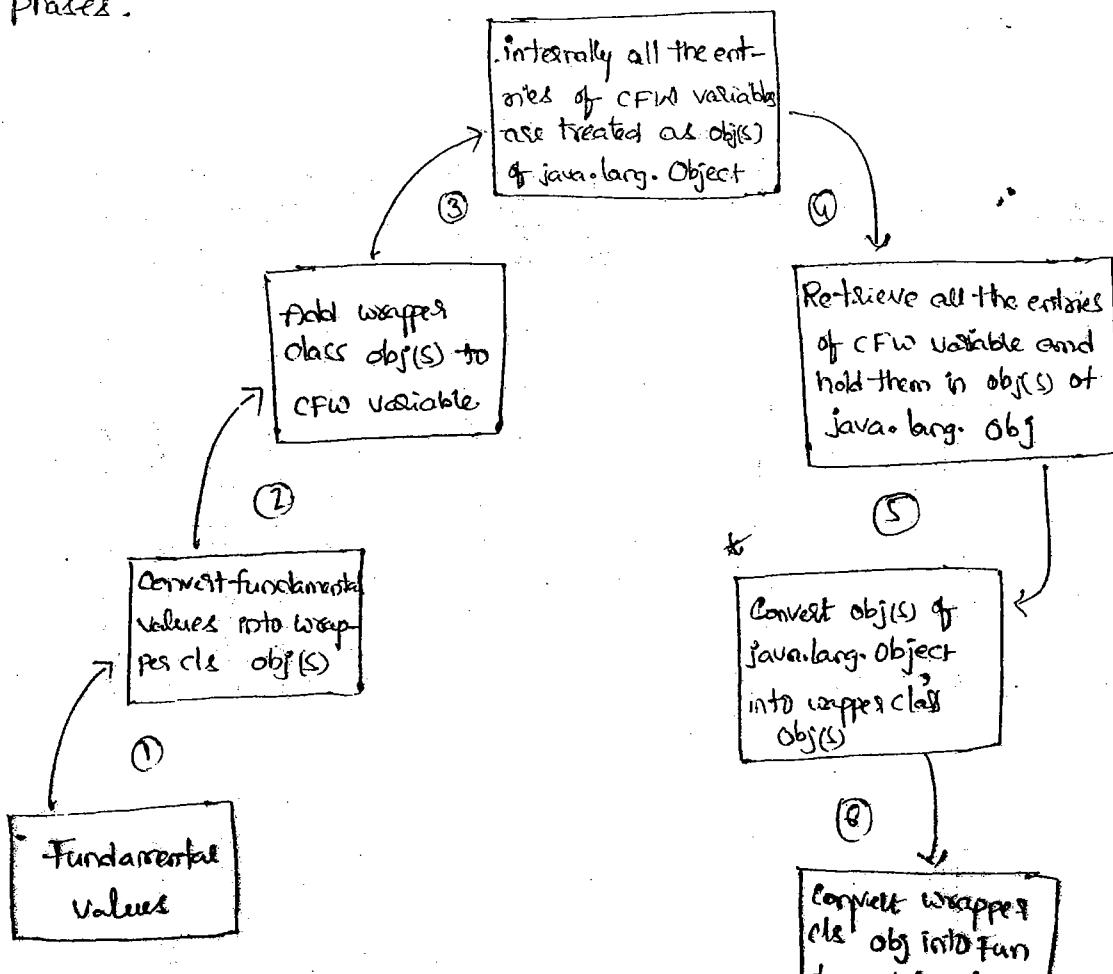
The objects of Collection, Set and SortedSet interfaces are known as unidirectional object whereas an object of List interface is known as multidirectional object.

### Collection framework process:

Collection framework work process is nothing but performing two phases. They are

1. assembling / grouping phase
2. de-assembling / un-grouping phase.

The following diagram gives sequence of steps for assembling and de-assembling phases.



The steps ①, ②, ③ are known as assembling phase and the steps ④, ⑤, ⑥ are called de-assembling phase.

⇒ Ex:

int a=10; //①

② Integer i0 = new Integer(a); //②

c.add(i0); //③

Object obj = Get the value of c; //④

Integer i0 = (Integer) obj; //⑤ → obj Type casting.

int x = i0.intValue(); //⑥

Methods In java.util.Collection Interface Collection is one of the predefined interface available on the top of hierarchy of all collection framework interface.

① public int size();

This method is used for finding no. of elements in any collection framework variable.

② public boolean isEmpty();

This method returns true provided collection framework variable is not containing any elements otherwise it returns false.

\* ③ public boolean add(Object);

This method is used for adding any value <sup>to</sup> any collection framework variable.

\* As long as we call this method with respect to Collection and List interface objects this method always returns true bcz these objects allows duplicates.

\* When we call this method with respect to Set and SortedSet interface objects and if we try to add duplicate values then this

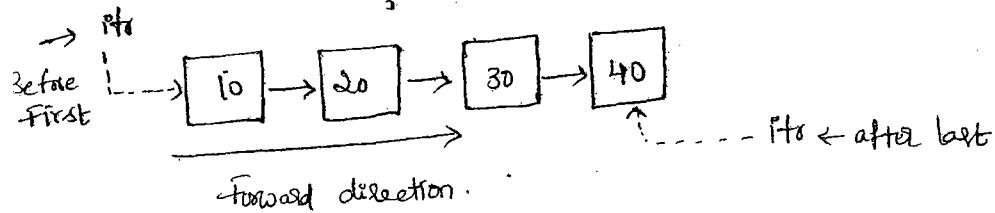
\* method returns false. bcz Set & SortedSet Obj doesn't allow duplicates

④ public Iterator iterator();

This method is used for retrieving the data from any collection framework variable in forward direction but not in backward direction.

Ex: `Sop (c); // {10,20,30,40}`

Iterator itr = c.iterator();



10
20
30
40

int s=0;

while (itr.hasNext())

{

Object obj = itr.next(); // 4<sup>th</sup> step

Integer i0=(Integer) obj; // 5<sup>th</sup> step OTC

int x = i0.intValue(); // 6<sup>th</sup> step

Sop (x);

s = s+x;

}

Sop ("Sum=" + s); // 100

⑤ public Object[] toArray();

This method is used for converting all the elements of a variable into the array of object of java.lang.Object

This method is used for extracting or retrieving the data from any Collection framework variable. This method returns array of objects of java.lang.Object and they will be processed based on 0-based indexing concept (arrays concept)

Ex: `Sop (c); // {10,20,30,40}`

int s=0;

Object obj[] = c.toArray(); // ① step

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

{}  
  {}

Integer i0=(Integer) obj[i]; // ② step OTC

int x = i0.intValue(); // ③ step

Sop (x);

s = s+x;

} Sop ("Sum=" + s); // 100

10
20
30
40

Obj	
10	0 ← i
20	1
30	2
40	3

## \* Iterator interface : (java.util.Iterator interface)

Iterator is one of the predefined interface present in java.util package

when we create an object of Iterator interface indirectly, which is by default pointing just before the first element and it allows us to retrieves the data in forward direction but not in backward direction.

### Methods:

① public boolean hasNext()

② public Object next()

Method ① returns true provided Iterator interface object is having next element, otherwise it returns false.

Method ② used for obtaining next element of any collection framework

variable with respect to ~~an~~ an object of Iterator interface provided method ① must returns true. Once method ① returns false, we can't apply method ②. One analog of Iterator interf returns false with respect to method(1), an obj

Methods in List interface: of Iterator interf is pointing after last.

The interface List is the sub interface of Collection. So that all the methods of Collection are inherited into List interface.

① public void add(int, Object);

This method is used for adding an element to any collection framework variable either at the end or at specific position.

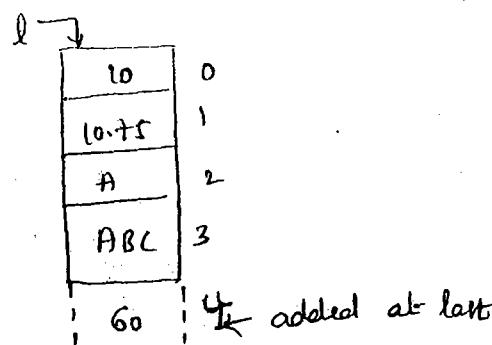
How

(Q) ~~What~~ do you add an element 60 to the existing List object at last by using add(int, Object)?

(A) l.add(l.size(), new Integer(60));

When the above stmt is executed,

l.add(4, new Integer(60))



### ④ public Object get(int);

This method is used for obtaining a value of any CFW variable in the form of object of `java.lang.Object`. The parameter int represents valid position of the CFW variable.

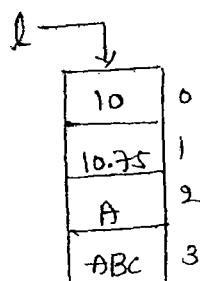
Ex:

```
Object obj = l.get(1);
```

```
System.out.println(obj); // 10.75
```

```
Object obj = l.get(10)
```

```
System.out.println(obj); // null bcoz 10 does not exist.
```



### ⑤ public Object remove(int);

### ⑥ public void remove(Object);

The above methods are used for removing the elements of any CFW variable either based on the position or based on content.

### ⑦ public void removeAll();

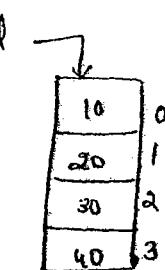
This method is used for removing all the elements of any CFW variable.

### ⑧ public ListIterator listIterator();

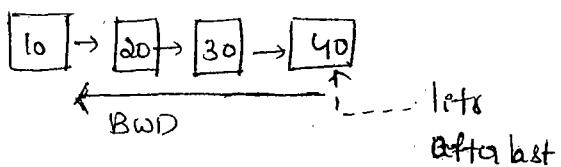
This method is used for retrieving the elements of any CFW in both forward and backward direction.

```
System.out.println(l); // [10, 20, 30, 40]
```

```
ListIterator litr = l.listIterator();
litr --> fwo
before first
 10 --> 20 --> 30 --> 40
int s=0;
while(litr.hasNext())
  {
```



```
    Object obj = litr.next(); // 4th step
    Integer iobj = (Integer) obj; // 5th step
    int x = iobj.intValue(); // 6th step
}
System.out.println(x); // 10
```



Point  $lptr$ ;

while ( $lptr$ .hasPrevious())

{ Object obj =  $lptr$ .previous(); // 4<sup>th</sup> step }

    Integer i0 = (Integer) obj; // 5<sup>th</sup> step

    i0.intValue(); // 6<sup>th</sup> step

    System.out.println(i0); // 10 20 30 40

    lptr = i0; // 7<sup>th</sup> step

}

System.out.println("Sum in BWD = " + i0);

### ListIterator interface:

ListIterator is one of the predefined sub interface of Iterator interface and it present in `java.util.*` package.

When we create an obj of ListIterator indirectly, which is by default pointing just before the first element of any collection framework variable and whose functionality is to retrieve the data from any any CFW variable in both forward and backward directions.

### Methods:

- ① public boolean hasNext()
- ② public Object next() } Inherited from Iterator interface.
- ③ public boolean hasPrevious()
- ④ public Object previous()

Method ③ returns true provided ListIterator interface object is having previous element otherwise returns false.

Method ④ used for obtaining previous element of CFW variable with respect to ListIterator interface object provided. Method ③ must return true.

(Q) What are the differences b/w Iterator and ListIterator?

### Iterator

① Iterator is one of the Super interface  
of ListIterator ② ListIterator is the sub interface of Iterator

③ An obj of Iterator allows us to retrieve the data only in Forward direction but not in back word direction.

### methods

public boolean hasNext()  
public Object next()

### ListIterator

③ An obj of ListIterator allows us to retrieve the data both in forward and back word directions

### Methods

public boolean hasPrevious()  
public Object previous()

### List methods continued:

① public object first()

② public object last()

③ public List headList(object obj) →  $x_i \leq obj$

④ public List tailList(object obj) →  $x_i > obj$

⑤ public List subList(object obj1, object obj2) →  $obj1 \leq x_i < obj2$

→ Methods ① and ② are used for obtaining first and last elements in the List.

→ Method ③ is used for obtaining those values ( $x_i$ ) which are less than or equal to target object. Mathematically

$$x_i \leq obj$$

→ Method ④ is used for obtaining those values ( $x_i$ ) which are greater than the target object. Mathematically

$$x_i > obj$$

In other words methods ③ & ④ are used for data-  
ing few top most and few bottom most values of the List.

→ Method ⑤ is used for obtaining range of values from the List.

In other words this method retrieves those values ( $x_i$ ) which are greater than or equal to target object obj1 and less than target object obj2. mathematically

$$\text{Obj1} \leq x_i < \text{Obj2}$$

Ex:

Object obj = l.first();

① [10]

10
20
30
40
50
60
70
80

List hl = l.headList(new Integer(30))

10
20
30

③ ④ ⑤ List sl = l.subList(new Integer(40), new Integer(70));

40
50
60

List tl = l.tailList(new Integer(60));

70
80

### Methods in java.util.Set interface:

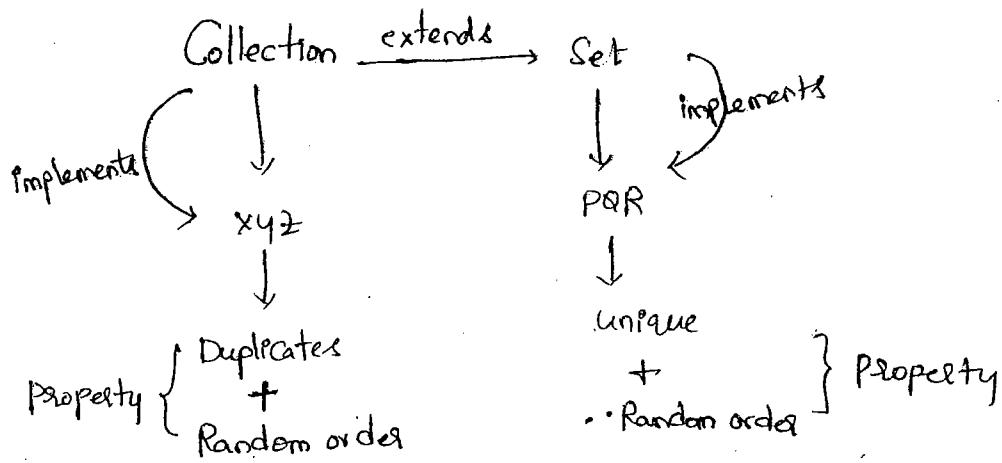
Set interface is one of the sub interface of Collection interface. So that all the methods of collection are inherited into Set interface.

The interface Set does not contain any special methods except Collection interface methods.

\* Even though methods of Collection and Set are same, the methods of Collection interface are defined in some predefined class in such a way that duplicate elements are allowed. The methods of Set interface are defined in some other predefined class in such a way that unique elements are allowed.

In other words, even though methods of Collection and Set are same but their implementations / definitions are diff in different predefined

## Classes.



## Methods in `java.util.SortedSet`:

- \* `SortedSet` is one of the sub interfaces of `Set` interface so that all the methods of `Set` are inherited into `SortedSet` + `SortedSet` containing some special methods on its own.
- \* Even though methods of `Set` and `SortedSet` are same, `Set` methods are defined in some predefined class in such a way that unique elements are allowed and displaying in random order.  
 `SortedSet` methods are defined in some other predefined class in such a way that unique elements are allowed and displaying in Sorted order.

- ① `public Object first();`
  - ② `public Object last();`
  - ③ `public SortedSet headSet ( Object obj) →` 
$$x_i \leq obj$$
 target obj
  - ④ `public SortedSet tailSet ( Object obj) →` 
$$x_i > obj$$
  - ⑤ `public SortedSet subSet( Object obj1, Object obj2) →` 
$$obj_1 \leq x_i < obj_2$$
- ( See continued methods of List)

## 1D Collection Framework classes:

1D collection framework classes are those which are containing definition or implementation for those abstract methods which are inherited from 1D CFW interfaces.

All 1D CFW classes are presents in `java.util.*` package. The following table gives 1D CFW interface name, its corresponding class name and its hierarchy.

<u>1D CFW intf name</u>	<u>1D CFW clsnme</u>	<u>Hierarchy</u>
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	extends AbstractSequentialList
	⑥ ArrayList	extends AbstractSequentialList
	⑦ HashSet	extends AbstractSet
	⑧ TreeSet	extends AbstractSet extends SortedSet

The classes ①, ②, ③, ④ are abstract predefined classes and their objects can't be created directly. So that we may not be using them directly in our real time applications. But their client classes

are playing an important role in defining the abstract methods of 1D CFW interface and supplying them to concrete sub classes of 1D CFW. In other words these abstract classes provide a middle man role in predefined interfaces of 1D CFW and concrete sub classes of 1D CFW.

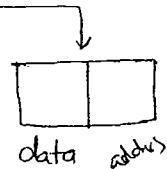
The classes ⑤, ⑥, ⑦, ⑧ are treated as bottom most concrete sub classes of 1D CFW so that we can create their objects directly and hence we use them in real-time applications.

### LinkedList :

LinkedList is one of the bottom most concrete subclass of all CFW classes.

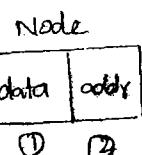
Creating a LinkedList is nothing but creating an object of LinkedList class.

```
LinkedList ll = new LinkedList();
```



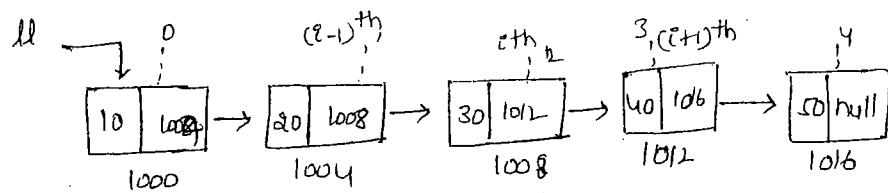
Here ll is one of the CFW variable, which allows us to group similar type of values or different type of values & both and it can be treated as name of the node.

LinkedList organizes the data in the form of nodes in the main memory. A node contains two parts and whose structure is given below.



Here data part represents type of data being represented in LinkedList class object. Addr part represent address of the next node and for the last the value of the address part must be null which indicates end of the linked list.

Ex: Represent 10, 20, 30, 40, 50 in a `LinkedList` class variable.



The principle followed by the LL to organize the data in the form of nodes is the address of  $i^{\text{th}}$  node stored in the address part of  $(i-1)^{\text{th}}$  node, address part of  $i^{\text{th}}$  node contains the address of  $(i+1)^{\text{th}}$  node and address part of last node contains null which indicates end of the LL.

The advantage of LL over arrays concept is that we can insert the data either in the beginning or at the end or in the specific position. That is LL allows dynamic insertions but not in arrays.

### LinkedList class profile:

#### Constructors

- ① `LinkedList()`
- ② `LinkedList(int size)`

#### Instance methods :

- 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()`

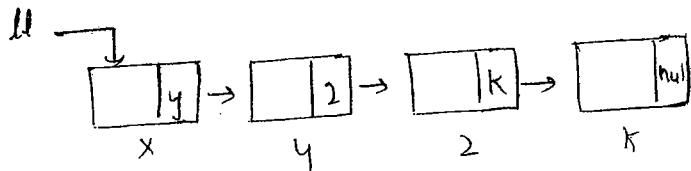
\* Constructor ① is used for creating an obj of LL without specifying any size.

Ex: `LinkedList ll = new LinkedList()`

\* Constructor ② is used for creating an object of LL by specifying the no. of nodes.

In realtime applications parameterized constructors of CFW are not recommended to use.

Ex: `LinkedList ll = new LinkedList(4);`



\* Methods ⑤ and ⑥ are used for adding an element to the `LinkedList` at beginning and last respectively.

Ex: `ll.addFirst(new Integer(5));`

(or)

`ll.add(0, newInteger(5));`

`ll.addLast(new Integer(50));`

(or)

~~`ll.add(ll.size() new, Integer(50));`~~

\* Methods ⑦ and ⑧ are used for removing first and last elements of the LL respectively.

Ex: `Object fojb = ll.removeFirst();`

(or)

`Object fojb = ll.remove(0);`

`Object lobj = ll.removeLast();`

`Object lobj = ll.remove(ll.size() - 1);`

\* Methods ⑨ and ⑩ are used for obtaining first and last elements of LL

`Object fojb = ll.getFirst();`

(or)

`Object fojb = ll.get(0);`

`Object lobj = ll.getLast();`

(or)

`Object lobj = ll.get(ll.size() - 1);`

\* Write a Java program which illustrate the concept of Linked List

// LL.java

```

import java.util.*;
class LL
{
    public static void main(String[] args)
    {
        LinkedList ll = new LinkedList();
        System.out.println("Size = " + ll.size()); // 0
        System.out.println("Content = " + ll); // []
        // add the data to the ll
        ll.add(new Integer(20));
        ll.add(new Integer(30));
        ll.add(new Integer(40));
        System.out.println("Size = " + ll.size()); // 3
        System.out.println("Content = " + ll); // [20, 30, 40]
        ll.addFirst(new Integer(10));
        ll.addLast(new Integer(50));
        System.out.println("Size = " + ll.size()); // 5
        System.out.println("Content = " + ll); // [10, 20, 30, 40, 50]
        // retrieving the data from LL
        System.out.println("Retrieving the data from LL-->iterator()");
        Iterator itr = ll.iterator();
        int s=0;
        while(itr.hasNext())
        {
            Object obj = itr.next();
            Integer i0 = (Integer)obj;
            int x = i0.intValue();
            System.out.println(x);
            s=s+x;
        }
    }
}

```

Sop C "Sum - iterator() = " + s1;

Sop C "Retrieving the data from LL - listIterator - FWD";

ListIterator litr = ll.listIterator();

while (litr.hasNext())

{

Object obj1 = litr.next();

Sop C obj1;

}

// at this stage 'litr' is pointing to after last

int s1 = 0;

Sop C "Retrieving the data from LL - listIterator - BWD";

while (litr.hasPrevious())

{

Object obj2 = litr.previous();

Integer iobj2 = (Integer) obj2;

int x1 = iobj2.intValue();

Sop C x1;

s1 = s1 + x1;

}

Sop C "Sum - listIterator() = " + s1;

Sop C "Retrieving the data from LL - toArray();";

Object obj3[] = ll.toArray();

int s2 = 0;

for (int p=0; p < obj3.length; p++)

{

Integer iobj3 = (Integer) obj3[p]; // OTC

int x2 = iobj3.intValue();

Sop C x2;

s2 = s2 + x2;

}

Sop C "Sum - toArray() = " + s2;

```

    // Random get & retrieval
    Object robj = ll.get(2);
    System.out.println("Random element = " + robj); // 30
} // main
} // LL

```

### Limitations of Linked List:

When we use LinkedList as a part of our application development we may get the following limitations.

1. explicit memory space is created for the address part in main memory that is more space is created for address part bcoz our data is represented in the form of nodes.
2. Retrieving the data from LinkedList takes considerable amount of time i.e. retrieval time is more.

### ArrayList:

ArrayList is one of the bottom most concrete subclass of all collection framework classes.

Creating an ArrayList is nothing but creating an object ArrayList class

```
ArrayList al = new ArrayList();
```

Here 'al' is an object of ArrayList treated as CFW variable which allows us to group different type & similar type of elements.

In ArrayList the data is organizing in the form of cells. Cell values stored in main memory. Cell addresses are stored in associative memory.

### Advantages of AL over LL:

1. Memory space is less for organizing the data in AL bcoz cell addresses are stored in associative memory.

2. Retrieving the data from AL takes negligible amount of time ie retrieval time is less.

### ArrayList class profile:

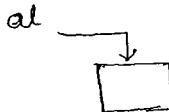
1. ArrayList()

2. ArrayList(int size)

3. ArrayList(int size, int incr ratio)

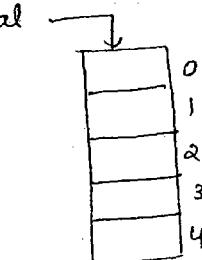
→ Constructor ① used for creating an object of ArrayList without specifying the size. The way which we allocate the cells with this object creation is known as late-cell-allocation.

Eg: ArrayList al = new ArrayList();



→ Constructor ② is used for creating an object of ArrayList by specifying the size which represents no. of cells to be created. The way which we allocate the cells in advance is known as advanced-cell-allocation.

Eg: ArrayList al = new ArrayList(5);



→ Constructor ③ is used for creating an object of ArrayList by specifying no of cells to be created and increment ratio for adapting the process of pre-cell-allocation for optimizing the functionality of ArrayList.

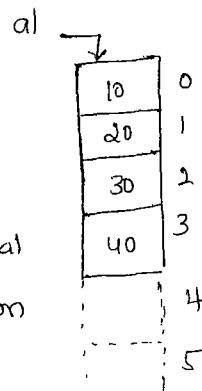
The formula for pre-cell allocation is

$$\text{Size of AL} = \text{size} + \text{incr ratio}$$

This formula makes us to understand restructuring the size of AL when the no. of entries in AL is equal to the size of the AL.

Ex: `ArrayList al = new ArrayList(4, 2);`

↓      ↓  
size    increment



- The no. of entries in AL are 4 which is equal to size of AL. Hence JVM will call pre-alloc allocation formula for restructuring the size of AL

$$\text{Size of AL} = 4 + 2 = 6$$

→ Write a Java program which illustrate the concept of ArrayList (AL.java)

Hint: Sub the following starts in the previous program (LL.java)

- Substitute ArrayList in place of LinkedList
- Substitute ll.add(0, new Integer(10)) in place of ll.addFirst(new Integer(10))
- Sub ll.add(ll.size, new Integer(50)) in place of ll.addLast(new Integer(50))

### HashSet and TreeSet:

HashSet and TreeSet are the bottom most concrete subclasses of all CFW classes.

#### HashSet

- An obj of HashSet never allows duplicates

\* HashSet organizes the data in hash table by following hashing mechanism

\* We can't determine in which order HashSet object displays the data. bcz SUN Micro System did not disclose which hashing mechanism is followed in HashSet

#### TreeSet

- An object of TreeSet never allows duplicates

\* TreeSet organizes the data by following binary trees concept

\* TreeSet class object always displays the data in sorted order.

### HashSet

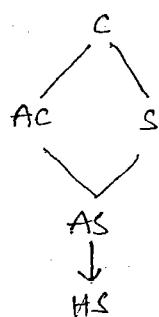
- ④ The operations like insertion, deletion, and modification takes considerable amount of time ie expensive operations.

- ⑤ Retrieving the data from HashSet takes considerable amount of time i.e. retrieval time is慢 (slow).

- ⑥ Creating HashSet is nothing but creating an obj of HashSet class.

Ex: HashSet hs = new HashSet();

### Hierarchy



### TreeSet

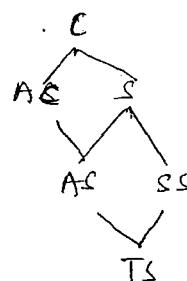
- ④ The operations like insertion, deletion, and modification takes negligible amount of time ie less expensive.

- ⑤ Retrieving the data from TreeSet takes negligible amount of time ie retrieval time is less / fast.

- ⑥ Creating TreeSet is nothing but creating an object of TreeSet class.

Ex: TreeSet ts = new TreeSet();

### Hierarchy:



→ Write a Java program which illustrate the concepts of HashSet & TreeSet

11 hsts.java

```

import java.util.*;
class hsts
{
    public static void main (String[] args)
    {
        // HashSet hs = new HashSet();
        TreeSet hs = new TreeSet();
        System.out.println("Size=" + hs.size()); // 0
        System.out.println("Content=" + hs); // []
      }
    
```

// add the data

```
hs.add (new Integer(100));
hs.add (new Integer(1));
hs.add (new Integer(10));
hs.add (new Integer(90)); hs.add (new Integer(90));
System.out.println("Size = " + hs.size()); // 4
System.out.println("Content = " + hs); // [100, 1, 10, 90]
```

// retrieving the data

```
Iterator itr=hs.iterator();
```

```
int s=0;
```

```
while (itr.hasNext())
```

```
{
```

```
Object obj=itr.next();
```

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

```
Integer io=(Integer)obj;
```

```
int x=io.intValue();
```

```
s=s+x;
```

```
} // while
```

```
System.out.println("Sum = " + s);
```

```
} // main
```

} // class

### 2D | Double CFW | maps :

2D CFW is one in which the data is organizing in a single variable in the form of (key, value).

In (key, value) the values of key must distinct and the value of value may & may not be distinct.

Ex: Deposit

10	1.5
20	1.9
30	3.9

Here Deposit is the name of 2D CFW variable.

(10, 20, 30) are called set of keys.

(1.5, 1.9, 3.9) are called set of values.

\* In the (key, value), the values of both key and value must be objects.

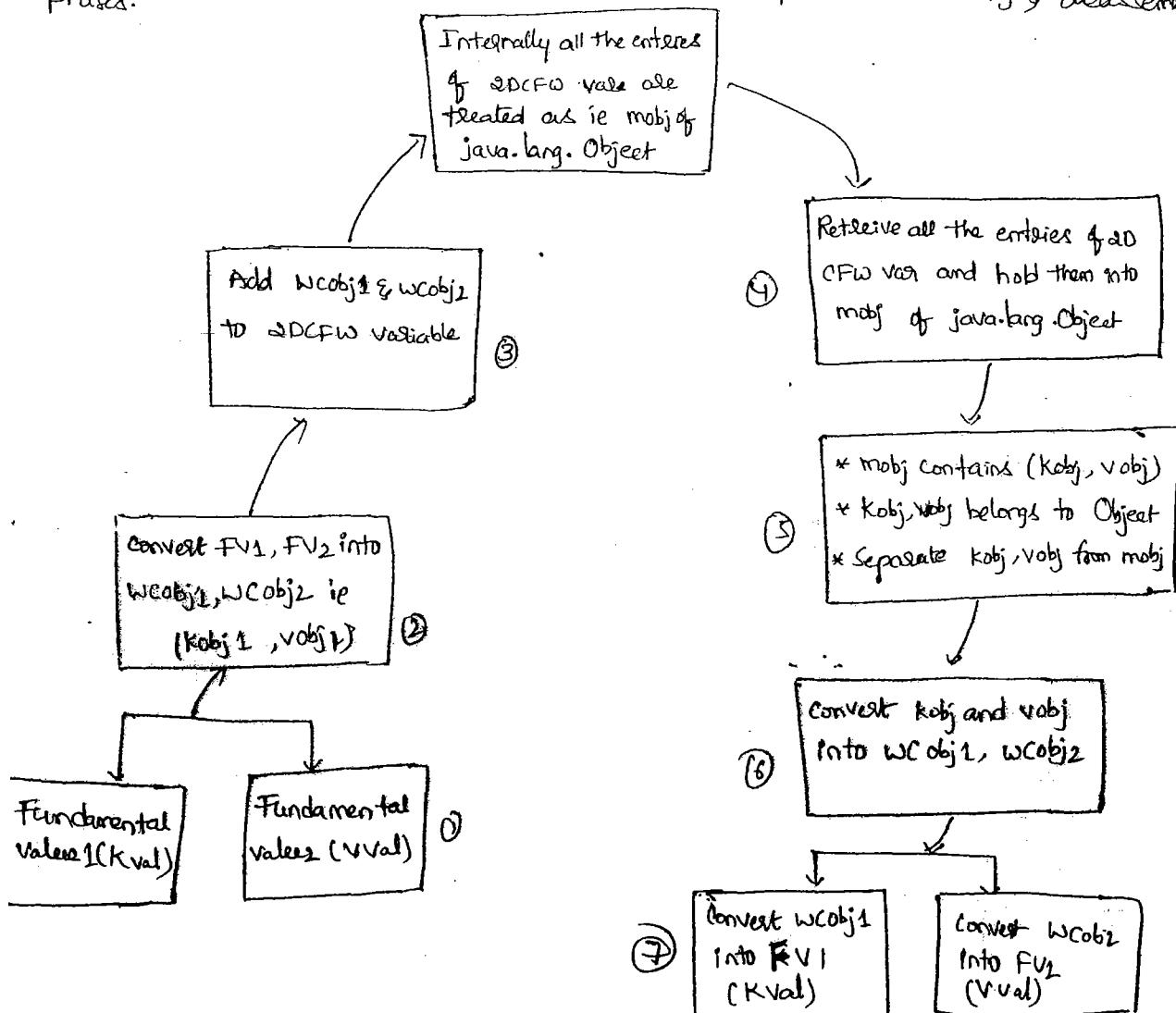
In order to deal with 2D CFW we must learn interfaces and classes and they are present in java.util.\*.

### 2D CFW process:

2D CFW process contains 2 phases. They are

1. assembling / grouping phase
2. de-assembling / ungrouping phase.

The following diagram gives the sequence of steps for assembling & deassembling phases.



In the above diagram the steps ①, ②, ③ will perform assembling phase and the steps ④, ⑤, ⑥, ⑦ will perform deassembling phase.

### 2D CFW interfaces or map interfaces:

Map interfaces are classified into 3 types. They are

1. `java.util.Map`
2. `java.util.Map.Entry`
3. `java.util.SortedMap`

#### `java.util.Map`:

Map is one of the predefined interface whose variable can be created indirectly and it allows us to organize the data in the form of (Key, value). Whatever ever the data we add to the Map interface variable, which is by default displayed in random order.

#### Methods in `java.util.Map` interface:

##### 1. `public int size()`:

This method is used for determining or counting the no. of elements in 2D CFW variable.

##### 2. `public boolean isEmpty()`:

This method returns true provided 2D CFW variable does not contain any elements (empty) otherwise it returns false (non-empty)

##### 3. `public void put (Object kobj, Object vobj)`:

This method is used for adding the data to 2D CFW variable in the form of (Key, value).

##### 4. `public Object get (Object kobj)`:

This method is used for obtaining value of Value by passing Key value.

If key value is found in 2D CFW then we get the value of value. If key value is not found in 2D CFW variable then value of value is null.

Ex: Object vobj = m.get (new Integer(0));

[1.5]

Object vobj = m.get (new Integer(100));

[null]

### ⑤ public Set entrySet();

This method is used for extracting & retrieving the data from an Collection  
able.

Ex: Sop(m); // { (0=1.5), (20=2.5), (30=3.5) }

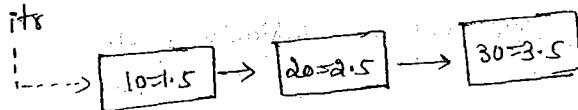
Set s = m.entrySet();

// s = { (0=1.5), (20=2.5), (30=3.5) }

// entrySet() - set - iterator()

Iterator itr = s.iterator();

itr



while (itr.hasNext())

{

Object obj = itr.next(); // 4<sup>th</sup>

Map.Entry me = (Map.Entry) obj;

Object kobj = me.getKey();

Object vobj = me.getValue(); } 5<sup>th</sup>

Integer iobj = (Integer) kobj;

Integer vobj =

Float fo = (Float) vobj; } 6<sup>th</sup>

int acno = fo.intValue();

float bal = fo.floatValue();

Sop ("bal + " is the value of " + acno);

key	value
10	1.5
20	2.5
30	3.5

m	↓
10	1.5
20	2.5
30	3.5

// entrySet - Set - toArray()

Object mobj[] = s.toArray(); // 4<sup>th</sup>  
for (int i=0; i < mobj.length; i++)  
{

Map.Entry me = (Map.Entry) mobj[i];

Object kobj = me.getKey();

Object vobj = me.getValue();

Integer i0 = (Integer) kobj;

Float fo = (Float) vobj;

int acno = i0.intValue();

float bal = fo.floatValue();

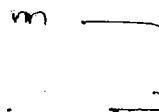
SOP ("bal + " is the value of " + acno);

}

mobj	]	
10	1.5	0
20	2.5	1
30	3.5	2

#### 6. public Set keySet():

This method is used for obtaining set of keys & obtaining  
in an object of set interface



10	1.5
20	2.5
30	3.5

ex: SOP(m); // {10 = 1.5}, {20 = 2.5}, {30 = 3.5}

Set s = m.keySet();

// s = {10, 20, 30}

// KeySet s = m.keySet();

Iterator ite = s.iterator();

ite  
10 → 20 → 30

while (ite.hasNext())

{ Object kobj = ite.next();

Object vobj = ite.next();

1.5

```

    Integer accobj = (Integer) kobj;
    float balobj = (Float) vobj;
    int acno = accobj.intValue();
    float bal = balobj.floatValue();
    System.out.println("bal + " + " is the value of " + acno);
}

```

Ex 2: with toArray() approach

```

Set s = m.keySet();
// s = {10, 20, 30}
// KeySet() -- Set -- toArray()
Object kobj[] = s.toArray();

```

10
20
30

```

for (int i=0; i < kobj.length; i++)
{
    Object vobj = m.get(kobj[i]);
    Integer accobj = (Integer) kobj[i];
    float balobj = (Float) vobj[i];
    int acno = accobj.intValue();
    float bal = balobj.floatValue();
    System.out.println("bal + " + " is the value of " + acno);
}

```

java.util.Map.Entry: Map is an interface & Entry is the predefined class defined in Map Interface

Map.Entry is used for separating key & value from key,value pair from any SD-collection framework variable.

### Methods:

1. public Object getKey();

2. public Object getValue();

Methods 1, 2 are used for retrieving key & values from (key,value)

2. java.util.SortedMap: Sorted Map is one of the predefined sub interface of map interface. An object of sorted map also allows use to organize the data in the form of (key,value)

In whatever the data we add to the SortedMap object, which is by default displayed in sorted order & sorting can be done based on the key.

### Methods:

1. public Object first()

2. public Object last()

3. public SortedMap headMap (Object kobj)  $\rightarrow x_i \leq kobj$

4. public SortedMap tailMap (Object kobj)  $\hookrightarrow x_i > kobj$

5. public SortedMap subMap (Object kobj1, Object kobj2)

$\hookrightarrow kobj_1 \leq x_i \leq kobj_2$

Sm

Ex:

object Smap = Sm.first();	10	1.5	Sorted map	Km = Sm.headMap(new Integer(10))
(10)	20	2.5		
	30	3.5		
	40	4.5		
	50	5.5	Sorted map	Smp1 = Sm.subMap
	60	6.5		new Integer(40), new Integer(60))
	70	7.5		
	80	8.5		
	90	9.5	SortedMap tm = Sm.tailMap(new Integer(90))	
Object Smap = Sm.last();	(90)			

## 2D-CFW (or) Map classes

2D-CFWI name	2D-CFW class name hierarchy
(1) java.util.map	(1) AbstractMap $\leftarrow$ Pimpl Map
(2) java.util.Map.Entry	—
(3) java.util.SortedMap	(2) AbstractSortedMap $\leftarrow$ extends AbstractMap Pimpl SortedMap (4) HashMap $\leftarrow$ extends AbstractMap (5) TailMap $\leftarrow$ extends AbstractSortedMap

2D CFW classes are those which contains definition for those abstract methods which are inheriting from 2D CFW interfaces. The above table gives 2D CFW interface name, whose corresponding class name & their hierarchy.

### HashMap & TreeMap:

HashMap & TreeMap are the bottom most concrete sub classes of 2D CFW classes.

#### HashMap

- HashMap class object organizes the data in the form of key,value with out duplicate key values
- HashMap class object follows 'hashing mechanism' to organize the data in the form of key,value pair.
- 3. We can not determine in which order HashMap obj displays the data bcoz the Java programmers is followed by sun micro system in hashing implementation.

#### TreeMap

- TreeMap class obj also organizes the data in the form of (key,value) pair without duplicate key value.
- d. TreeMap class obj organizes the data in the form of (key,value) by following Binary Tree & concept
- 3. An obj of TreeMap always displays the data in sorted order and sorting can be done based on the key

- 4. The operations like insertion, deletion & modification takes considerable amount of time i.e. expensive operations.
- 5. Retrieving the data from HashMap takes more time.
- 6. Creating a HashMap is nothing but creating an obj of HashMap class.

Ex: `HashMap hm = new HashMap();`

→ Write a Java program which illustrate the concept of HashMap & TreeMap

1) hmtm.java

```
import java.util.*;
class hmtm
{
```

    public static void main (String[] args)

    {

        HashMap hm = new HashMap();

        TreeMap tm = new TreeMap();

        System.out.println ("Size = " + hm.size());

        System.out.println ("Content = " + hm);

        // add the data

        hm.put (new Integer(100), new Float(1.5f));

        hm.put (new Integer(11), new Float(2.5f));

        hm.put (" " " 90 , " " (1.5f));

        " " ( " 40 , " " (4.5f));

        " , ( " (60) , " " (9.5f)); // duplicate

        // values are not allowed here 1.5f is replaced by 9.5f for the key, 10

```

Sop(" size = " + hm.size()); // 4
Sop("Content = " + hm); // { --- }

// extracting the data from an CFW variable
Sop(" Extracting the data ... entrySet() -- Set -- iterator()");
Set s = hm.entrySet();
Iterator itr = s.iterator();
while (itr.hasNext())
{
    Object obj = itr.next();
    Map.Entry me = (Map.Entry) obj; // OTS
    Object kobj = me.getKey();
    Object vobj = me.getValue();
    Integer i0 = (Integer) kobj;
    Float f0 = (Float) vobj;
    int acno = i0.intValue();
    float bal = f0.floatValue();
    Sop(" bal " + "---->" + acno);
}

Sop(" Extracting the data ... keySet() -- set -- toArray()");
Set s1 = hm.keySet();
Object kobj1[] = s1.toArray();
for (int i=0; i < kobj1.length; i++)
{
    Object vobj1 = hm.get(kobj1[i]);
    Integer i01 = (Integer) kobj1[i];
    Float f01 = (Float) vobj1[i];
    int acno1 = i01.intValue();
    float bal1 = f01.floatValue();
    Sop("bal1 " + "---->" + acno1);
}
} // main
} // class

```

## Legacy CFW:

- In the earlier days of SunMicro System, there was a concept called Datastructures (currently we called CFW). The concept of DS is fulfilling some of the demands of the industries and unable to fulfilling the some other demands of the industries. Hence SunMicro System has performed re-engineering process on existing DS Concept and related to the industry on the name of new CFW & the existing DS was renamed as Legacy Collection Framework i.e revised form of DS is known as new CFW and renamed form of DS is known as Legacy CFW.

Like new CFW, Legacy CFW also contains 1D CFW & 2D CFW with collection of interfaces & classes which are present in `java.util.*`

Legacy CFW contains the following interfaces & classes :-

- |                 |        |
|-----------------|--------|
| (1) Enumeration | 1D CFW |
| (2) Vector      |        |
| (3) Stack       |        |
| (4) Dictionary  | 2D CFW |
| (5) HashTable   |        |
| (6) properties  |        |

### (1) java.util.Enumeration:

Enumeration is one of the predefined interface which is used for extracting or retrieving the data from Legacy Collection Framework.

The functionality of Enumeration is extracting exactly similar to Iterator interface.

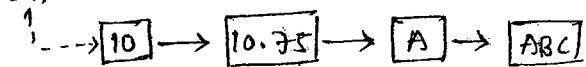
→ An object of Enumeration is by default pointing just before the first element of any LCFW variable.

#### Methods:

- (1) public boolean hasMoreElements()
- (2) public Object nextElement()

Method (1) returns true provided LCFW variable is having next element otherwise it returns false.

Method (2) is used for obtaining next element provided method (1) must returns true.

Ex: 

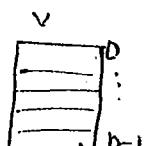
while (*en*.hasMoreElements())

```
{  
    Object obj = en.nextElement();  
    Op(obj);  
}
```

## (2) java.util.Vector:

- Vector is one of predefined 1D LCFW class.
- The functionality of Vector is exactly similar to ArrayList
- In Vector the Data is organizing in the form of cells & cell values are stored in main memory & cell addressed <sup>are</sup> or sorted in associative memory (like ArrayList)
- The value of the cells can be processed based on 0-based Indexing concept (Arrays)
- Creating a vector is nothing but creating an object of Vector class.

Vector *v* = new Vector();



### Constructors

- (1) *vector()*
- (2) *vector(int size)*
- (3) *vector(int size, int incrementor)*

### Methods:

- (1) *public void addElement(Object)*
- (2) *public void addElement(Object int)*
- (3) *public Object removeElement (int)*

- (7) public void removeElement (Object)
- (8) public void setElementAt (int, ~~element~~ Object)
- (9) public int size()
- \* (10) public Enumeration elements()
- (11) public int capacity()

Note: When we create an object of Vector without specifying any size whose capacity is 10 cells.

Constructor (3) is used for creating an object of vector by specifying default no. of cells to be created along with increment ratio.

When the no. of entries in vector is equal to size of the vector then JVM will call the precell allocation formula.

PreCell Allocation formula in vector is

$$\text{Size} = \text{Size} + \text{IncrRatio}$$

Ex: Vector v = new Vector (4, 2);

10	0
20	1
A	2
BBC	3

The no. of entries in vector 'v' is equal to the size of the vector i.e. 4 then

$$\text{Size} = 4 + 2 = 6$$

10	0
20	1
A	2
BBC	3
	4
	5

} PreCell Allocation entries

Methods (4), (5) are used for adding the elements either at the end or at the specific position.

Methods (6) & (7) are used for removing an element either based on the position & on the basis on content

Method (8) used for replacing an element by specifying its position.

Method (9) is used for finding no. of values present in the cells & method (11) gives total no. of cells available.

Ex: Vector v = new Vector(5);

v.addElement (new Integer (5));

10	0
	1
	2
	3
	4

```

    v.addElement (new Integer(5));
    System.out.println ("Size = " + v.size()); // 1
    System.out.println ("Capacity = " + v.capacity()); // 5

```

Method **(D)** is used for retrieving the elements from vector.

→ W.A.J.P which illustrate the concept of vector?

/\* vect.java

```

import java.util.*;
class Vect
{
    public static void main (String [] args)
    {

```

```

        Vector v = new Vector();
        System.out.println ("Size = " + v.size()); // 0
        System.out.println ("Capacity = " + v.capacity()); // 10
        System.out.println ("Content = " + v); // []

```

Adding the data to the vector class obj

```

        v.addElement (new Integer(10));
        v.addElement (new Float(10.75f));
        v.addElement (new Character('A'));
        v.addElement (new Boolean(true));
        System.out.println ("Size = " + v.size()); // 4
        System.out.println (v); // [10, 10.75, A, true]

```

/\* extract the data from vector

```

        Enumeration en = v.elements();

```

```

        while (en.hasMoreElements())
        {

```

```

            Object obj = en.nextElement();
            System.out.println (obj);
        }

```

} /\* main

} /\* class.

### 3) java.util.Stack

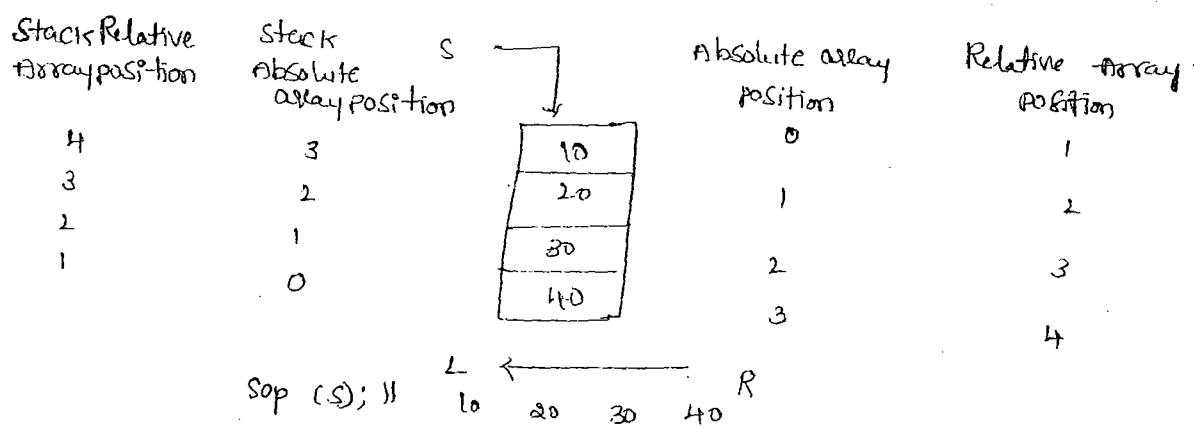
Stack is one of the subclass of vector & it is also treated as LIFO

LIFO class.

The basic working principle of stack is LIFO (the element which is inserted last, is that element will be taken out first)

Creating a stack is nothing but creating an object of stack class

The diagrammatic representation of stack is whose positions are given below.



#### Constructors:

(1) Stack()

#### Methods

(2) public boolean empty()

(3) public void push(Object)

(4) Public Object pop()

(5) public Object peek()

(6) public int search (Object)

Constructor (1) is used for creating an object of stack

Stack s = new Stack();

Method (2) returns true provided stack does not contain any elements

otherwise it returns false non empty.

Method (3) is used for inserting an element into the stack

Method (4) used for removing the top most element permanently from the stack.

Method (5) used for obtaining top most element.

→ Method (6) used for searching an element into the stack. If an element is found in the stack then we get stack relative array position (see the diagram) otherwise it returns "-1" which indicates element is not found.

→ W.A.J.P which illustrate the concept of stack

```
import java.util.*;  
class Stack  
{  
    public static void main (String [] args)  
    {  
        Stack s = new Stack();  
        System.out.println ("Size = " + s.size()); // 0  
        System.out.println ("Is s empty ? = " + s.empty()); // true  
        System.out.println ("Content = " + s); // []  
        s.push (new Integer (10));  
        s.push (new Integer (20));  
        s.push (new Integer (30));  
        s.push (new Integer (40));  
        System.out.println ("Size = " + s.size()); // 4  
        System.out.println ("Is s empty = " + s.empty()); // false  
        System.out.println ("Content = " + s); // [10, 20, 30, 40]  
        System.out.println ("Deleted element = " + s.pop()); // 40  
        System.out.println ("After removing Content = " + s); // [10, 20, 30]  
        System.out.println ("Top most element = " + s.peek()); // 30  
        System.out.println ("After peek element Content = " + s); // [10, 20, 30]  
        int pos = s.search (new Integer (10));  
        System.out.println ("Pos of 10 = " + pos); // 3
```

163

```
pos = s.search (new Integer(100));
System.out.println("pos of 100 = " + pos); // -1 indicates 100 not found
```

}

}

- (4) java.util.Dictionary: Dictionary is one of the predefined abstract class & whose object can't be created directly. so that it is not used directly in realtime applications

When we create an object of Dictionary class indirectly whose object organizes the data in the form of (key,value)

We know that in the (key,value) the value of key must be unique & the value of value may or may not be unique & both the values of key & value must be objects.

Since it is an abstract class which is playing an important background role in defining the methods & giving its sub classes.

#### Instance methods:

- (1) public int size()
- (2) public void put (Object, Object)
- (3) public Object remove (Object)
- (4) public void removeAll()
- (5) public Enumeration keys()
- (6) public Object get (Object)

→ Method (2), used for adding the data to the L2DCFW variable in the form of (key,value)

→ Method (3), is used for removing the entries of L2DCFW variable on the basis of content

→ Method (4), removes all the entries of L2DCFW variables

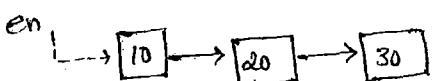
→ Method (5) is used for extracting the data from any Legacy 2DCFw

Variable in the form of keys & we pass these key values to the get method. So that we get set of values.

→ Method (G), used for obtaining set of values by taking key values.

Eg: `Sop (d); // {10=1.5}, {20=2.5}, {30=3.5} }`

Enumeration en = d.keys();



`while (en.hasMoreElements())`

{

Object kobj = en.nextElement();

Object vobj = d.get(kobj);

`Sop (vobj + " is the value of " + kobj);`

\*\*

## 5) java.util.HashTable:

→ HashTable is one of the predefined concrete sub class of Dictionary.

→ Since, it is a concrete class whose object can be created directly. So that we use this class in real time application.

→ An object of HashTable organizes the data in the form of (key,value) by following hashing mechanism.

→ We can't determine any o/p of the HashTable. bcoz it is following Hashing mechanism.

### Constructors:

(1) HashTable()

(2) HashTable (int size)

(3) HashTable (int size, float fillratio)  
load ratio

→ constructor (3) is used for creating an object of HashTable class by following a formula called pre-memory allocation.

→ The pre-memory allocation formula in HashTable

$$\text{size} = \text{size} + \lceil \text{size} * \text{fillratio} \rceil \quad \lceil \rceil \rightarrow \text{ceil function}$$

- Note:  $\lceil \rceil$  → represents ceiling which gives nearest highest integer value
- $\lfloor \rfloor$  → represents flooring gives nearest lowest integer value.
- > Set classpath = %classpath% ;
  - The no. of entries in Hashtable is equal to size of the array
  - then JVM will call automatically pre-memory allocation formula for restructuring the size of the Hashtable.
  - The range of fillratio or loadratio is 0.09 to 1.0 & the default value of fillratio is 0.75.
  - Inherit a Java program to implement the following by using hashtable

AP	Hyd
Karnataka	Banglore
TAMIL	CHENNAI
MH	MUM
⋮	⋮

- (a) Enter state & print its capital  
ex: Hyd is the Capital of AP
- (b) If state is not found then print "NO IDEA"
- (c) If user is not passing state then print "PLZ enter state name"

11 hashtable.java

```

import java.util.*;
class hashtable
{
    public static void main (String args[])
    {
        Hashtable ht = new Hashtable();
        ht.put ("AP", "Hyd");
        ht.put (new String ("KAR"), new String ("BANG"));
        ht.put ("TAMIL", "CHENNAI");
        ht.put ("MH", "MUM");
        Enumeration en = ht.keys();
        System.out.println ("=====");
        while (en.hasMoreElements ())
        {
    }
}
```

## Limitations of HashTable:

When we use Hash-table as a part of our application we are able to maintain user defined data in the form of key,value but we are unable to perform the following.

- (a) unable to read the data from properties file / resource bundle file.
  - (b) Unable to read Environmental Variable & their values (Env variable & are those which are having predefined meaning to the operating system & those values will be read by os at the time of booting  
Eg: classpath, path, os, CPU are Env variables.

## Def of properties file:

A properties file is one which is organizing the data in the form of (key,value). In industry properties file is also known as resource bundle file & they must be saved on some filename with an extension either .prop or .rbf.

Properties file always resides in secondary memory.

Ex: student.prop / Student.rbf

Sno=10
Sname= Sathya
Marks = 90.99

Here (Sno, Sname, Marks) are called keys  
(10, Sathya, 90.99) are called values.

To eliminate the limitations of Hash-table we use the concept of properties class.

Q: What are the differences between Hash-table & Hashmap?

### Hashtable

(1) No null values permitted for (Key, value)

(2) Hash-table is one of the synchronized class

(4) Null values are permitted once for (Key, value)

(2) Hashmap is one of the non synchronized class.

### java.util.properties:

→ properties is one of the pre-defined sub class of Hash-table.

→ Using creating a properties object is nothing but creating an object of properties file.

Ex: properties p = new properties();

Here p organizes the data in the form of (Key, Value)

### Advantages of properties class over Hash-table:

(1) It is able to maintain userdefined data in the form of (key,value) in the main memory of the computer.

- 2) Using properties class we can read the data from properties file/resource bundle file.
- 3) We can read the data from Env. variables or system properties.
- Finally using properties class we are able to develop 'flexible java applications'.

Constructor:

- 1) properties()

Instance Methods:

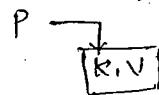
- 2) public void load (FileInputStream)

- 3) public String getProperty (String)

Constructor (1) is used for creating an obj of properties class.

Ex:

properties p = new properties();

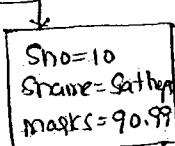


'p' is available in heap memory

Method (2) is used for loading the content of properties file in properties class obj by opening properties file in read mode.

Ex: FileInputStream fis = new FileInputStream ("student.prop");

p.load (fis);



Sno, Sname, marks are known as keys & properties name.

10, Sathya, 90.99 are known as values & properties values.

Method (3) is used for obtaining a property value by passing property name.

String sno = p.getProperty ("sno");

↳ p.get (sno); // 10

String sname = p.getProperty ("sname");

↳ p.get (sname); // Sathya

String marks = p.getProperty ("marks");

↳ p.get (marks);

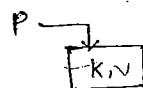
→ Steps for developing flexible java application by using properties class:

(1) Create a properties file & save it on some filename with an extension either .prop or .rbf.

Properties file is one of the text file & it always resides in secondary memory.

student.prop

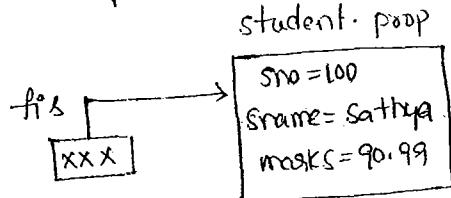
Sno=100
Sname=Sathyu
marks=90.99



(2) Create an obj of properties class  
properties p = new properties();

(3) Open the properties file in read mode with the help of FileInputStream by passing the properties filename.

Ex: FileInputStream fis = new FileInputStream ("student.prop");

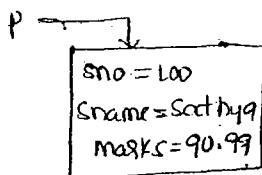


fis  
null

(In the case of properties file does not exist)

(4) Load the content of properties file in properties class object from secondary memory to primary memory.

Ex: p.load (fis);



(5) Obtain property values by passing property names from properties class object

Ex: System.out.println ("Student Sno = " + p.getProperty ("sno"));

System.out.println ("Student Sname = " + p.getProperty ("sname"));

System.out.println ("Student marks = " + p.getProperty ("marks"));

(6) Close the properties file which was opened in read mode.

Ex: fis.close();

→ 11 prop1.java

```

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

```

class prop1
{
    public static void main (String[] args)
    {
        try
        {
            // 1-- we create the prop file
            // 2
            Properties p = new Properties();
            // 3
            String fname = args[0];
            FileInputStream fis = new FileInputStream (fname);
            // 4
            p.load (fis);
            // 5
            System.out.println ("student sno = " + p.getProperty ("sno"));
            System.out.println ("student sname = " + p.getProperty ("sname"));
            System.out.println ("student marks = " + p.getProperty ("marks"));
            // 6
            fis.close();
        }
        catch (FileNotFoundException fe)
        {
            System.out.println ("prop file does not exist... ");
        }
        catch (Exception e)
        {
            System.out.println (e);
        }
    }
}

```

}/main

}/ class

→ Write a java program which will print or read environment variables of the system

import java.util.\*;

class prop2

```

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

```

Properties p = System.getProperties();
Enumeration en = p.keys();
while (en.hasMoreElements())
{
    Object kobj = en.nextElement();
    Object vobj = p.get(kobj);
    System.out.println(kobj + " ---> " + vobj);
}
}

```

In the above program `getProperties()` method is one of the static methods present in `System` class & it returns all the environmental variables of our system in the form of properties class object

`System` → `public static Properties getProperties()`

### java.util.Scanner:

It is one of the predefined class added as a part of Java 1.5 to read the data from keyboard.

#### Constructor:

(1) `Scanner(InputStream)`

#### Instance Methods:

- (2) `public byte nextByte()`
- (3) `public short nextShort()`
- (4) `public int nextInt()`
- (5) `public long nextLong()`
- (6) `public float nextFloat()`
- (7) `public char nextChar()`
- (8) `public boolean nextBoolean()`
- (9) `public double nextDouble()`
- \* (10) `public String nextLine()`

→ `public XXX nextXXX()`  
`XXX` → Any fundamental data type

constructor (1) is used for creating an obj of Scanner which allows us to read the data from the key board.

Ex: Scanner s = new Scanner(System.in);

Here 'in' is an obj of Input stream class created in System class as a static datamember & it gives control of the key board at the time of running the application.

Method(2) to (9) are used for reading fundamental values from keyboard.

Method (10), used for reading any kind of data in the form of String.

→ Write a JP which will compute sum of two numbers by accepting the data from the keyboard.

```
import java.util.*;
class DataRead
{
    public static void main (String[] args)
    {
        Scanner s = new Scanner (System.in);
        System.out.println ("Enter first no:");
        String s1 = s.nextLine();
        System.out.println ("Enter second no:");
        String s2 = s.nextLine();
        int x = Integer.parseInt(s1);
        int y = Integer.parseInt(s2);
        System.out.println ("Sum = " + (x+y));
    }
}
```

## Files (or) Stream Handling

- (1) Java program →   
 → volatile prog / non-persistence ⇒ result ⇒ stored in mm ⇒ temp  
 → non-volatile prog ⇒ result stored in SM ⇒ permanent

In the context of files, a Java programmer can develop two types of applications. They are

- (1) Volatile / Non-persistence programming
- (2) Non-volatile / persistence programming

A volatile program is one whose results stored temporarily in Main memory of the computer.

Eg: All the programs we have seen so far also comes under volatile.

A Non-volatile program is one whose result stored in secondary storage devices permanently.

We know that data can be stored permanently by using two approaches. They are

- (1) Through the concept of files
- (2) Through the concept of DataBase

If we store the data permanently in the form of files then the data of the file can be manipulated by any unauthorized users. & more over Hence the concept of files is unable to provide enough security to prevent the confidential information from unauthorized users & moreover they provide less security. Hence Industry is not using files concept for storing data permanently but it is using popular DB like oracle, DB2, SQL Server, MySQL.. etc..

In order to deal with the file programming we use various predefined classes & interfaces which are present in `java.io.*` package

Def of file: Collection of records is known as file.

Ex:

Student.data			file name
10	abc	99.99	← lec1
20	PQR	80.86	← lec2
30	abc	90.99	← lec3

Each and every permanent data must have a filename in the secondary memory for future identification purpose.

→ files are always residing in secondary memory (ex: hard disk)

→ collection of field values known as a record.

Def of Stream: The flow of data bytes/bits b/w primary memory and secondary memory is known as "Stream".

Operation on files:

→ On file we perform two types of operations. They are

- (1) Write operation / OutputStream operation
- (2) Read operation / InputStream operation.

(1) Write operation: Write operation is used for transforming the data from primary memory to secondary memory.

In order to perform write operation, the java programmer has to perform the following sequence of steps.

- (1) choose the filename
- (2) open the file in write mode
- (3) perform cycle of write operation.

While we are performing the write operation on the file, we get the following consequences.

- (a) Trying to write the data on files when the secondary memory is full.
- (b) Trying to write the data to read only files.

Because of the above two consequences, we get a predefined exception called `java.io.IOException`.

- (2) Read operation: Read operation is used for reading the data from secondary memory to primary memory.

To perform the read operation on files concept, as a Java programmer we perform the following sequence of operations.

- (1) → choose the filename
- (2) → open the file in read mode
- (3) → perform cycle of read operations.

While we performing read operation on the files we may get the following consequences.

- (a) Trying to read the data from the file which is not existing in secondary memory. Bcoz of this consequence we get a predefined exception called `java.io.FileNotFoundException`.
- (b) We are trying to read the data from corrupted file i.e. the file name is existing in secondary memory but the data of the file is not in proper state & corrupted.

Bcoz of these consequence, we get a predefined exception called `java.io.IOException`.

In file programming we get `java.io.FileNotFoundException` & `java.io.IOException` & more over these two exceptions are known as checked exceptions.

`java.io.IOException` is one of the super class for `FileNotFoundException`.

### Types of streams in Java:

Based on Transferring the data b/w primary memory & secondary memory. Streams are classified into two types. They are

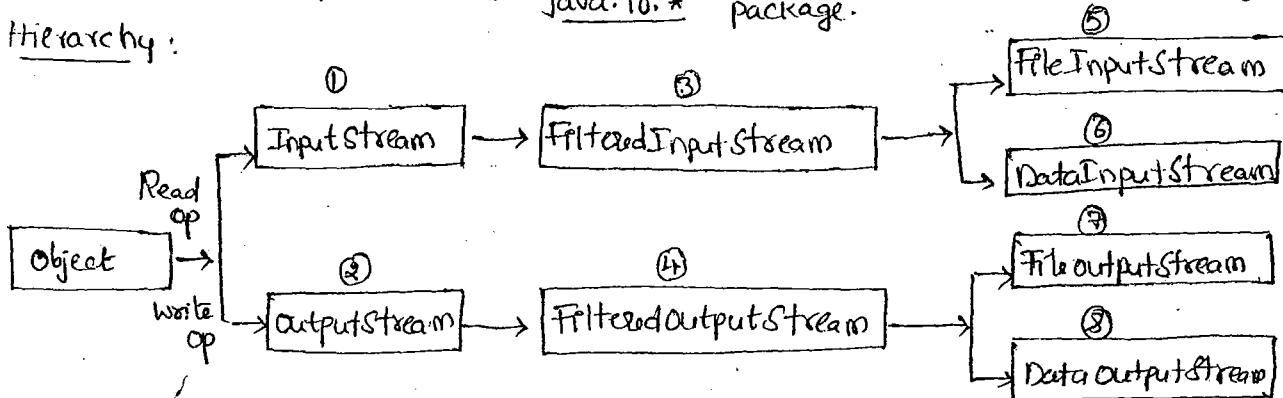
- (1) Byte streams
- (2) character streams

(1) Byte streams: Byte streams will transfer the data b/w primary memory & secondary memory with the approach of byte by byte. In other words the process of transferring one byte at a time b/w primary memory and secondary memory is known as "Byte streams".

Byte streams are not that much effective compare to character streams.

Byte streams contains two categories of classes. Some category of classes are used for performing write operation & some other category of classes are used for performing 'read' operation. These two category of classes are present in java.io.\* package.

Hierarchy:



In the above hierarchy chart the classes ①, ②, ③ & ④ represents predefined abstract classes & whose objects can't be created directly. So that we never use them directly in file.

Applications & these classes are playing a very important role in giving the methods to the concrete classes.

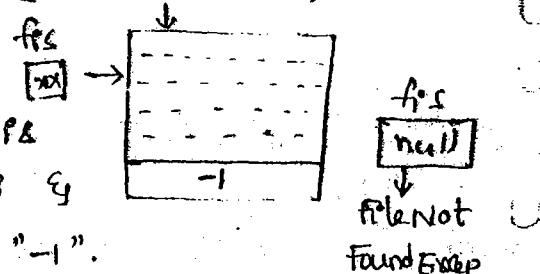
The classes ⑤, ⑥, ⑦ & ⑧ are the bottom most concrete subclasses of all byte stream category classes & hence whose objects can be created directly so that we can use them in file app development.

Inputfilestream:

It is one of the bottom most concrete subcls of all byte stream cls. The purpose of the class is to open a file in read mode. In other words opening a file read mode nothing but creating an obj of FileInputStream etc.

FileInputStream fis = new FileInputStream ("student. data");

- > If the filename student. data is opened successfully then the address of the open file is placed in an obj of FileInputStream class & end of the file in java is indicated by "-1".
- > If the filenot opened successfully in read mode then the obj of FileInputStream contains "null" & we get an exception called filenotfound



fis  
null  
↓  
FileNotFoundException

## Exception:

### Constructor:

(1) `FileInputStream (String name)` throws `FileNotFoundException, IOException`.

### Instance methods:

(2) `public byte readByte()`

(6) `public double readDouble()`

(3) `public short readShort()`

(7) `public char read(char)`

(4) `public int readInt()`

(8) `public boolean readBoolean()`

(5) `public long readLong()`

(9) `public void close()`

(10) `public int read()`,

using to read the data until we go to the end of file.

Constructor (1) is used for creating an obj of `FileInputStream` by specifying the name of the file which is nothing but open in the specified file in read mode.

Methods (1) to (8) are used for reading fundamental data from the file.

As soon we start reading the data from the file, an obj of `FileInputStream` automatically gets advanced.

### FileOutputStream:

This is one of the bottom most concrete subcls of all `InputStream` classes.

The basic purpose of this cls is to open the file in write mode. In other words opening a filename in write mode is nothing but creating an obj of `FileOutputStream` cls.

### Constructors:

(1) `FileOutputStream (String name)`

} throws `IOException`.

(2) `FileOutputStream (String name; boolean flag)`

The above constructors are used for opening a file in write mode with the following flags.

Case (1): If filename is not existing & flag = true || false then

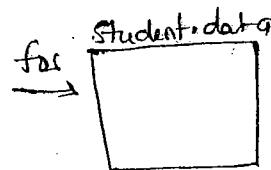
ans: specified file is opened in write mode ready.

Eg: `FileOutputStream fos = new FileOutputStream ("student.data");`

(Q9)

`FileOutputStream fos = new FileOutputStream("student-data", false);`  
(ex)

`FileOutputStream fos = new FileOutputStream("student-data", true);`



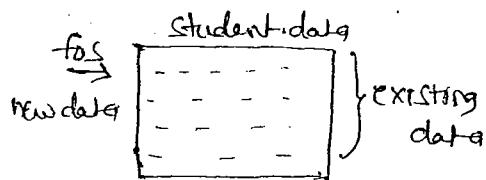
Case(2): If filename existing & flag = false then

Ans: new Data overlaps existing data.

Ex: `FileOutputStream fos = new FileOutputStream("student-data");`

(ex)

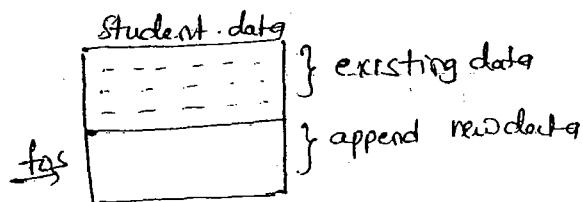
`FileOutputStream fos = new FileOutputStream("student-data", false);`



Case(3): If filename is existing & flag true then

Ans: new data append to the existing data.

Ex: `FileOutputStream fos = new FileOutputStream("student-data", true);`



### Instance methods:

- |  |  |
|--|--|
| 3) <code>public void write(byte)</code>  | (10) <code>public void write(boolean)</code> |
| 4) <code>public void write(short)</code> | (11) " " " " (string)                        |
| 5) <code>public void " (long);</code>    | (12) <code>public void close()</code>        |
| 6) " " " (float)                         |  |
| 7) " " " (int)                           |  |
| 8) " " " (double)                        |  |
| 9) " " " (char)                          |  |

- ↪ Method (3) to (10) are used for writing fundamental data to the file.
- ↪ Method (11) is used for closing the file which was opened in write mode
- ⇒ Write a Java program to do the following
  - (a) Create a file in write mode & write 1 to 10 numbers
  - (b) open a file in read mode & which contains 1 to 10 numbers & read and print.

(a) [1] prog-(9)

```

import java.io.*;
import java.util.*;
class fwrite
{
    public static void main (String [] args)
    {
        try
        {
            Scanner s=new Scanner (System.in);
            System.out.print ("Enter the file name");
            String frame = s.nextLine();
            FileOutputStream fos= new FileOutputStream (frame, true);
            for (int i=1; i<=10; i++)
            {
                fos.write (i);
            }
            System.out.print ("Data written to the file");
            fos.close();
        }
        catch (IOException ioe)
        {
            System.out.print ("problem in open a file in write mode");
        }
        catch (Exception e)
        {
            System.out.print (e);
        }
    }
}
  
```

11 prog - i)

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

class fread
{
    public static void main(String[] args)
    {
        try
        {
            Scanner s = new Scanner(System.in);
            System.out.println("Enter the file name to read");
            String frame = s.nextLine();
            FileInputStream fis = new FileInputStream(frame);
            int p;
            while((p = fis.read()) != -1)
            {
                System.out.print(p);
            }
            fis.close();
        }
        catch(FileNotFoundException fe)
        {
            System.out.println("file does not exist..");
        }
        catch(IOException ioe)
        {
            System.out.println("unable to read from the corrupted file..");
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

11 class.

⇒ write a java prog which will copy the content of one file into another file.

(or)

Implement Copy cmd of Dos - Os

↪ htns copy cmd is used for transferring the content of one file into another file.

⇒ Ex: copy fname.java xyz.java.

fname.java is called source file & it must be opened in read mode.

xyz.java is called destination file / target file must be opened in write mode.

For copy cmd less than 2 & more than 2 if any body pass then it is invalid syntax.

```
import java.io.*;
class Sathyacopy
{
```

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

```
        if (args.length != 2)
```

```
        {
            System.out.println ("Invalid arguments | syntax");
        }
```

```
    else
```

```
        {
            String sfile = args [0];

```

```
            String dfile = args [1];

```

```
            try
```

```
            {
                FileInputStream fis = new FileInputStream (sfile);

```

```
                FileOutputStream fos = new FileOutputStream (dfile, true);

```

```
                int k;

```

```
                do

```

```
                {
                    k = fis.read();

```

```
                    char ch = (char) k;

```

```
                    fos.write(ch);

```

```
                } (while k != -1);

```

```
                System.out.println (sfile + " content copied into " + dfile);

```

```
                fis.close();

```

```
            } fos.close();

```

```

    catch(FileNotFoundException fe)
    {
        s.e.p("file "+file+" does not exist..");
    }
    catch(IOException ioe)
    {
        s.e.p("Unable to open a file in write mode..");
    }
    catch(Exception e)
    {
        s.e.p(e);
    }
}
else
{
    main
}

```

### YCLASS

⇒ WAP to display the content of given file.

(or)

Implement 'Type' cmd of DOS.

```

import java.io.*;
class Satyacopy
{
    p.s.v.main(String[] args)
    {
        if(args.length != 1)
        {
            s.o.p("Invalid arguments | Syntax");
        }
        else
        {
            String file = args[0];
            try
            {
                FileInputStream fis = new FileInputStream(file);
                int k;
                while(k = fis.read() != -1)
                {
                    char ch = (char)k;

```

```

        - s.o.p (cch);
    }
    fis.close();
}

Catch (FileNotFoundException fe)
{
    s.e.p (sfile + " does not exist ...");
}

Catch (IOException e)
{
    s.e.p ("unable to read from corrupted files ...");
}

Catch (Exception e)
{
    s.e.p (e);
}

} else
} main
} class.

```

### DataInputStream :

This is one of the bottom most concrete subclass of all `InputStream` classes.

Purpose: `DataInputStream` class is used for achieving the following.

- read the data from keyboard (local reading)
- read the data b/w multiple machines which are located either in same n/w or in diff n/w (remote reading)

### Constructors:

(1) `DataInputStream(InputStream)` throws `FileNotFoundException`

Eg: `DataInputStream d1s = new DataInputStream(System.in);`



local reading

(2) public byte `readByte()`

- (3) public short readShort()
- (4) public int readInt()
- (5) public float readFloat()
- (6) " long readLong()
- (7) " double readDouble()
- (8) " char readChar()
- (9) " boolean readBoolean()
- (10) " String readLine()

Constructor (1) is used for reading the data either from keyboard or read from multiple location which are located either in same n/w or in different n/w by creating an obj of DataInputStream class.

Methods (2) to (9) are used for reading fundamental data (in both the local reading & remote reading).

Method (10) used for reading any kind of data in the form of string in both local reading & remote reading.

→ W.A.P to generate a multiplication table by accepting a number from keyboard

```
import java.io.*;
class DataRead
{
    public static void main(String[] args)
    {
        DataInputStream dis = new DataInputStream(System.in);
        System.out.print("Enter a no:");
        String s = dis.readLine();
        int n = Integer.parseInt(s);
        for (int i=1; i<=10; i++)
            System.out.println(n + " * " + i + " = " + (n*i));
    }
}
```

DataOutputStream: It is one of the bottom most concrete subclass of all OutputStream classes

- Purpose: The purpose of this class is to write the data b/w multiple machines which are located either in same NW or in different NW.

(Known as Remote writing).

- Constructors:

(1) DataOutputStream (outputstream) throws IOException.

(2) Instance methods:

(a) public void writeByte (byte)

(b) public void writeShort (short)

(c) " " " writeInt (int)

(d) " " " writeFloat (float)

(e) " " " ~~long~~ writeLong (long)

(f) " " " writeChar (char)

(g) " " " writeDouble (double)

(h) " " " writeBoolean (boolean)

(i) " " " writeLine (String)

public void writeXXX (xxx)

      ^  
public void write (xxx)

    → Array of FDT

Constructor (1) is used for creating an obj of DataOutputStream for performing remote writing.

Method (a) to (9) & (11) to (18) are used for writing fundamental data b/w multiple machines.

Methods (10) & (19) are used for writing string data b/w multiple machines.

## SERIALIZATION AND DESERIALIZATION

In Java programming the result of Every Java program is available in MM. In other words the data of the Java program is available in the form of objects.

If an obj is containing multiple values & if we want to transfer those value from primary memory to SM then we need to perform multiple write operations. which is the time consuming process. To eliminate this time consuming process we must use the concept of serialization.

The basic advantage of serialization concept is to perform single write operation irrespective of no. of values available in an object

### Definition of Serialization:

The process of saving the state of the object at a time to the file in SM from PM is known as serialization.

(or)

The ability to store the entire content of the object at the time to the file of SM. Some primary memory is known as serialization.

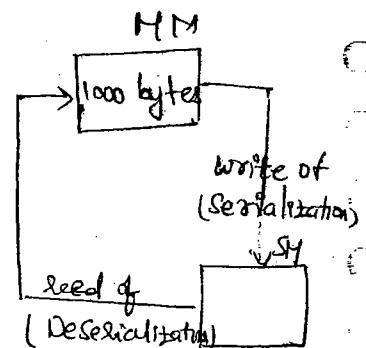
Serialization concept always participates in write operation.

We know that a file is a collection of records. A record is a collection of field values.

To read one record from a file of secondary memory to the PM we need to perform multiple read operations, which is the time consuming process.

To eliminate this time consuming process, we get use the concept of deserialization.

The advantage of deserialization concept is to eliminate multiple read operations & it will perform all only single read operation to read the entire record irrespective of no. of field values in the record.



## Def of deserialization:

The process of obtaining the entire record of a file from SM to PM is known as deserialization.  
(or)

The ability to store the entire content of a record of a file from SM to PM is known as deserialization.

Deserialization concept always participates in read operation.

If we want to deal with serialization & deserialization concepts then we need to develop the three categories of apps. They are.

- (1) development of serializable sub class program.
- (2) development of serialization process program.
- (3) " " deserialization " "

### (1) Development of Serializable subclass:

Whenever class objects want to participate in serialization process that class definition must implements a predefined interface called java.io.Serializable.

Whenever class is implementing java.io.Serializable, the class is known as Serializable sub class.

The interface java.io.Serializable is known as marker / Taged interface bcoz it is not containing any methods.

### Def of Marker Interface:

A marker interface is one which is not containing any methods.

e.g.: java.io.Serializable

java.lang.Clipable

javax.rmi.Remote

java.util.ServiceSingleThreadMode

java.ejb.EJB

In order to develop any Serializable sub class, the java programmer must follow the following sequence of steps.

- (1) choose the appropriate package & ensure the package statement must be first executable statement & this package is containing Serializable sub class - for common accessibility.
- (2) choose an appropriate user defined class & ensure whose modifier must be public & this class is going to be known as Serializable sub class
- (3) The class which is selected in step no(2) must implement a predefined interface called java.io.Serializable.
- (4) Every Serializable sub class must contain set of data members
- (5) Define for each & every datamember a set of setXXX() methods.
- (6) Define set of getXXX() methods for each and every data member of a class.
- (7) whichever class belongs to the package the class name must be given as a filename with an extension .java.

e.g.: Develop a Serializable sub class for Student

```

    M Student.java ... ⑦
    package sp; ①
    public class Student implements java.io.Serializable
    {
        int stno;
        String name; } ⑫
        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;
}

```

} student

→ Compile the above program

> javac -d . student.java <

Ensure student.class file must be generated

SP

student.class.

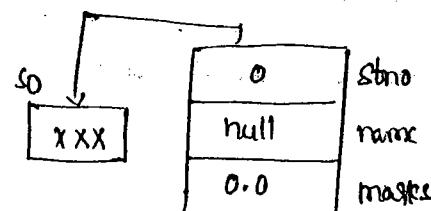
Note: If any class is containing set of setXXX() methods & set of getXXX() methods then that class is known as Java Bean class or pojo(plain Old Java Object) class this style of writing is known as component based style programming.

## (2) Development of serialization process program:

In order to develop serialization process program the java program must follow the following sequence of steps:

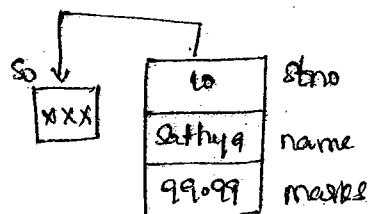
(i) Create an obj of serialization sub class.

Eg: sp.Student s0=new sp.Student();

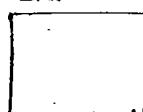


(ii) Call set of setXXX() methods for setting the dynamic values to the data members of serialization sub class.

e.g.:  
so.setSno(10);  
so.setName("Sathya");  
so.setMarks(99.99);



(iii) choose the filename & open it into writemode with the help of FileOutputStream class.

FileOutputStream fos = new FileOutputStream("student.data"); for 

N) An obj of FileOutputStream can't write the entire content of the object to write the entire content of the obj we create an obj of ObjectOutputStream class with the help of the following constructions

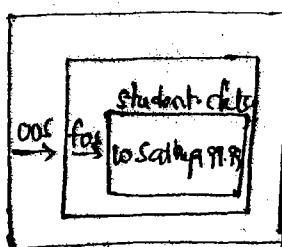
ObjectOutputStream  
a. ObjectOutputStream(FileOutputStream)  
b. public void writeObject(Object)

ObjectOutputStream oos = new ObjectOutputStream(fos);

Here oos & fos are known as Chained / Sequence streams  
(if one stream obj is pointing to another stream obj & both the streams must belongs to same type then those obj are known as chained / sequence streams

(iv) Write the entire serializable subclass obj data at a time to the file.

oos.writeObject(so);



(v) close the file which was opened in write mode

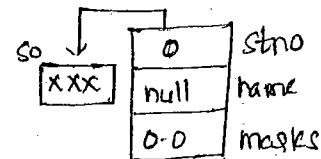
e.g.: fos.close();

oos.close();

### ③ Development of Deserialization process:

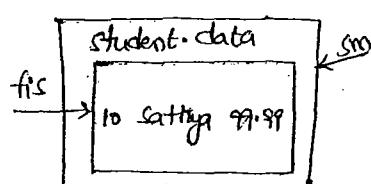
Step ①: Create an obj of Serializable sub class & it is used for holding deserialized data.

Eg: `sp.student s0 = new sp.Student();`



Step ②: choose the file name & open it into read mode with the help of FileInputStream.

Eg: `FileInputStream fis = new FileInputStream("Student.data");`

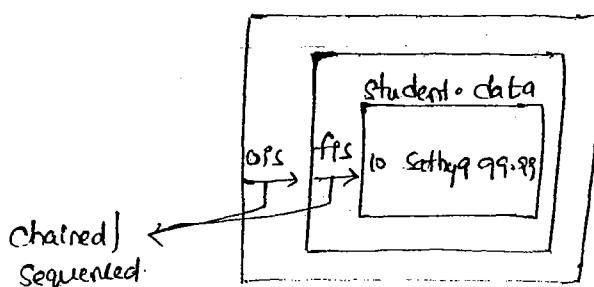


Step ③: An obj of FileInputStream may not be able to read all the values of a record. To read all the values of a record, we must create an obj of ObjectInputStream by using the following constructor

`ObjectInputStream` →

`ObjectInputStream(FileInputStream)`

Eg: `ObjectInputStream ois = new ObjectInputStream(fis);`



Step ④: To read the entire record from a file of secondary memory into the primary memory we use the following method.

`ObjectInputStream` →

`Public Object readObject()`

Step ⑤: When the Serializable sub class data is Serialized internally Serialized data is treated as obj of `java.lang.Object`

Eg: Object obj = ois.readObject();

↓

10 | Sathya | 99.99 ← deserialized data.

Step④: Typecast an obj of java.lang.Object into Serializable subclass object. bcoz set of getXXX() methods are not present in object of java.lang.Object

Eg: so=(Student) obj;

Step⑤: Apply set of getXXX() methods to retrieve the deserialized data

Eg: System.out.println("Student No:" + so.getStno()); // 10  
System.out.println("Student Name:" + so.getName()); // Sathya  
System.out.println("Marks :" + so.getMarks()); // 99.99.

Step⑥: Close the files which was opened in 'read' mode & or terminate the stream objects which are pointing to the file.

Eg: fis.close();  
ois.close();

→ Write a Java program which illustrate the concept of serialization process.

```
// Serp.java
import sp.student;
import java.io.*;
import java.util.*;
class Serp
{
    public static void main(String[] args)
    {
        try
        {
            // Step 1
            Student so = new Student();
        }
    }
}
```

|| Step 1

Student so = new Student();

|| Step 2 (accepting the values dynamically)

Scanner sc = new Scanner(System.in);

```

        System.out.println("Enter student number");
        int sno = Integer.parseInt(sc.nextLine());
        System.out.println("Enter student name");
        String name = sc.nextLine();
        System.out.println("Enter marks");
        float marks = Float.parseFloat(sc.nextLine());
        so.setSno(sno);
        so.setName(name);
        so.setMarks(marks);

    // Step ③
    System.out.println("Enter the filename to write student data");
    String fname = sc.nextLine();
    FileOutputStream fos = new FileOutputStream(fname);

    // Step ④
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(so);

    System.out.println("Student data written successfully to the file...");

    // Step ⑤
    fos.close();
    oos.close();
}

catch (IOException ie)
{
    System.out.println("unable to open file in write mode");
}

catch (Exception e)
{
    System.out.println(e);
}

// main
public class

```

→ Write a Java program which illustrate the concept of deserialization process

1) deSerp.java

```
import sp.student;
import java.io.*;
import java.util.*;

class deSerp
{
    public static void main(String[] args)
    {
        try
        {
            // Step 1
            Student so = new Student();
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the filename to read the data");
            String fname = sc.nextLine();
            FileInputStream fis = new FileInputStream(fname);

            ObjectInputStream ois = new ObjectInputStream(fis);

            Object obj = ois.readObject();
            so = (Student) obj; // OTC

            // Step 6
            System.out.println("Student number = " + so.getStno());
            System.out.println("Student name = " + so.getName());
            System.out.println("Student marks = " + so.getMarks());

            // Step 7
            fis.close();
            ois.close();
        }
    }
}
```

```

    catch (FileNotFoundException fe)
    {
        System.out.println("file does not exists..."); 
    }
    catch (IOException ie)
    {
        System.out.println("Unable to read from the corrupted file..."); 
    }
    catch (Exception e)
    {
        System.out.println(e); 
    }
}

```

if main

if class

### Character streams:

Character streams are those which will transfer char at a time b/w primary memory and secondary memory.

→ Character streams are more effective than byte streams.

→ Character streams are containing two categories of classes. They are

(1) Some of the classes are performing write operation.

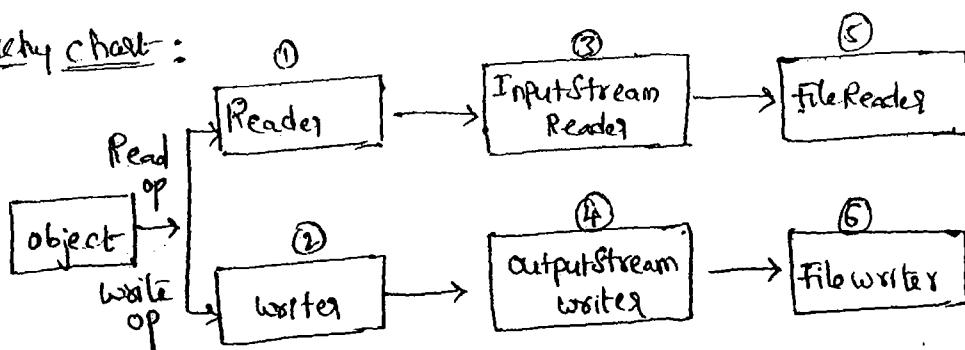
(2) Some of the classes are performing read operation.

→ Both these classes belongs java.io.\* package.

→ Finally java.io.\* is containing some of the classes for byte stream for transferring byte by byte b/w primary memory & secondary memory.

3 Some of the classes for character streams for transferring char at a time b/w primary memory & secondary memory.

### Hierarchy chart:



The classes ①, ②, ③ & ④ are abstract classes.

So, that whose objects can't create directly & we never use them in real time applications.

- The classes ⑤ & ⑥ are bottom most character concrete sub classes of all character stream classes since these classes are concrete we may create these class objects directly in our application.
- The functionality of FileReader is to open a file in Read mode & the functionality of file writer is to open a file in write mode.
- In other words the functionality of FileReader is exactly similar to fileInputStream but the difference is an obj of FileReader reads bytes at a time where as an obj of fileInputStream reads the one byte at a time from the file.
- The functionality of filewriter is similar to fileOutputStream but the difference is an obj of filewriters writes bytes at a time and an obj of fileOutputStream writes one byte at a time to the file.

## MULTITHREADING

Multi-threading is one of the distinct facility of Java programming.

- "A flow of control" is known as thread.
- The basic usage/purpose of thread is to execute user/programmer defined methods.
- If a Java program is containing multiple flow of controls then that Java program is known as multithreaded.
- The basic aim of multithreading is to achieve concurrent execution.
- The languages like C, C++, PASCAL, COBOL etc are treated as single threaded modeling languages. Bcoz their runtime execution environment contains only one flow of control.
- By using single threaded modeling languages we can achieve only sequential execution but not concurrent execution & more over these languages does not contain any pre-defined library for development of multi-threading application.  
i.e. multithreading is not a predefined concept of single threaded modeling language.

- The languages like Java and .Net are treated as multi-threaded modeling languages. bcoz their run-time execution environment contains multiple flow of controls & provides a facility to the programmer to create multiple flow of controls.
- By using multi-threaded modeling language, we can achieve both sequential & concurrent execution.
- All multi-threaded modeling languages contains an effective API for development of multi-threading applications.
- In Java programming we have the following API for developing multithreaded applications

- (a) `java.lang.Thread` (class)
- (b) `java.lang.Runnable` (interface)

→ The real-time implementation of multi-threading is that to develop real world server SW like Tomcat, web logic, web sphere, glassfish etc.

→ Whenever we create a Java program by default there exist two types of threads. They are:

- (a) Foreground / child thread
- (b) Background / parent thread

→ A foreground thread is one which always executes the logic of the Java program or user/programmer defined methods.

→ A background thread is one which is always monitoring the execution status of foreground thread(s).

→ By default, a Java programming environment is containing single foreground thread & single background thread. programmatically a Java environment contains multiple foreground threads and recommend to have single background thread for Java program.

→ Multi-threading is one of the specialized form of multicasting concept of ols.

Q) How do you justify "Every Java program is multithreaded"?

Ans: Whenever we execute the Java program the logic of the Java program is executed by one of the thread known as Foreground thread. To monitor the execution status of the foreground thread, one more thread is created internally which is known as Background thread so the Java execution environment containing multiple flow of controls Hence every Java program is multithreaded.

Q) What are the differences b/w program & process?

### Program

### Process

- (1) set of optimized instructions is known as program.
- (1) A program is under execution is known as process.
- (2) programs always resides in secondary memory (Hard disk).
- (2) processes are always resides in primary memory.
- (3) programs exist for a long time in secondary memory until we delete.
- (3) process will exist for a limited span of time in primary memory.
- (4) In programming point of view every program is one of the static object.
- (4) In programming point of view every process is treated as a dynamic object.

### Process based Application & Thread based applications:

In the software industry a programmes can develop two types of Applications They are

1. process based applications &
2. Thread based applications.

#### process based application

(1) process based app are those whose execution environment contains single flow of control.

(2) All C, C++, PASCAL, COBOL etc apps are comes under process based apps.

(3) Context switch is more in process based apps so that execution time of these apps will be more.

(4) For each & every subprogram there exist a separate address based space.

#### Thread based applications

(1) Thread based app are those whose execution environment contains multiple flows of controls.

(2) All Java & .Net apps are comes under thread based apps.

(3) Context switch is less in thread based app & whose execution time is very less.

(4) Irrespective of no. of subprograms there exist a single address space.

### Process based Apps

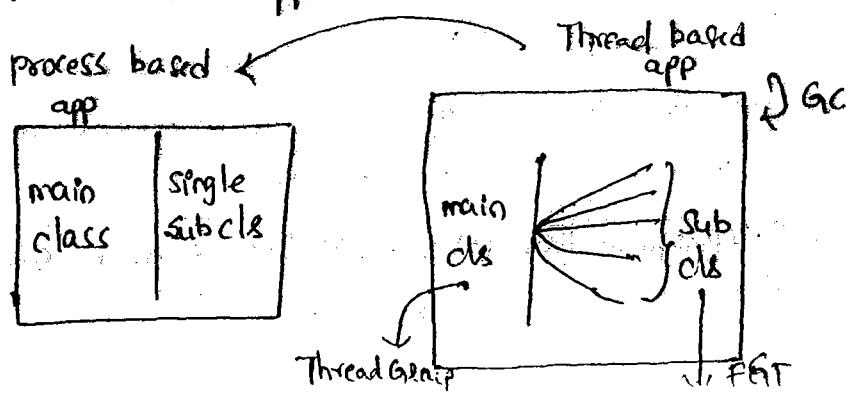
- 5) All process based apps are treated as heavy weight components.
- 6) With process based we are able to achieve only sequential execution but not concurrent execution.

### Thread based Apps

- 5) All thread based apps are treated as light weight components.
- 6) With thread based apps we can achieve both sequential & concurrent execution.

#### Note :

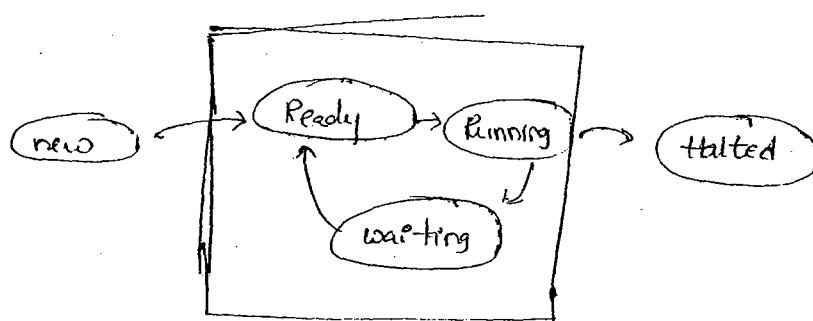
1. The mechanism of switching the control of CPU from one address space to another address space is known as Context Switch.
  2. For the best apps context switch must be less.
  3. The amount of temporary memory space created by OS on stack memory for temporary execution of the method is known as "address space".
  4. Heavy weight components are those whose processing time is more.
  5. Light weight Components are those whose processing time is very less.
- When ever we execute a thread based application, internally it is a process based application. In which one main process is created by the JVM & the main process is internally creating various sub processes. The sub processes are known as foreground threads & the main process is known as Thread Group. Hence each and every thread based app is one of the process based app but a process based app is not a thread based app.



## States (or) Life cycle of a thread :

- When ever we develop any multithreading app there is a possibility of having multithreads. When the multiple threads are under execution. They will undergo various states of thread in multithreading. States of the thread are classified into 5 types. They are
1. new state
  2. ready state
  3. running state
  4. waiting state and
  5. Halted state

## State chart diagram:



A new state is one in which thread is created and is about to enter into the main memory.

### Ready state :

A ready state is one in which thread is entered into the main memory, address space is allocated for the method of first time waiting for the CPU.

Running state: A running state is one in which the thread is under the control of CPU or the thread is under execution.

\* Waiting state:

The thread is said to be in waiting state if and only if it satisfies any of the following factors.

(a) Thread is coming to waiting state for the remaining CPU burst time (it is an amount of time required by the thread from the CPU for its complete execution).

(b) Thread is coming to the waiting state when it is suspended from its current execution.

(c) Thread is coming to the waiting state when the thread makes wait by specifying amount of waiting time in terms of msec  
(1 sec = 1000 msec)

(d) Thread is coming to waiting state when the thread is made wait without specifying any time.

(e) Thread is coming to waiting state when the thread is currently executing thread decided to sleep by specifying amount of sleeping time in terms of msec.

(f) Thread is coming to waiting state when the completed threads are joined after their completion completion of the execution.

Halted state: Halted state is one in which the thread is completed its total execution.

∴ As long as the thread is in ready, running & waiting states whose execution status is true whereas when the thread is in new & halted states, whose execution status is false.

The states ready, running & waiting are known as in memory states & the states new & halted are known as out memory states.

### No. of ways to create a thread :

In Java programming to create a thread we have two approaches. They are

- (1) By using `java.lang.Thread` class and
- (2) By using `java.lang.Runnable` interface.
- (3) By using `java.lang.Thread` class:

By using `Thread` class we can create a flow control(thread) in 3 ways. They are

- (a) directly  
Ex: `Thread t1 = new Thread();`
- (b) by using `factory()` method  
Ex: `Thread t2 = Thread.currentThread();`
- (c) An obj of sub class of `Thread` class is nothing but an object of `java.lang.Thread` class.

Ex: class `th` extends `Thread`

```

    {
    }
}
```

`Th t3 = new Th();` // Here `t3` is an obj of 'th'.

'`th`' is a sub class of `Thread` class & finally '`t3`' is an indirect obj of `Thread` class.

### Thread class profile:

#### Data members:

- (1) `public static final int MAX_PRIORITY (=10)`
- (2) `public static final int MIN_PRIORITY (=1)`
- (3) `public static final int NORM_PRIORITY (=5)`

The above data members are known as Thread priority modifiers & they must be access with its class name. The default priority of thread is NORM\_PRIORITY.

### Constructors:

(1) Thread(): This constructor is used for creating an obj of Thread class without specifying name of the thread.

Ex: Thread t1 = new Thread();

When we create 'n' no. of thread class obj & using this constructor, by default we get the names of the thread as thread-0, thread-1, ..., thread-(n-1).

(2) Thread(string):

This constructor is used for creating an obj of Thread class by specifying the name of the Thread.

Ex: Thread t1 = new Thread("javath");  
            ↑  ↑  
            ref  thread name.

(3) Thread(Runnable) :

(4) Thread(Runnable, string) :

### Instance Methods:

(1) public final void setName(String)

(2) public final String getName()

Methods (1) & (2) are used for setting & getting the name of the thread.

Eg: Thread t1 = new Thread();

String tname = t1.getName();

```

    System.out.println("Sop(tname); // Thread-0");
    t1.setName("javath");
    tname = t1.getName();
    System.out.println("Sop(tname); // javath");
  
```

- (3) public final void setPriority (int)  
 (4) " " int getPriority ()

The above methods are used for setting & getting the priorities of the thread.

Ex: Thread t1 = new Thread ("sumth");  
 int pri = t1.getPriority ();  
 System.out.println("Sop(pri); // 5-- NORM\_PRIORITY");  
 t1.setPriority (Thread.MAX\_PRIORITY);  
 pri = t1.getPriority ();  
 System.out.println("Sop(pri); // 10");

- (5) public final boolean isAlive():

This method returns true provided thread is in ready & running & waiting states & it returns false provided the thread is in new & started states.

- (6) public final void suspend()

- (7) public final void resume()

Method (6) is used for suspending the execution of currently executing thread. Once the thread is suspended, -no temporarily result of the thread will be placed into PCB/JCB (process/job control block). When the thread is suspended, that particular thread will be running state to waiting state.

Method (7) is used for transferring the thread from waiting state to ready state when the thread is resumed. (so started after the suspension) the thread starts its execution.

where it left off before its suspension by retrieving the temporary  
the temporary result from PCB.

(8) public final void stop():

This method is used for stopping the currently executing  
thread. When the thread is stopped, it enters from running state  
to Halted state. When the thread is restarted, then it starts  
executing from the beginning but not where it was stopped.

(9) public final void start():

This method is used for bringing the thread from new state to  
ready state when we call start method upon the thread obj, it is  
automatically calling the run() by providing an

extra

\* public void final start():

This method is used for bringing the thread from new state to ready stat. When we call start() upon the Thread object, it is automatically calling the run() method by providing internal services of multi-threading such as states of the Thread, Thread synchronization, inter thread communication applications etc.

\* psx: Thread t = new Thread(); // new state

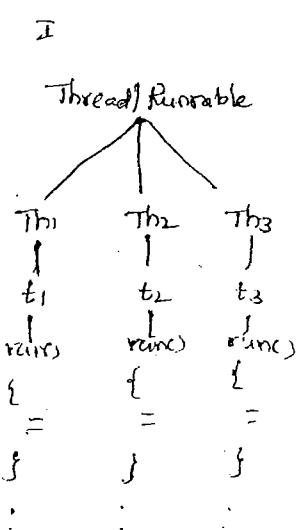
boolean b = t.isAlive();

Sop(b); // false

t.start(); // st ReadyState.

b = t.isAlive();

Sop(b); // true



\* Public void run():

This method will be automatically called by the start() provided java programmers call the start() upon the Thread object. Java programmers is not recommended to call run() of Thread class bcz we never get any internal services of multi-threading.

As a java programmer we define logic of the Thread within the run(). In other words run() is always used for defining the logic of the Thread.

Originally run() is defined in Thread class with null body.

As a Java programmer to define a logic of the Thread we must override the run() of Thread class into our derived class by extending java.lang.Thread.

```
Ex: class Th1 extends Thread  
{  
    public void run()  
    {  
        =  
        // logic of the Thread  
        =  
    }  
}  
// Th1
```

As long as thread is executing + whose run(), whose state is running state.

\* public final void join() throws java.lang.InterruptedException

This method is used for joining the threads after their completion of their execution as a single unit. Because of this internal performance of the multithreading will be improved.

The default approach of multithreading is that as and when threads are completed their execution, those threads are collected by Thread Group individually and handing it to Garbage Collector one by one, because of this there is a possibility of loosing the performance.

Ex: `tey`  
{  
`t1.join();`  
`t2.join();`  
`t3.join();`

Catch (InterruptedException ie.)

{

System.out.println("problem in thread execution");

}

- \* InterruptedException is never occurs in single machine but it occurs only in the case busy environment like client-server environment.

Let us assume n threads are started their execution and

(n-1) threads are completed their execution, joined and still nth thread is under execution. During this period of time if ThreadGroup is trying to collect (n) completed threads and trying to handover it to the Garbage Collector then JVM generates a predefined exception called java.lang.Interruptedexception.

### Static methods:

Static methods of java must be called with respect to class name.

③ ② ①  
1. public static final Thread currentThread();

This method is used for obtaining those threads which are by default running in each and every java program and more over it is one of the factory method.

Ex: Thread t1 = Thread.currentThread();

FGT → ForeGround Thread

TGp → Thread Group

System.out.println(t1); // Thread [main, 5, main]  
 ↓      ↓      ↓  
 FGT    priority    TGp

t1.setName("javath");

System.out.println(t1); // Thread [javath, 5, main]  
 ↓      ↓      ↓  
 FGT    priority    TGp

One can change the name of the foreground thread but we can't change Thread Group name.

## 2. public static final sleep (long msec) throws InterruptedException

This method is used for making the thread to go to sleep from its current execution for a period of time and it must be specified in terms of millisecond ( $1\text{ sec} = 1000\text{ msec}$ ). Making the thread to sleep is nothing but making the thread to come to waiting state. Once the sleep time is completed, automatically the thread will be entered into ready state from waiting state.

This method throws a predefined exception called `java.lang.InterruptedException`.

Let us assume that there exist two different multithreaded programs and first program threads are sleeping in the server memory and becoz of the server mismemory management if second thread programs are trying to sleep in the same place of server where first program threads are sleeping then we get a predefined exception called `java.lang.InterruptedException`.

→ Write a java program which will print the preliminary information about thread such as name of the threads which are by default executing, whose execution status, default name of the FGT, whose execution status, priority modifier values of the thread etc.

```
class Thread1
{
    public main (String[] args)
    {
```

Thread t<sub>1</sub> = Thread. CurrentThread();

Sop(" default threads = " + t<sub>1</sub>) ; // Thread [main,5, main]

t<sub>1</sub>. setName ("janath");

Sop(" default threads after modification = " + t<sub>1</sub>) ; // thread[janath,5  
main]

Sop(" executionstatus of t<sub>1</sub> = " + t<sub>1</sub>. isAlive()); // true

Thread t<sub>2</sub> = new Thread(); // new

Sop (" default name of t<sub>2</sub> = " + t<sub>2</sub>. getName()); // Thread-0

Sop (" exec status of t<sub>2</sub> = " + t<sub>2</sub>. isAlive()); // false

Sop (" val of Max pri = " + Thread. MAX\_PRIORITY); // 10

Sop (" val of Min pri = " + Thread. MIN\_PRIORITY); // 0

Sop (" val of Norm pri = " + Thread. NORM\_PRIORITY); // 5

}

}

### Internal flow of threads :

When we write any multithreading application, there is a possibility of creating multiple threads. When multiple threads are executing, they follow the following sequence of steps

- (1) Java program starts executing
- (2) JVM creates a Thread Group Name (TGN) & resides in main()
- (3) TGN creates ForeGround Thread (FGT) & resides in run()
- (4) TGN dispatches the FGT(s) to their respective run()
- (5) FGT enters into run(), executes run() and gives result to TGN either all at once or one by one.

- (6) TGN receives the result from FGTS either all at once or one by one
- (7) TGN gives the result to Java program/ user.
- (8) TGN collects FGTS and handover to GC.
- (9) JVM collects TGN and handover to GC.
- (10) Java program stops executing.

→ Write a Java program which will display 1 to 10 numbers & after each and every sec. using threads

class th extends Thread

```
{
    public void run()
    {
        try
        {
            for (int i=1; i<=10; i++)
            {
                System.out.println("val of i = " + i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException ie)
        {
            System.out.println("problem in Thread execution");
        }
    }
}
```

} // th

class Thdemo2

```
{
    public static void main(String[] args)
    {
        th t1 = new th(); // new state
        System.out.println("status of t1 b4 start = " + t1.isAlive());
    }
}
```

```

    t1.start();
    System.out.println("Status of t1 after start = " + t1.isAlive());
    try {
        Thread.sleep(1000);
    } catch (InterruptedException ie) {
        System.out.println("problem in Thread execution");
    }
    System.out.println("Status of t1 after completion = " + t1.isAlive());
}
}

```

### Output:

Status of t1 before start = false

Status of t1 after start = True

Value of f1, 2, 3, ..., 10

Status of t1 after execution = false.

In the above program Thread Group Name always resides in main() and executes the block of starts in main().

t1 is an obj of th which informs which is an obj of Thread class will always resides in main() and executes block of starts in main().

### (2) By using java.lang.Runnable interface:

Runnable is one of the predefined interface which is used for development of multi threading applications.

This interface contains only one method and whose prototype is as follows:

The above method is used for defining the logic of the thread. In order to provide the logic of the thread, the run() of Runnable (I) must be overridden into our derived classes by implementing the Runnable interface.

Ex: class th1 implements Runnable

```
    {  
        public void run()  
        {  
            }  
            } } logic of the thread  
        }  
    } // th1
```

Runnable t1 = new th1();

(Or)

th1 t1 = new th1();

An obj of Runnable interface is nothing but an object of its subcl

#### ⑤ Thread (Runnable):

This constructor is used for converting Runnable (I) obj into Thread clc obj to make use of all the methods of Thread clc (specially start()) to enter into run() of Runnable interface.

Ex: Runnable t1 = new th1();

(Or)

th1 t1 = new th1();

// t1.start(); invalid

Thread t11 = new Thread(t1);

t11.start(); // valid

## 4 Thread (Runnable, String):

- This constructor is also used for converting Runnable (I) obj into Thread cls obj and additionally this constructor gives name of the thread

```
Runnable t1 = new thi();
(Or)
thi t1 = new thi();
Thread t11 = new Thread(t1, "avath");
t11.start();
```

Sop "Name of the Thread" + t11.getName();

In real time applications it is highly recommended to develop multi threading applications by implementing Runnable interface but not with Thread cls and we must convert Thread Runnable intf obj into Thread cls obj.

→ WAP which will print 1 to 10 nos after each and every second with Runnable interface

```
class thi implements Runnable
{
    public void run()
    {
        // code is similar to previous prog run()
    }
}
```

```
class thdemo3
{
    public static void main ( )
    {
        // Runnable t1 = new thi();
        thi t1 = new thi();
```

```

Thread t1 = new Thread(t1, "javath"); // Converting Runnable(I) obj
// into Thread clz obj by giving the name of the thread
System.out.println("Name of the Thread=" + t1.getName());
System.out.println("Status of t1 before start=" + t1.isAlive());
t1.start();
System.out.println("Status of t1 after start=" + t1.isAlive());
}
}

```

⇒ Write a Java program which will implement banner animation or  
 " " " " which will scroll the given msg.

1Ani.java

```

import java.awt.*;
import java.applet.*;

/* <applet code="Ani" height=300 width=300>
</applet> */

public class Ani extends Applet implements Runnable
{
    String msg = "WELCOME TO JAVA";
    public void init()
    {
        setBackground(Color.cyan);
        setForeground(Color.red);
    }
    public void start()
    {
        Thread t = new Thread(this);

```

// 'this' refers to current clz obj, which implements Runnable  
 interface and converts Runnable intf obj into Thread clz obj  
 for making use of all the methods of Thread clz especially start()

```

        t.start();
    }

    public void paint(Graphics g)
    {
        Font f = new Font("arial", Font.BOLD, 60);
        g.setFont(f);
        g.drawString(msg, 100, 100);
    }

    public void run()
    {
        try
        {
            while (true)
            {
                char ch = msg.charAt(0);
                msg = msg.substring(1, msg.length());
                msg = msg + ch;
                Thread.sleep(400);
                repaint(); // Calling the paint() repeatedly
            } // while
        } // try
        catch (InterruptedException ie)
        {
            System.out.println("problem in Thread exc");
        }
    } // run
} // Ani - class

```

⇒ WAP which will increment the number automatically

// AutoIncr.java

```

import java.awt.*;
import java.applet.*;

```

```
/* <applet code = "AutoIncr" height = 300 width = 300 >
</applet> */
```

```
public class AutoIncr extends Applet implements Runnable
```

```
{
```

```
int i = 0;
```

```
public void init()
```

```
{ // Similar to previous prog
}
```

```
public void start()
```

```
{ // code is similar to prev prog
}
```

```
public void paint(Graphics g)
```

```
{
```

```
Font f = new Font("arial", Font.BOLD, 60);
```

```
g.setFont(f);
```

```
String s = String.valueOf(i);
```

```
g.drawString(s, 100, 100);
```

```
}
```

```
public void run()
```

```
{
```

```
try
```

```
{ while (true)
```

```
{
```

```
++i;
```

```
repaint();
```

```
Thread.sleep();
```

```
}
```

```
}
```

```
Catch (InterruptedException e)
```

```
{ System.out.println("problem in thread exec");
```

```
} // end of run
```

```

        t.start();
    }

    public void paint (Graphics g)
    {
        Font f = new Font ("arial", Font.BOLD, 60);
        g.setFont (f);
        g.drawString (msg, 100, 100);
    }

    public void run()
    {
        try
        {
            while (true)
            {
                char ch = msg.charAt (0);
                msg = msg.substring (1, msg.length ());
                msg = msg + ch;
                Thread.sleep (400);
                repaint (); // Calling the paint() repeatedly
            } // while
        } // try
        catch (InterruptedException ie)
        {
            System.out.println ("problem in thread exc.");
        }
    } // run
} // Ani - cls

```

⇒ W A J P which will increment the number automatically

// AutoIncr.java

```

import java.awt.*;
import java.applet.*;

```

```
> <applet code="AutoIncr" height=300 width=300>  
</applet></pre>
```

```
public class AutoIncr extends Applet implements Runnable
```

```
{
```

```
    int i = 0;
```

```
    public void init()
```

```
{  
    // Similar to previous prog  
}
```

```
    public void start()
```

```
{  
    // code is similar to prev prog  
}
```

```
    public void paint(Graphics g)
```

```
{
```

```
        Font f = new Font("arial", Font.BOLD, 60);
```

```
        g.setFont(f);
```

```
        String s = String.valueOf(i);
```

```
        g.drawString(s, 100, 100);
```

```
}
```

```
    public void run()
```

```
{
```

```
    try
```

```
    { while (true)
```

```
    {
```

```
        ++i;
```

```
        repaint();
```

```
        Thread.sleep();
```

```
    }
```

```
}
```

```
    catch (InterruptedException e)
```

```
    { System.out.println("problem in thread exec");
```

```
} // end of run
```

- WAP to design a digital clock using applets with threads  
(Hint: HH:MM:SS)

### Synchronization:

- If anything is sharable and multiple application programs are trying to access then there is a possibility of getting inconsistent result. To eliminate this inconsistent result we must apply the concept of thread synchronization.

The advantage of thread synchronization is to achieve consistent result.

- Def of Synchronization: The process of allowing only one thread at a time among many threads into the area which is sharable to perform read and write operations.

- A sharable area is the one in which if multiple threads are operating concurrently then the sharable area is containing inconsistent result. To eliminate this inconsistent result, we apply thread synchronization techniques.

### Necessity of Synchronization:

- Let us assume there exist a sharable variable balance whose initial value is '0'. There exist two threads  $t_1$  and  $t_2$  and they want to update the sharable variable value with 10 and 20 respectively. When both the threads are started at same time for updating a bal

variable value (assume  $t_1$  started first and later  $t_2$  started).

- After completion of these two threads the final value of the bal variable is either 10 or 20 but not 30 which is an inconsistent result.

To eliminate this inconsistent result we apply the concept of thread synchronization techniques.

## Synchronization concept applied:

Assume two threads  $t_1$  and  $t_2$  started their execution ( $t_1$  starts first and  $t_2$  starts later) for updating the value of the `bal` variable. Since  $t_1$  starts first the value of `bal(10)` given to  $t_1$  and locks the variable value by the JVM. During this period of time if the thread  $t_2$  trying to access the `bal` variable variable then the thread  $t_2$  will be made wait until  $t_1$  thread completed its execution. The thread  $t_1$  completes its execution and the `bal` variable will be unlocked. JVM gives the value of `bal` variable `(10)` given to thread  $t_2$  and once again the `bal` variable will be locked. Once the thread  $t_2$  completed its execution, the value of the `bal` variable is `30` and it is unlocked. The value `30` is one of the consistent result.

Therefore in the synchronization concept the process of locking and unlocking will be continued until all the threads are completed their execution.

## Thread Synchronization Techniques:

In Java programming Synchronization techniques are divided into two types. They are

- (1) Synchronized methods
- (2) Synchronized blocks

### Synchronized methods:

If any ordinary method is sharable or accessed by multiple threads then there is a possibility of getting inconsistent result.

To get the consistent result, the definition of the ordinary method must be made as synchronized method by using 'synchronized' keyword.

Based on the synchronized keyword we are writing before the methods, Synchronized methods are divided into two types. They are

- a. synchronized instance methods
- b. synchronized static methods

### Synchronized Instance methods:

If any ordinary instance method is sharable or accessed by multiple threads then there is a possibility of getting inconsistent result.

To eliminate this inconsistent result the ordinary instance methods def must be made as synchronized by using synchronized key word.

\* Once the instance method is synchronized, the corresponding class obj will be locked.

Syntax: synchronized returnType methodName(list of formal params if any)

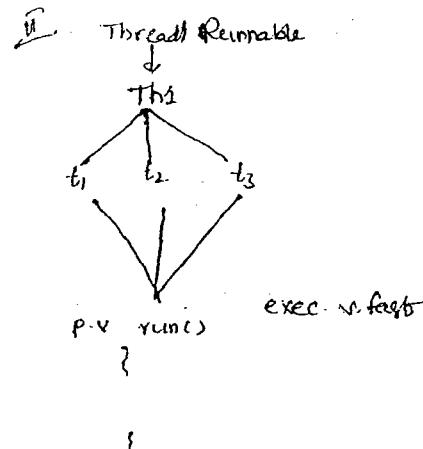
```

    {
        Block of stat(s);
    }
  
```

Eg: class Account

```

    {
        int bal = 0;
        synchronized void deposit(int amt)
        {
            bal = bal + amt;
            System.out.println("curr bal = " + bal);
        }
    }
  
```



### Synchronized static methods:

If any ordinary static method sharable or accessed by multiple threads then there is a possibility of getting inconsistent result.

To eliminate this inconsistent result the ordinary static method must be made as synchronized method by using synchronized key word.

Once the ordinary static method is synchronized, the corresponding class will be locked.

### Syntax:

Synchronized static return-type methodname( list of formal params if any)  
{  
    Block of Stmt(s);  
}

Ex: Class Account

```
{  
    static int bal=0;  
    synchronized static void deposit( static int amt)  
    {  
        bal= bal+amt;  
        System.out.println("curr bal = "+bal);  
    }  
}
```

### Synchronized blocks:

If we inherit non synchronized instance methods either from base class or from base interface and if the inherited non synchronized method sharable or accessed by multiple threads then there is a possibility of getting inconsistent result.

To eliminate this inconsistent result, the derived class programmer is attempting to write synchronized keyword before inherited non synchronized instance method. which is not possible becoz a derived class programmer does not have a privilege to change the prototype of either base class methods or base interface methods.

In any circumstance to get the consistent result for the above problem we must use the concept of synchronized blocks.

Syntax: synchronized( Obj of current class)  
{  
    Block of Stmt(s);  
}

- \* Synchronized blocks must be always written within non synchronized instance methods.

O Q:

```
package ap;
public abstract class Account
{
    public void deposit( int amt )
    {
        // null body
    }
}
```

```
Package ap;
public interface Account
{
    void deposit( int amt );
    // public abstract void deposit(
    //     int amt );
}
```

import ap.Account;

```
class SAccounts
{
    int bal = 0;
```

public void deposit( int amt )

```
{
    synchronized( this )
    {
        bal = bal + amt;
        System.out.println("Curr bal=" + bal);
    }
}
```

}

⇒ WAP which illustrate the concept of synchronization (or)

problem: Let us assume there exist an account with the variable bal and whose initial value is 0.

Let us assume there exists 5 customers who want to update or deposit Rs 10 in the bal variable for each. Concurrently.

Ensure the bal variable of account must contain a consistent result.

Class Account

```
{ int bal=0;  
  synchronized void deposit( int amt)  
  {  
    bal = bal + amt;  
    System.out.println("Current bal=" + bal);  
  }  
  int getBal()  
  {  
    return bal;  
  }  
}
```

} // Account

Class cust extends Thread

```
{ Account ac;  
  Cust(Account ac)  
  {  
    this.ac = ac;  
  }  
  public void run()  
  {  
    ac.deposit(10);  
  }  
}
```

} // Cust

Class Synch

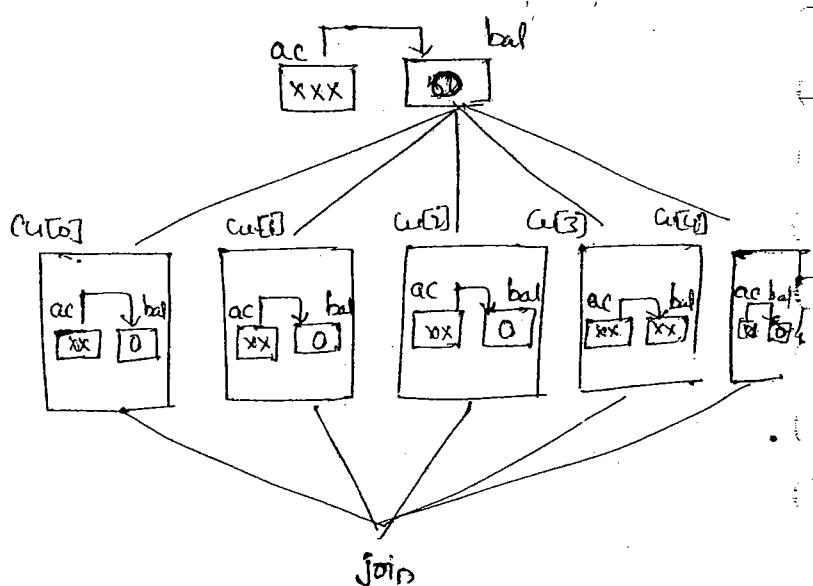
```
{  
  public static void main(String[] args)  
  {
```

// Step-1  
 Account ac = new Account();

// Step-2  
 Cust cu[5] = new Cust[5];

// Step-3  
 for (int i=0; i<5; i++)

```
    {  
      cu[i] = new Cust(ac);  
    }
```



```

    || Step -4
    for (int i=0; i<5; i++)
    {
        cu[i].start();
    }
    || Step -5
    try
    {
        for (int i=0; i<5; i++)
        {
            cu[i].join();
        }
    }
    catch (InterruptedException e)
    {
        System.out.println("problem in thread exec");
    }
    || Step -6
    System.out.println("total bal = " + ac.getBal());
} || main
}

```

|| class

(DD5 & Windows 95)

Note: If we run the above prog on single user OS's then it is mandatory for the java programmer to write synchronized keyword where ever it is necessary otherwise we get inconsistent result.

When we run the above program on multi user OS's (Windows 98, XP, 2000...) it is optional to the programmer to write synchronized keyword bcoz the concept of synchronization is automatically taken care by OS.

It is highly recommended for the Java programmer to write synchronized keyword where ever we applying synchronization concept.

In the above prog

Step1: creating an obj of Account class

Step2: creating no. of Customer objs (thread objs)

Step3: Give Account class obj to the multiple Customer class objs

Step4: Start the Customer class objs for depositing Rs 10 in the bal  
variable of Account class

Step5: Join the completed Cust threads as a single unit for  
collectively handing over to garbage Collector.

Step6: Display the total balance which was deposited by various  
Cust class objs

### Inter Thread Communication (ITC):

#### Definition:

Inter Thread Communication is one of the distinct  
facility in multi-threading. This concept brought  
from ds and it was available in os as inter  
process communication.

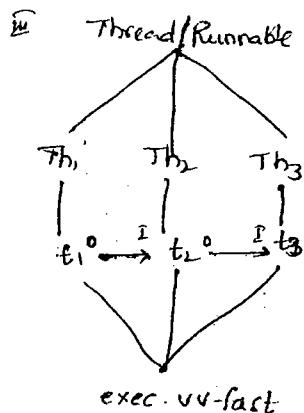
InterThread Communication apps are very fast compared to all the apps  
in java.

The real time implementation of ITC is in Realtime Servers development.

Definition: The process of establishing the communication b/w multiple  
threads is known as Interthread communication. (ox)

If the result of first thread is given as input to second thread, the  
result of second thread is given as input to third thread. etc. Then the  
communication b/w first, second, third threads is known as ITC.  
(ox)

When we are able to exchange the data b/w multiple threads of  
different subclasses of either Thread or Runnable is known as ITC.



- In order to develop ITC apps we need to use the methods of java.lang.Object class. And these methods are known as ITC methods.

### The ITC methods of java.lang.Object:

1. public void wait (long msec) } throws InterruptedException
2. public void wait () }
3. public void notify()
4. public void notifyAll()

Method(1), is used for making the thread to wait for a period of time in terms of msec. If the waiting time of the thread is completed then the thread will be entered automatically into ready state from waiting state.

Method(2), used for making the thread to wait without specifying any waiting time. This method is more advisable than method (1) to the Java programmers bcoz a Java programmer may not be exactly determine at what time, the thread completes its execution to make the other thread to wait.

Methods(1) and (2) generates a predefined exception called java.lang.InterruptedIOException. This exception never occurs in single machine but it occurs only in the case of busy environment like client-server environment.

Let us assume some of the threads of first program are waiting in the server memory and some other threads of second prog of different user trying to wait in the same place where first prog threads are waiting due to mismemory management of the server, JVM generates a predefined exception called java.lang.InterruptedIOException with respect to threads of first program.

Method (3) is used for transforming one thread at a time from waiting state to ready state.

Method (4), used for transforming all the threads at a time from waiting state to ready state.

The classical examples of ITC are:

- a. producer-consumer problem
- b. Dining philosophers problem
- c. Readers-writers problem
- d. Barber shop problem

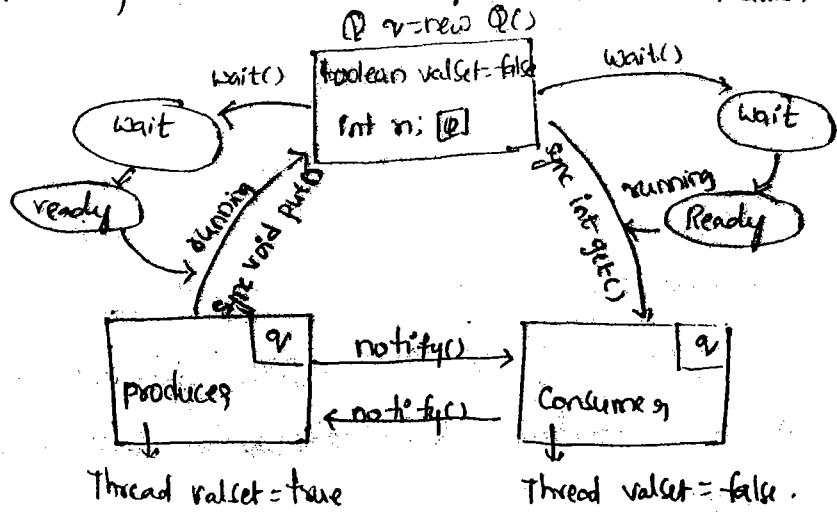
### Implementation of producer-consumer problem:

Producers: producer is one of the thread in Java whose basic functionality is to produce only one item at a time. that is no two consecutive items to be produced by the producer. producer will be in waiting state until the produced items to be consumed by the consumer.

Consumers: consumer is also one of the thread in Java and whose basic functionality is to consume the items which are produced by the producer. Consumer thread can't consume two items at a time.

Consumer thread will be in waiting state until the producer thread produces the same items.

This process is continued b/w producer and consumer threads until we stop. Ensure the consistency communication b/w producer and consumer threads.



Q → Write a Java program which will implement the producer-consumer programs

class Q

{

int n;

boolean valSet = false;

synchronized void put(int i)

{

try

{

if (valSet)

{

wait();

}

else

{

n = i;

System.out.println("put = " + n);

valSet = true;

notify();

}

}

Catch(InterruptedException ie)

{

System.out.println("problem in thread exec");

}

} // put

synchronized int get()

{

try

{

if (!valSet)

{

wait();

}

else

{

System.out.println("got = " + n);

valSet = false;

```
    notify();
}
}

Catch (InterruptedException ie)
{
    S.eplinc" problem in thread exec();
}
return(n);
} // get
```

Class prod implements Runnable

```
{
    Q q;
    prod(Q q)
    {
        this.q=q;
        Thread t1=new Thread(this);
        t1.start();
    }
    public void run()
    {
        int i=0;
        while(true)
        {
            q.put(i++);
        }
    }
}
```

Class cons implements Runnable

```
{
    Q q;
    cons(Q q)
    {
        this.q=q;
        Thread t1=new Thread(this);
        t1.start();
    }
}
```

```

    public void run()
    {
        while (true)
        {
            int i = q.get();
        }
    } // run
} // cons

class PCDemo
{
    public static void main (String[] args)
    {
        Q q = new Q(); // ①
        Prod p0 = new Prod(q); // ②
        Cons c0 = new Cons(q); // ③
    }
}

```

> classpath = %classpath% ; ;

In real world applications <sup>thread</sup> interthread communication apps are very fast compared to any of the applications.

## DESIGN PATTERNS :

Def: Design patterns are the best series invented and proved by industry experts and released to the industry to eliminate the side effects which are recursively occurring in SW development.

The purpose of design patterns is to provide a permanent solution for a problem by eliminating the side effects.

In SW development there are several design patterns. As

As a part of J2EE we have two design patterns. They are-

1. factory method
2. Singleton class

### Factory method:

A factory method is one whose return type must be similar to class name.

The purpose of factory method is to create an obj of a class without using 'new' operator.

### Rules for factory method:

- (1) The return type of the factory method must be similar to class name where it presents.
- (2) Every factory method must be static.
- (3) The access specifier of the factory method must be public.

Eg 1: java.lang.Thread

③ ② ①  
public static final Thread currentThread()

2. java.lang.String

③ ② ①  
public static final String valueOf(XXX)

3. Graphics Environment

① ① ①  
public static G-E getLocalG-E()

## Singleton class:

A singleton class is a Java class which allows us to create only one

obj for JVM

Steps for developing singleton class:

(1) choose an appropriate package and ensure it should be the first executable statement.

(2) choose an appropriate class and this class is going to be called as Singleton class and ensure whose modifier must be public.

(3) Each and every singleton class must contain self declared object on its own and ensure that it should be private static.

(4) Each and every singleton class must contain private constructor.

(5) Each and every singleton class must contain a factory method by satisfying its rules.

(6) Whatever the Singleton class which is placing in the package, that class name must be given as a filename with an extension .java.

Ex: Design a singleton class.

```
1. stest.java // →⑥
  package st; // →⑦
  public class stest // →⑧
  {
    private static Stest st = null; // →⑨
    private stest(); // →⑩
    {
      System.out.println("Obj created first time");
    }
    public static Stest create() // →⑪
    {
      if(st==null)
      {
        st = new Stest();
      }
      else
        System.out.println("Obj already created");
    }
  } // method
```

Compile the above Java program

>javac -d. Stest.java ↴



⇒ Write JP which illustrate the concept of Singleton class functionality.

```
|| dppdemo.java
import st.Stest;
class dppdemo
{
    public static void main (String[] args)
    {
        || Stest s1 = new Stest(); ----> invalid
        . Stest s1 = Stest.create();
        Stest s2 = Stest.create();
        Stest s3 = Stest.create();
        if ((s1 == s2) && (s2 == s3) && (s3 == s1))
            System.out.println("All objs are same & one");
        else
            System.out.println("All objs are different");
    }
}
```

In the above prog the class Stest allows us to create only one obj per JVM  
Hence Stest class is known as Singleton class.

pop C" Ifs (we found it = " + c. search(mia.AutoComplete("101"))); U3

pop C" Ifs 10 found it = " + c. search(mia.AutoComplete("102")); U3

pop C" current after peek = "fs"; U [10, 20, 30]

pop C" Top most element = " + s.peek(); // 30

pop C" Current after deletion = " + s; // [10, 20, 30]

pop C" deleted element = " + s.pop(); // 14

pop C" s is empty? = " + s.isEmpty(); // false

pop C" size = " + s.size(); // 4

pop C" content = " + s; // [10, 20, 30, 40]

s.push (new Integer(40))

s.push (new Integer(30))

s.push (new Integer(20))

s.push (new Integer(10))

add the data to the stack

pop C" Is s empty = " + s.isEmpty(); // true

p C" size = " + s.size(); // 4

p C" content = " + s; // [10, 20, 30, 40]

All Stack & Java

- Method (5) is used for searching or obtaining the top most element in the stack.
- Method (6) is used for searching an element into the stack. This method return relative stack array position provided the search is successful. If returns -1 if failed search is unsuccessful.
- Method (7) is a Java program which illustrates the concept of stacks.

} llmain

} ll stack

## Methods in Java.util.SortedSet:

We know that SortedSet is the sub interface of Set. So that all the methods of Set are inherited into SortedSet which are nothing but methods of Collection.

Even though the methods of Set and SortedSet are same, the method of Set are defined in some predefined class in such a way that unique elements are allowed and they must be displayed in random order. Whereas methods of SortedSet are defined in some other predefined class in such a way that unique elements are allowed and they are displaying in sorted order.

① public Object first()

② public Object last()

③ public SortedSet headSet (Object obj)  $\Rightarrow$   $x_i <= obj$

④ public SortedSet tailSet (Object obj)  $\Rightarrow$   $x_i > obj$

⑤ public SortedSet subset (Object obj1, Object obj2)  $\Rightarrow$   $obj1 \leq x_i \leq obj2$

Ex: (see the explanation of last methods)

ss

Object obj = ss.first();

10

SortedSet

Object hs = ss.headSet (new Integer(30))

10

20

30

40

50

60

70

80

SortedSet

Object hs = ss.headSet (new Integer(30))

10

20

30

SortedSet

Object ls = ss.tailSet (new Integer(50))

60

70

80

Object obj = ss.last();

80

Collection Framework classes: 1D CFW classes are those which are containing the definition for those abstract methods which are coming from 1D CFW interfaces.

All the predefined classes of 1D CFW are present in java.util.\* package.

The following table gives 1D CFW interface name, whose corresponding 1D CFW class name and their hierarchy.

<u>1D CFW intf name</u>	<u>1D CFW class name</u>	<u>Hierarchy</u>
Collection	① AbstractCollection	implements Collection
List	② AbstractList	{ extends AbstractCollection implements List }
Set	③ AbstractSet	{ extends AbstractCollection implements Set }
SortedSet		
	④ AbstractSequentialList	extends AbstractList
	⑤ LinkedList	extends AbstractSequentialList
	⑥ ArrayList	extends AbstractSequentialList
	⑦ HashSet	extends AbstractSet
	⑧ TreeSet	{ extends AbstractSet implements SortedSet }

In the above hierarchy chart the classes ①, ②, ③ & ④ are predefined abstract classes and whole obj can't be created directly. Hence they will not be used directly in real time applications. But all these classes are playing a very important background role in defining abstract methods which are inherited from predefined 1D CFW interfaces. More over these classes are treated as playing a middle man role b/w predefined 1D CFW interfaces and predefined bottom most concrete subclasses of 1D CFW.

The classes **⑤**, **⑥**, **⑦** & **⑧** are predefined concrete sub classes of all abstract classes of 1D CFW. Since these are concrete, whole objs can be created directly. So that we can use in real-time applications.

### LinkedList:

→ LinkedList is one of the concrete subclass of all abstract classes of 1D CFW.

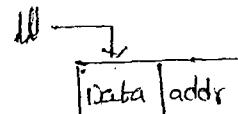
1000	10
1004	20
1008	30

$$a[0] \Rightarrow a+0.$$

$$\begin{aligned} &= a[\text{size}(a) * 0] \\ &= 1000 + (1 * 0) \\ &= 1000 \end{aligned}$$

→ Creating a linkedList is nothing but creating an object of **LinkedList**.

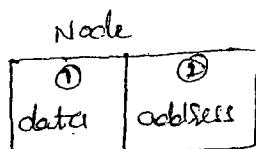
Ex: `LinkedList ll = new LinkedList();`



Here **ll** is an object of **LinkedList** which is nothing but CFW available, which allows us to group diff type of values or similar type of values & both.

→ In **LinkedList** the data is organizing internally in the form of nodes.

→ Node contains two parts and whose structure is

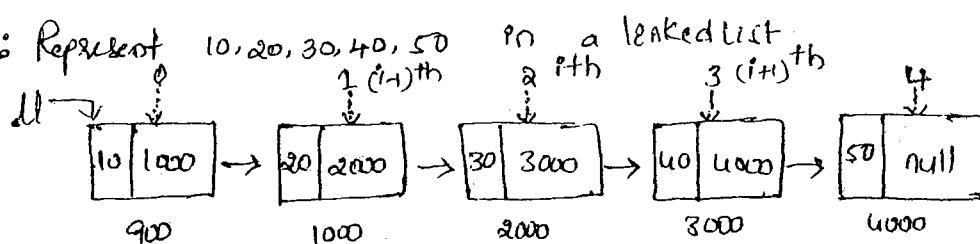


Every obj of **LinkedList** is nothing but a name of the node.

→ The data part represents the type of data being represented in **LinkedList** class object.

The address part represents address of the next node and for the last node address part must be null which indicates end of the **LinkedList**.

Ex: Represent 10, 20, 30, 40, 50 in a linked list.

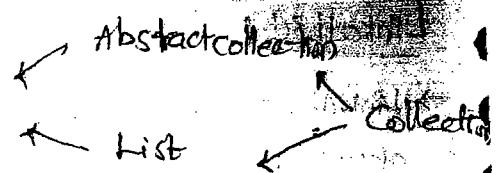


The data organized in **LinkedList** in such a way that address of *i*th node stored in address part of  $(i-1)^{\text{th}}$  node, address part of *i*th node contains address of  $(i+1)^{\text{th}}$  node. and for the last node address part must be null which indicates end of the **LinkedList**.

Linked list in Java programming, the node values can be inserted at the positions.

The hierarchy of LinkedList is

LinkedList ← AbstractSequentialList ← AbstractList



LinkedList class profile:

Constructors:

- ① LinkedList()
- ② LinkedList(int size)

Instance Methods

- 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()

2/5/11

AB API Interfaces:

AB API interfaces are divided into 3 types

1. java.util.Map

2. java.util.Map.Entry

3. java.util.ListedMap

java.util.Map:

1. It is one of the top most predefined interface in AB API interfaces.
2. An object of Map interface can't be created directly but it can be instantiated indirectly.

Ex: Map m = new xyz();

Here m is an indirect obj of Map interface and xyz is the sub class of Map interface.

3. An object of Map allows us to organize the data in the form of (key, value) in (Key, Value). The value of key must be distinct and the value of value may or may not be distinct.
4. The values of (key, value) must be objects.

5. Whatever data we add to the Map interface variable, that data is by default displayed in random order, that is in which ever

In order we add the entries to the Map interface variable, in the same order the data will be displayed.

### Methods in Map interface:

(1) public int size();

This method is used for finding no. of elements in 2D CFW variable determining the size of 2D CFW variable.

(2) public boolean isEmpty();

This method returns true provided 2D CFW variable does not contain any elements (empty). This method returns false provided 2D CFW var contains some elements (non-empty).

(3) public void put (Object kobj, Object vobj);

This method is used for adding the data to the 2D CFW variable in the form of (key, value)

(4) public Object get (Object kobj);

This method is used for obtaining value of value by passing key value

If the key value is not found in 2D CFW variable then the value of value is null. otherwise it is not null.

Ex: Object vobj = m.get (new Integer(10));

System.out.println(vobj); // 1.5

Object vobj1 = m.get (new Integer(90));

System.out.println(vobj1); // null

m	↓
10	1.5
20	1.0
30	1.9

(5) public Object void remove (int);

(6) public void remove (Object kobj);

The above methods are used for removing the entries of 2D CFW variable.

Either on the basis of position or on the basis of Content ie key object

(7) public void removeAll();

This method removes all the entries of 2D CFW variable.

10) public Set entrySet();

This method is used for extracting or retrieving the content of Map variable.

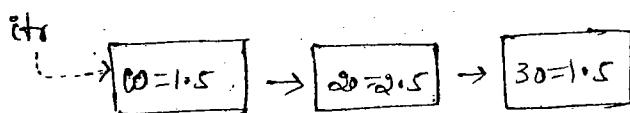
Ex: `Sop(m); // {10=1.5, 20=2.5, 30=3.5} (Unnamed Set)`

Set s = m.entrySet();

// s = {10=1.5, 20=2.5, 30=3.5}

// entrySet → Set → Iterator()

Iterator itr = s.iterator();



while(itr.hasNext())

{  
Object obj = itr.next(); // 4<sup>th</sup> step

Map.Entry me = (Map.Entry) obj; // 5<sup>th</sup> step

Object kobj = me.getKey();  
Object vobj = me.getValue(); } 5<sup>th</sup> step

Integer iobj = (Integer) kobj; } 6<sup>th</sup> step

Float fobj = (Float) vobj; } 6<sup>th</sup> step

int aeno = iobj.intValue(); } 7<sup>th</sup> step

float bal = fobj.floatValue(); } 7<sup>th</sup> step

Sop(bal + " is the bal of " + aeno);

} // while

Ex: `Sop(m); // {10=1.5, 20=2.5, 30=3.5}`

Set s = m.entrySet();

// s = {10=1.5, 20=2.5, 30=3.5}

// entrySet → Set → toArray()

Object obj[] = s.toArray(); // 4<sup>th</sup> step

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

{

10	1.5
20	2.5
30	3.5

```

Map.Entry me = (Map.Entry) mobj[i]; // 5th step
Object kobj = me.getKey();
Object vobj = me.getValue(); } 5th step

Integer iobj = (Integer) kobj; } 6th step
Float fo = (Float) vobj; }

int acno = iobj.intValue();
float bal = fo.floatValue(); } 7th step

System.out.println("The bal of " + acno);
}
}

```

### public Set keySet();

This method is used for obtaining set of keys from map variable and pass these set of key values to get() and obtains set of values.

Ex: `Set<String> s = m.keySet();` { (10=1.5), (20=2.5), (30=3.5)}

`Set s = m.keySet();`

`Set s = {10, 20, 30};`

`for (KeySet s : Set s) {`

`Iterator it = s.iterator(); } 4th step`

`it`

`10 → 20 → 30`

`while (itr.hasNext())`

`{`

`Object kobj = itr.next();`

`Object vobj = m.get(kobj); } 5th step`

`Integer iobj = (Integer) kobj;`

`Float fo = (Float) vobj; } 6th step`

`int acno = iobj.intValue();`

`float bal = fo.floatValue(); } 7th step`

`System.out.println("The bal of " + acno);`

m	↓
10	1.5
20	2.5
30	3.5

~~Set~~ Set: Sop(m); || {(10=1.5), (20=2.5), (30=3.5)}

Set s = m.keySet();

|| s = {(10=1.5), (20=2.5), (30=3.5)}

|| keySet() → Set → toArray()

Object kobj[] = s.toArray();

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

{

Object vobj = m.get(kobj[i]);

Integer i0 = (Integer) kobj[i];

Float f0 = (Float) vobj;

int acno = i0.intValue();

float bal = f0.floatValue();

Sop("bal + " is the bal of " + acno);

} || for

(Q) What are the approaches used for extracting the data from 2D CFW?

(A) We have two approaches to extract the data from 2D CFW variable.

(1) public Set entrySet()

(2) public Set keySet()

java.util.Map.Entry:

Map is an interface and Entry is one of the predefined class in Map interface.

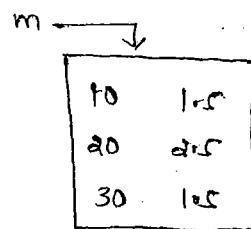
Map.Entry is used for separating key value and value of value from 2D CFW variable.

Methods

1. public Object getKey()

2. public Object getValue()

The above 2 methods are used for obtaining key value and value of value from any 2D CFW variable by type casting mobj of java.lang.Object



into `java.util.Map.Entry`.

### java.util.SortedMap:

- `SortedMap` is one of the sub interface of `Map`. So that all the methods of `Map` are inherited into `SortedMap`.
- An object of `SortedMap` also organizes the data in the form of `(key,value)`.
- An object of `SortedMap` always displays the data in sorted order. and sorting can be done based on key value.
- The real-time implementation of `SortedMap` in wireless application development is phonebook of the mobiles.

(1) `public Object first()`

(2) `public Object last()`

(3) `public SortedMap headMap (Object obj1) →  $x_i <= obj_1$`

(4) `public SortedMap tailMap (Object obj2) →  $x_i > obj_2$`

(5) `public SortedMap subMap (Object obj1, Object obj2) →  $obj_1 \leq x_i < obj_2$`

(See the explanation of `SortedSet`)

$\downarrow Sm$

Ex:  $\text{Object } obj_1 = Sm.\text{first}();$

$$10 = 10.5$$

10	10.5
20	2.5
30	3.5
40	4.5
50	5.5
60	6.5
70	7.5
80	8.5

`SortedMap Sm1 = Sm.headMap (new Integer(70));`

`Integer(70));`

10	10.5
20	2.5
30	3.5

`SortedMap Sm2 = Sm.subMap (new Integer(40), new Integer(70));`

40	4.5
50	5.5
60	6.5

`Object obj2 = Sm2.last();`

$$80 = 8.5$$

`SortedMap Sm3 = Sm.tailMap (new Integer(60));`

70	7.5
80	8.5

## Q1) Name CFW classes or Map classes:

Q1) QFW classes contains definition for those abstract methods which are inherited from QFW interfaces.

All QFW classes are present in a predefined class called `java.util`.

The following table gives QFW interface name, whose corresponding class name and their hierarchy.

<u>QFW Interfaces</u>	<u>QFW classes</u>	<u>Hierarchy</u>
Map	① AbstractMap	Map
Map.Entry	—	—
SortedMap	② AbstractSortedMap	AbstractMap { ↳ extends AbstractMap ↳ implements SortedMap}
	③ HashMap	AbstractMap
	④ TreeMap	AbstractSortedMap

The classes ① and ② are abstract classes so that whose objects can't be created directly and we never use them directly in real-time applications and more over these classes are playing a very important background role in defining abstract methods of QFW interfaces. In other words these classes are playing the middle man role b/w QFW interface names and QFW concrete classes.

The classes ③ and ④ are the bottom most concrete subclasses of all QFW abstract classes. Hence the classes ③ & ④ objects can be created directly and we use them in real-time applications.

### HashMap and TreeMap:

The objects of HashMap and TreeMap never allows duplicate values for the keys.

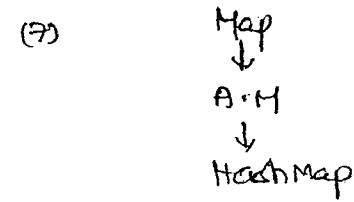
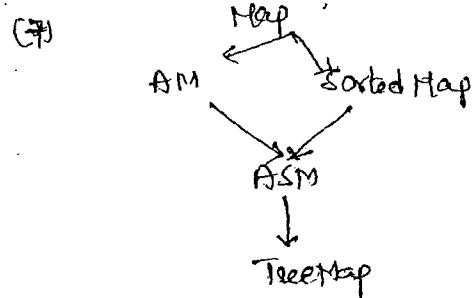
#### HashMap

- (1) An obj of HashMap organizes the data in the form of (key, value) by following hashing mechanism | (2) An obj of TreeMap organizes the data in the form of (key, value) by following binary trees concept

#### TreeMap

HashmapTreeMap

- (1) We can't determine in which order Hashmap obj displays the data bcoz SunMicrosystems did not disclose which hashing mechanism is followed.
- (2) An obj of TreeMap always displays the data in sorted order.
- (3) The operations like insertion, deletion, modification takes considerable amount of time i.e quite expensive.
- (3) The operations like insertion, deletion, modification takes negligible amount of time i.e less expensive
- (4) Retrieving the data from HashMap takes considerable amount of time i.e. more time.
- (4) Retrieving the data from TreeMap takes negligible amount of time i.e less time.
- (5) Creating HashMap is nothing but creating an object of HashMap class
- (5) Creating a TreeMap is nothing but creating an object of TreeMap class.
- Ex: `HashMap hm = new HashMap();`
- Ex: `TreeMap tm = new TreeMap();`

(6) `HashMap()`(6) `TreeMap()`

→ Write a Java program which illustrate the concept of HashMap & TreeMap

1) `hmtn.java`

```

import java.util.*;
class hmtn
{
    public static void main(String[] args)
    {
        // 1) HashMap hm = new HashMap();
        TreeMap hm = new TreeMap();
        System.out.println("Size" + hm.size());
    }
}
  
```

```

Sop ("Content = " + hm); // { }

// Adding the data
hm.put (new Integer(100), new Float(1.5f));
hm.put (new Integer(1), new Float(2.5f));
hm.put (new Integer(90), new Float(1.5f));
hm.put (new Integer(40), new Float(4.5f));
hm.put (new Integer(100), new Float(9.5f)); // duplicate entries
                                                // are not allowed

```

Sop ("Size = " + hm.size()); // 4

Sop ("Content = " + hm); // { ... }

// extracting the data - from 2D CFW variable.

Sop ("Extracting the data - from 2D CFW var → entrySet() → Set →  
Iterator()");

Set s = hm.entrySet();

Iterator ite = s.iterator();

while (ite.hasNext())

{

Object mobj = ite.next();

Map.Entry me = (Map.Entry) mobj; // OTC

Object kobj = me.getKey();

Object vobj = me.getValue();

Integer i0 = (Integer) kobj;

Float fo = (Float) vobj;

int ac no = i0.intValue();

float bal = fo.floatValue();

Sop ("bal+" is the bal of "acno");

}

Sop ("Extracting the data - from 2D CFW var → keySet() → Set →  
toArray()");

Set s1 = hm.keySet();

Object kobj[] = s1.toArray();

```

for (int i=0; i<kobj.length; i++)
{
    Object vobj1 = hm.get(kobj1[i]);
    Integer iobj1 = (Integer) kobj1[i];
    Float fobj1 = (Float) vobj1;
    int acnol = iobj1.intValue();
    float bal1 = fobj1.floatValue();
    System.out.println(bal1 + " → " + acnol);
}

```

## Legacy Collection Framework:

- In the older versions of java the concept of Collection Framework was known as Data Structures. The concept of DS was meeting some of the demands of the industry and unable to fulfil some other requirements of the industry. Hence SUN MICRO systems had revised or re-engineered the concept of DS and released to the industry on the name of new CFW and the existing DS concept was renamed as Legacy CFW.

- The basic difference b/w new CFW and Legacy CFW is that all the classes and interfaces of new CFW are belongs to non-synchronized and they provides concurrent access. All the classes and interfaces of legacy CFW are belongs to synchronized and they provides sequential access.

- (1) # 1D Legacy CFW

- ## (2) 2D Legacy CFW

- Dealing with 1D and 2D of any Legacy CFW is nothing but learning about their interfaces and classes.

- (1) `java.util.Enumeration` → Interface

- `java.util.Vector` ↳ 2D List API

- ### 3) java.util.Stack

(4) `java.util.Dictionary`  
 (5) `java.util.HashTable`  
 (6) `java.util.Properties`

} 2D L.CFW

### java.util Enumeration:

It is one of the predefined interface whose object is always used for extracting or retrieving the data from Legacy CFW classes in forward direction only. An object of Enumeration is by default pointing just before the 1st element of any Legacy CFW variable.

The functionality of Enumeration is exactly similar to Iterator but Enumeration retrieves the data sequentially and Iterator retrieves the data concurrently.

#### Methods:

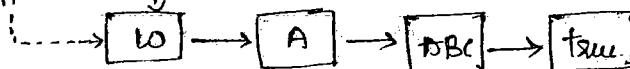
(1) `public boolean hasMoreElements();`

(2) `public Object nextElement();`

Method (1) returns true provided Enumeration Interface object is having more elements in Legacy CFW variable otherwise it returns false.

Method (2) is used for obtaining an element of Legacy CFW variable until method (1) returns false.

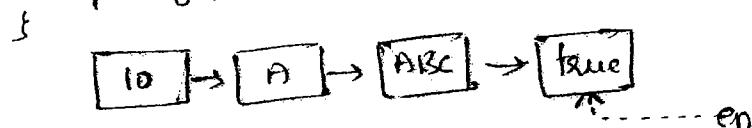
Ex: en      ; L.CFW variable



while (`en.hasMoreElements()`)

{  
    Object obj = en.nextElement();

    Sop(obj); #



### java.util.Vector:

1. Vector is one of the concrete class presents in 1D LCFW.
2. The functionality of Vector is exactly similar to ArrayList but Vector data can be accessed in Sequential and ArrayLists data.

Can be accessed in concurrent.

(3) In Vector the data is organizing in the form of Cells. Cell values are stored in heap memory and Cell addresses are stored in associative memory.

(4) Creating a vector is nothing but creating an object of Vector class.

Ex: Vector v = new Vector();

(5) The default capacity of the vector obj is 10 cells.

#### Constructors

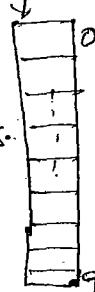
(3) public int size();

(1) Vector()

(4) public int capacity();

(2) Vector(int size)

(5) public void addElement(Object)



(6) public void addElement(int, Object)

(7) public Object removeElementAt(int)

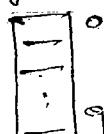
(8) public void removeElement(Object)

(9) public void setElementAt(int, Object)

(10) public Enumeration elements()

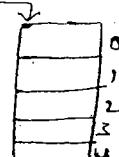
→ Constructor(1) is used for creating an object of Vector class without specifying the explicit size.

Ex: Vector v = new Vector();



→ Constructor(2) is used for creating an object of Vector by specifying the explicit size.

Ex: Vector v = new Vector(5);



→ Method(3) is used for finding the no. of elements present in the Vector.

→ Method(4) will give the total no. of Cells available in the Vector.

Ex: Vector v = new Vector();

System.out.println(v.size()); // 0

System.out.println(v.capacity()); // 10



- Method (6) is used for adding an element to the vector.
- Method (6) is used for adding an element to the vector at the position.

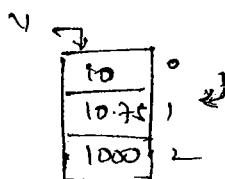
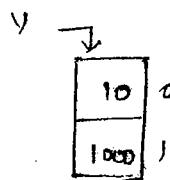
Ex: Vector v = new Vector();

```
v.addElement(new Integer(10));
v.addElement(new Integer(1000));
```

Sop(v); // [10, 1000]

v.addElement(1, new Float(10.75f));

Sop(v); // [10, 10.75, 1000]



All the elements of vector can be processed based on 0-based indexing.

### Concept

- Methods (7) and (8) are used for removing the elements from the vector either based on the position or based on the content.

Ex: Object obj = v.removeElementAt(1); → based on position

Sop(obj); // 10.75

// Sop('Removed element' + v.removeElement(new Float(10.75f)));  
↓ based on content

- Method (9) is used for replacing an element into the vector by specifying its position.

Ex: v.setElementAt(1, new Float(100.75f));

Sop(v); // [10, 100.75, 1000]  
↑

- Method (10) is used for extracting the data from 1D LCFW variable in forward direction.

Ex: Sop(v); [10, ABC, 1000]

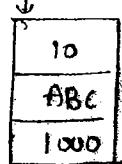
Enumeration en = v.elements();

en → [10] → [ABC] → [1000]

while (en.hasMoreElements())

{ Object obj = en.nextElement(); }

} Sop(obj); // prints all the elements in forward direction.



⇒ Write a program which illustrates the concept of Vector class.

```
/* Vect.java
import java.util.*;
class Vect
{
    public static void main(String[] args)
    {
        Vector v = new Vector();    System.out.println("Content = " + v); // []
        System.out.println("Capacity = " + v.capacity()); // 0
        System.out.println("Size = " + v.size()); // 0
        // adding the data.
        v.addElement(new Integer(10));
        v.addElement(new Float(10.75f));
        v.addElement(new Character('A'));
        v.addElement("Sathya");
        System.out.println("Content = " + v); // [10, 10.75, A, Sathya]
        System.out.println("Size = " + v.size()); // 4
    }
}
```

// extracting the data

```
Enumeration en = v.elements();
while(en.hasMoreElements())
{
    Object obj = en.nextElement();
    System.out.println(obj);
}
```

} // main

} // Vect-class

### java.util.Stack:

→ Stack is one of the 2D legacy CFW class.

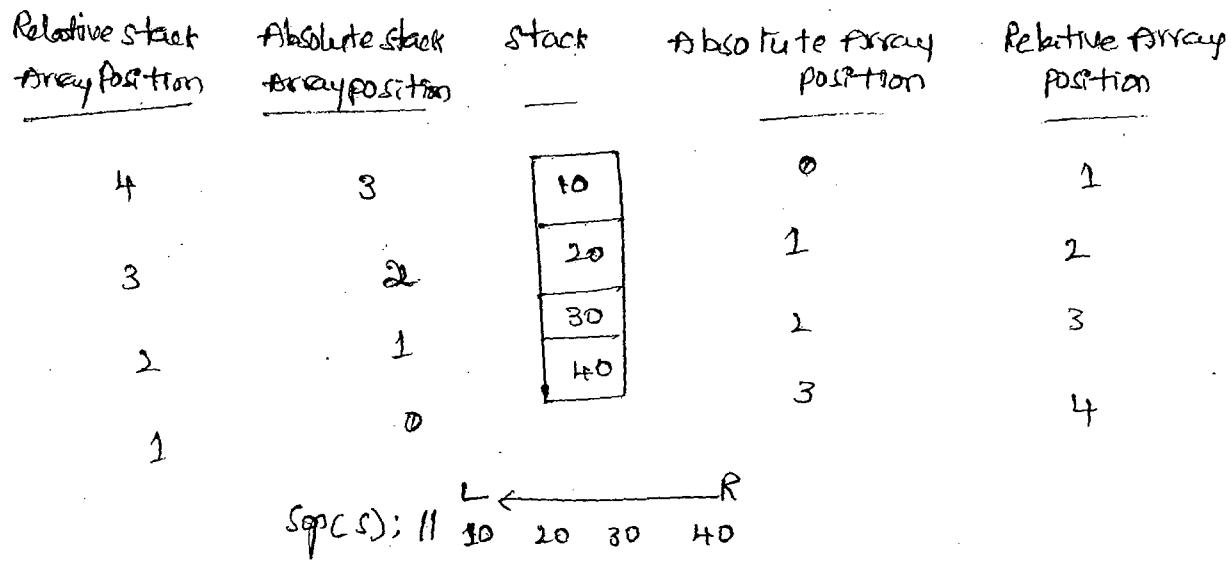
→ The class Stack is the sub class of Vector so that all the methods of Vector are inherited into Stack.

→ The basic working principle of Stack is LIFO (Last In First Out) i.e. the element which is inserted last, is that element will be taken out first.

Memory wrt obj of Stack class.

Ex: Stack s = new Stack();

- The elements of the stacks are processed and stored and displayed with right to left associativity.
- The diagrammatic representation of stack class object in the heap memory is given below.



### Stack class profiles

#### Constructor

(1) Stack();

#### Instance Methods

- (2) public boolean empty();
- (3) public void push(Object);
- (4) public Object pop();
- (5) public Object peek();
- (6) public int search(Object);

→ Constructor(1) is used for creating an object of stack.

Ex: Stack s = new Stack();

→ Method(2) returns true provided stack is not containing any elements.  
it returns false provided stack contains some elements.

→ Method(3) is used for inserting an element into the stack.

→ Method(4) is used for removing topmost element of the stack permanently (recent absolute stack array position element)

- Method (5) is used for retrieving or obtaining the top most element of the stack.
- Method (6) is used for searching an element into the stack. This method return relative stack array position provided the search is successful. It returns -1 provided search is unsuccessful.
- Write a Java program which illustrate the concept of Stack.

/\* Stack.java \*/

```

import java.util.*;
class Stack
{
    public static void main (String[] args)
    {
        Stack s = new Stack();
        System.out.println ("Content = " + s); // []
        System.out.println ("Size = " + s.size()); // 0
        System.out.println ("Is 'empty' = " + s.empty()); // True.
        // Add the data to the stack
        s.push (new Integer (10));
        s.push (new Integer (20));
        s.push (new Integer (30));
        s.push (new Integer (40));
        System.out.println ("Content = " + s); // [10, 20, 30, 40]
        System.out.println ("Size = " + s.size()); // 4
        System.out.println ("Is 'empty' ? = " + s.empty()); // false
        System.out.println ("Deleted element = " + s.pop()); // 40
        System.out.println ("Content after deletion = " + s); // [10, 20, 30]
        System.out.println ("Top most element = " + s.peek()); // 30
        System.out.println ("Content after peek = " + s); // [10, 20, 30]
        System.out.println ("Is 10 found in = " + s.search (new Integer (10))); // 1
        System.out.println ("Is 100 found in = " + s.search (new Integer (100))); // -1
    }
}

```

