Core Java Class Notes

By Srinivasulu Reddy K

What Is Java?

- Java is a computer programming language.
- It enables programmers to write computer instructions using English based commands, instead of having to write in numeric codes.
- It's known as a "high-level" language because it can be read and written easily by humans.
- Like English, Java has a set of rules that determine how the instructions are written.
- These rules are known as its "syntax". Once a program has been written, the high-level instructions are translated into numeric codes that computers can understand and execute.

Who Created Java?

In the early nineties, Java was created by a team led by James Gosling for Sun Microsystems. It was originally designed for use on digital mobile devices, such as cell phones. However, when Java 1.0 was released to the public in 1996, its main focus had shifted to use on the Internet. It provided more interactivity with users by giving developers a way to produce animated webpages. Over the years it has evolved as a successful language for use both on and off the Internet.

History:

- In 1990, Sun Microsystems began an internal project known as the Green Project to work on a new technology.
- In 1992, the Green Project was spun off and its interest directed toward building highly interactive devices for the cable TV industry. This failed to materialize.
- In 1994, the focus of the original team was re-targeted, this time to the use of Internet technology. A small web browser called Hot Java was written. Oak was renamed to Java after learning that Oak had already been trademarked.
- In 1995, Java was first publicly released.
- James Gosling is generally credited as the inventor of the Java programming language
- He was the first designer of Java and implemented its original compiler and virtual machine
- JDK(Java Development Kit) 1.0, code named Oak and released on January 23, 1996.
- JDK 1.1, released on February 19, 1997.
- JDK 1.2, code named Playground and released on December 8, 1998.
- JDK 1.3, code named Kestrel and released on May 8, 2000.
- JDK 1.4, code named Merlin and released on February 6, 2002 (first release under JCP).
- JDK 1.5, code named Tiger and released on September 30, 2004.
- JDK 1.6, code named Mustang and released on December 11, 2006.
- JDK 1.7, code named Dolphin and released on July 28, 2011, the most widely used version.
- JDK 1.8, was released on 18 March 2014. The code name culture is dropped with Java 8 and so no official code name going forward from Java 8., the latest version.

List the three Java technology product groups:

A Java Platform is the set of APIs, class libraries, and other programs used in developing Java programs for specific applications

There are 3 Java Platform Editions

1. Java 2 Platform, Standard Edition (**J2SE**)

   Core Java Platform targeting applications running on workstations

2. Java 2 Platform, Enterprise Edition (**J2EE**)

   Component-based approach to developing distributed, multi-tier enterprise applications

3. Java 2 Platform, Micro Edition (**J2ME**)

Targeted at small, stand-alone or connectable consumer and embedded devices

Features of Java:

Services provided by the java language to the industry programmers

1) Simple
2) Platform independent
3) Architectural Neutral
4) Portal
5) Multi-Threading
6) Network
7) Distributed
8) High performance
9) Robust
10) Secured
11) Dynamic
12) Interpreted
13) Object Oriented Programming Language

1) Simple:

JAVA is simple because of the following factors:

- JAVA is free from pointers hence we can achieve less development time and less execution time [whenever we write a JAVA program we write without pointers and internally it is converted into the equivalent pointer program].
- Rich set of API (application protocol interface) is available to develop any complex application.
- The software JAVA contains a program called garbage collector which is always used to collect unreferenced (unused) memory location for improving performance of a JAVA program. [Garbage collector is the system JAVA program which runs in the background along with regular JAVA program to collect unreferenced memory locations by running at periodical interval of times for improving performance of JAVA applications.
- JAVA contains user friendly syntax's for developing JAVA applications.

2) Platform independent:

Data types of of the language must take same amount of memory space on all available operating systems.
The S/W must has some special program to convert one format to another OS formats

3) Architectural Neutral:

The applications run on every processor architecture without considering their architecture and vendor.

32 -bit
64 -bit

ABC --> 1001100101 - 32 bit
ABC --> 1011101101 - 64 bit

4) Portable:

Portable = Platform Independent + Architecture Neutral

5) Multi-Threading:

A flow of control is known as thread.
We have created Test class --> one flow (Foreground/Child Thread)
Inbuilt Garbage Collector --> 2nd Flow (Background/ Parent Thread)

Process and Program

Program	Process
A program is set of optimized instructions	A program under execution is called as process.
Program always resides in secondary memory(Hard Disk)	It always resides in main memory of the system(RAM memory)
It resides long time until we delete it	It resides limited time, until system reboot or execution completes
it treated as static object	Its dynamic object

<u>6) Networked :</u>
Collection of interconnected autonomous / non autonomous computers is called network
Sharing data among machines either locally or globally.
2 types
1) Un trusted
2) Trusted

<u>7) Distributed :</u>
We can create distributed applications in java.
RMI and EJB are used for creating distributed applications.
We may access files by calling the methods from any machine on the internet.

<u>8) High performance:</u>
Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

<u>9) Robust:</u>
Robust simply means strong.
Java uses strong memory management.
There are lack of pointers that avoids security problem.
There is automatic garbage collection in java.
There is exception handling and type checking mechanism in java.
All these points makes java robust.

<u>10) Secured:</u>
Java is secured because:
No explicit pointer
Programs run inside virtual machine sandbox.
Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.
Byte code Verifier- checks the code fragments for illegal code that can violate access right to objects.
Security Manager- determines what resources a class can access such as reading and writing to the local disk.

<u>11) Interpreted:</u>
.java --> .class(byte code) --> interpreted

<u>12) Dynamic:</u>
WORA - Write Once Run Anywhere

<u>13) Object Oriented Programming Language:</u>
To put it simply, object-oriented programming focuses on data before anything else.

How data is modeled and manipulated through the use of objects is fundamental to any object-oriented program.

What Are Objects?

If you look around you, you will see objects everywhere.
Perhaps right now you are drinking coffee.
The coffee mug is an object, the coffee inside the mug is an object, even the coaster it's sitting on is one too.
Object-oriented programming realizes that if we're building an application it's likely that we will be trying to represent the real world.
This can be done by using objects.

Class?

The data an object holds and how it manipulates that data is specified through the creation of a class. For example, below is a very basic definition of a class

```
class HelloWorld
{
//
}
```

Java tokens:

A java program is collection of classes. A class is defined by a set of declaration statements and methods containing executable statements.
Smallest individual units of program are known as tokens.
There are five types of tokens:

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

Reserved key words in java:

1) In java we have **53**
   Reserved literals(3)
   Keywords (50)
      Un Used keyword - goto, const
      Used - 48 (if, else, while, for, int double, float char .. etc)

keywords for data types (8):

byte
short
int
long
float
double
char
boolean

Keywords for flow Control(10):

- if
- else
- switch
- case
- default
- for
- while
- do
- break
- continue

Keywords for modifiers (11):

- public
- private
- protected
- static
- final
- abstract
- strictfp(out dated)
- native
- synchronized
- volatile
- transient

Keywords for Exception handling(6):

- try
- catch
- finally
- throw
- throws
- assert(at time writing junit test cases)

Class related keywords (6):

- Class
- interface
- package
- extends
- implements
- import

Object related keywords (4):

- new
- instanceof
- super
- this

Return type keywords(2):

void
return

un used (2):

goto
const

Reserved Literals(3) :

true
false
null

- **enum** keywords

```
enum month
{
        JAN, FEB, MAR.. DEC;
}
```

Total Keywords : - 8+11+11+6+6+4+1 = 47 keywords
Un used Keyword : - 2
Reserved Literals : - 3
Enum Keyword : 1

Total Keywords : 53

Q) Which of the following list contains only java reserved/keywords words
x1) new, delete(its a identifier)
x2) break, continue, goto, construct(construct is identifier)
x3) extends, implements, imports(imports is invalid, import is valid)
x4) final, finally, finalize(finalize is a predefined method)
x5) break, continue, exit(exit is predefined method)
x6) synchronized, extends, volatile, virtual(virtual c++)
x7) instanceOf, strictfp
8) none of the above

A) break, continue, implement
B) extends, implements, imports
C)final, finally, finalize
D)instanceOf, strictfp
E) none of the above

Q) Which of the following are valid java reserved/keywords words

public	- v(valid)
main	- iv(invalid)
static	- v
args	- iv
void	- v
String	- in

Identifiers:

```
class Test {
	public static void main(String args[]) {
		int x = 0;
	}
}
```

Rules :

1) a to z. A to Z, 0-9, _,$
    - ca$h - valid
    - total# - invalid
2) identifier is not allowed to start with digit
    - total123 - valid
    - 123total - invalid
3) Identifiers are case sensitive
    - number - valid
    - Number - valid

4) There is no length limit for identifiers
5) Good Programming practice is use 15 characters of length
6) Reserve words are not used as identifiers
    - int i =1; valid
    - int if = 2; invalid
7) All predefined java class names and interface names we can use as identifiers
    - String
    - Runnable
    - int String = 1; Valid
    - java.lang.String name = "kummitha"; Valid
    - int Runnable = 3; valid

    1) -$- (Invalid)
    2) _$_ (Valid)
    3) ca$h (valid)
    4) int (Invalid)
    5) Integer (valid) its predefined wrapper class

Literals:
Literals in java are group of characters (digits, letters, and underscore) that represent constant values to be stored in variables.

The literals are:

- Integer literals
- Floating-point literals
- Character literals
- String literals
- Boolean literals

Separators:
Separators are symbols used to indicate where groups of code are divided and arranged.

Name	What is used for

Parentheses()	. Used to enclose parameters in method definition and invocation. . Used for defining precedence in expressions. . Used for Type casting.
Braces {}	. Used for initialization of arrays . Used to define a block of code for classes, methods and local scopes.
Brackets []	. Used for declaring array types. . Used for dereferencing array values.
Semicolon ;	. Used to separate statements.
Comma ,	. Used to separate consecutive identifiers in variable declaration . Used to write expression together inside a for loop.
Dot .	. Used to separate package names from sub-packages and classes. . Used to separate a variable or method from a reference variable.

Data types in java:

Data types are basically used for representing the input of the program in the main memory of the computer by allocating sufficient amount of memory space.
In any programming language we have 3 types of data types.

1) Fundamental/ predefined / primitive Data types
2) Derived Data types
3) User / Programmer / Custom defined data types

1) Fundamental/ predefined / primitive Data types:

Fundamental data types are those who's variables allows to represent only one value but they are unable to represent multiple values of same type.
Ex: int, float, char, boolean

2) Derived Data types:

Derived data types are those who's variables allows to represent multiple values of same type but they never allows to represent multiple values of different type.
ex: arrays

3) User defined data types:

User defined data types are those which are developed by programmers and who's variables are allows us to represent either same type of values are different type of values.
ex: All class and interfaces (either predefined or user defined comes under user defined data types);

Fundamental Data types in java:

In java programming we have 8 fundamental data types whoch are organized into 4 groups they are

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

1) Integer category data types:-
=====================

Integer category data types are used for representing integer data(which does not allows decimal places)
data types available in integer category,

Data type	size(bytes)	Range
byte	1	+127 to -128
short	2	+x to -(x+1)

int	4	+x to -(x+1)
long	8	+x to -(x+1)

Note:-
The formula for calculating range of any data type in any language is

$$\text{Range of any data type} = \left(\begin{array}{l}\text{No of bits available in the language}\\ \text{which is understand by the computer}\end{array}\right)^{\text{No bits occupied by the data type}}$$

Means binary data i.e (0,1) = 2

short:-

$$(2)^{16} = 65536$$

--> 1 to 65536
--> 0 to 65535/2
--> 32767.5
(+)32767 to (-)32768

2) Float category:

Float category data types are used for representing the data in the form of scale, precision i.e., these category data types are used for representing float values. This category contains two data types; they are given in the following table:

Data Type	Size(bytes)	Range	Number of decimal Places
float	4	+2147483647 to -2147483648	8
double	8	$\pm 9.223*10^{18}$	16

Whenever we take any decimal constant directly in a JAVA program it is by default treated as highest data type in float category i.e., double.

3. Character category data types:

A character is an identifier which is enclosed within single quotes.
In JAVA to represent character data, we use a data type called char. This data type takes two bytes since it follows UNICODE character set.

$$\begin{tikzcd}
 & \text{Char} \arrow[dl] \arrow[dr] & \\
\text{c/c++} \arrow[d] & & \text{Java} \arrow[d] \\
\text{1 byte} \arrow[d] & & \text{2 bytes} \arrow[d] \\
\text{English} \arrow[d] & & \text{i18n (18 International Languages)} \arrow[d] \\
\text{ASCII} & & \text{UNICODE}
\end{tikzcd}$$

Data Type	Size(bytes)	Range

char	2	+32767 to -32678

JAVA is available in 18 international languages and it is following UNICODE character set.
UNICODE character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters.

4. Boolean category data types:

- Boolean category data type is used for representing logical values i.e., TRUE or FALSE values.
- To represent logical values we use a keyword called Boolean.
- This data type takes 0 bytes of memory space.

NOTE: All keywords in JAVA must be written in small letters only.

Variables:

what is variable?
A variable is an identifier who's value can't be changed during program execution

Rules :
1) 1st Letter alphabet
2) length <=32
3) _
4) no keywords

Declaration of variables:

Syntax

```
        datatype v1,v2,..vn;
        here
ex:
        int a;
        int a,b,c;
        float x,y;
        char c1,c2;
        boolean b1, b2;

        Array[] ary;

        String s ;
        Sample s;
```

Declaration Cum initialization:
It is a process of placing our own values instead of placing default values.

Syntax:

```
datatype v1=value1,v2=value2,v3,.....vn=valuen;
```

Ex:

```
float PI=3.1417f;
int a=10, b=20;
```

Constants in java:

A constant is an identifier who’s value can't be changed during execution of the program.

In order to make anything as constant in java programming we use a keyword called "final"

1) "final" at variable level

If we don't want to change the value of variable in the program execution then that variable value must be made as constant using final variable.

syntax:

```
	final datatype v1=value1;

	final int a;
	a=100+1; // valid
	a=200; // invalid
```

2) "final" at method level

```
	final return type methodName(list of final parameters if any) {
		// block of stmts

	}

	final float simpleInterest(float p, float T, float R) {
		float calValue;
		calValue =(P*T*R)/100;

		return calValue;
	}
```

3) "final" at class level

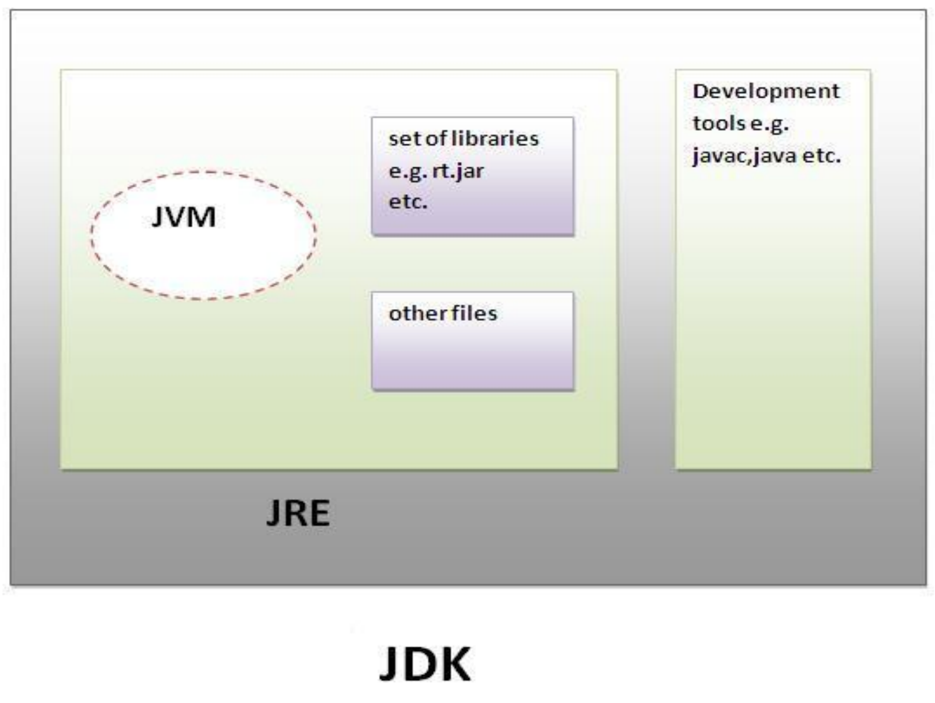
```
	final class <class Name>
	{
		variable declaration;
		methods def;
	}
```

<u>Java environment:</u>

Java environment includes a large number of development tools and hundreds of classes and methods.

The development tools are part of the java development kit (JDK).
The classes and methods are part of the java standard library (JSL).

Note: JSL is also known as Application programming interface (API), Java API is Just like library files of c and c++, library files contains number functions and java API collection of classes and methods.


set of libraries e.g. rt.jar etc.
JVM
other files
Development tools e.g. javac,java etc.
JRE
JDK

Java Development kit:
Java development kit comes with a collection of tools that are used for developing and running java applications.

javac (java compiler)
The java compiler, which translates java source into bytecode.

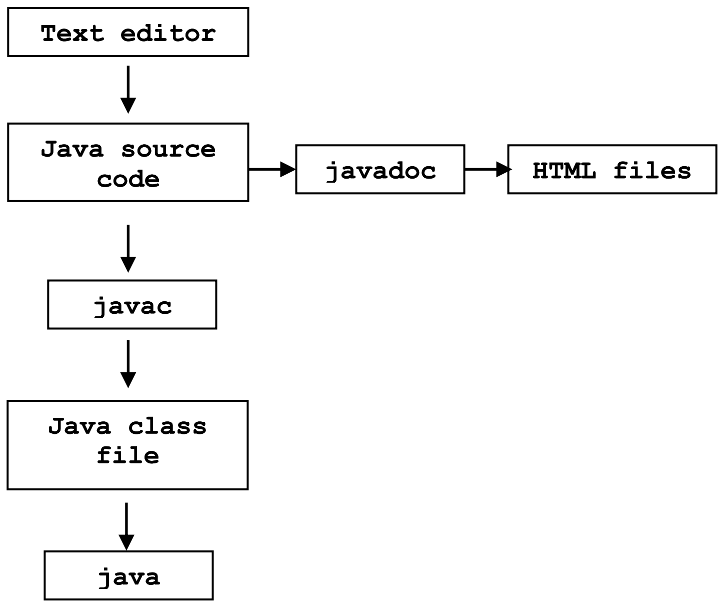
java (java interpreter)
The java interpreter which runs the applications by reading bytecode.

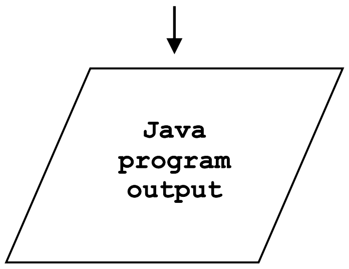
appletviewer
To run applets (without using web browser).

javap
To view API's description, and also converts bytecode into program description.

javadoc
To create HTML format documentation from java source code files.

The way these tools are applied to build applications programs is shown below.





$$\left.\begin{array}{l}\text{The java language}\\ \text{Class library(API/JSL)}\\ \text{Development tools}\end{array}\right\}\text{Java environment/Java development kit(JDK)}$$

<u>**Hungarian Notations**</u>:
Naming convention used by sun micro system for development of predefined classes and interfaces, predefined methods and data members(variables)

1) Hungarian rules for classes and interfaces:
If a class or interface name contains 1 or more words then all words 1st letter must be in capital.

Examples:

<u>Normal Notation</u>	<u>Hungarian notation</u>
system	System
printstream	PrintStream
arrayindexoutofboundsexception	ArrayIndexOutOfBoundException

2) Hungarian rules for methods:
If a method name is containing either one or more words then 1st letter is small and rest of the sub sequent words 1st letter must be capital.

Examples:

<u>Normal Notation</u>	<u>Hungarian notation</u>
println()	println()
ActionPerformed()	actionPerfomed()
getrailfalldata()	getRainFallData()
adjestmentvaluechanged()	adjestmentValueChanged()

3) Hungarian rules for data members:
If we use any free defined data members then all letters must be in capital.
If the data members contains more than one word then those words must be separated with _.

Examples:

<u>Normal Notation</u>	<u>Hungarian notation</u>
pi	PI
maxpriority	MAX_PRIORITY
segmentData	SEGMENT_DATA

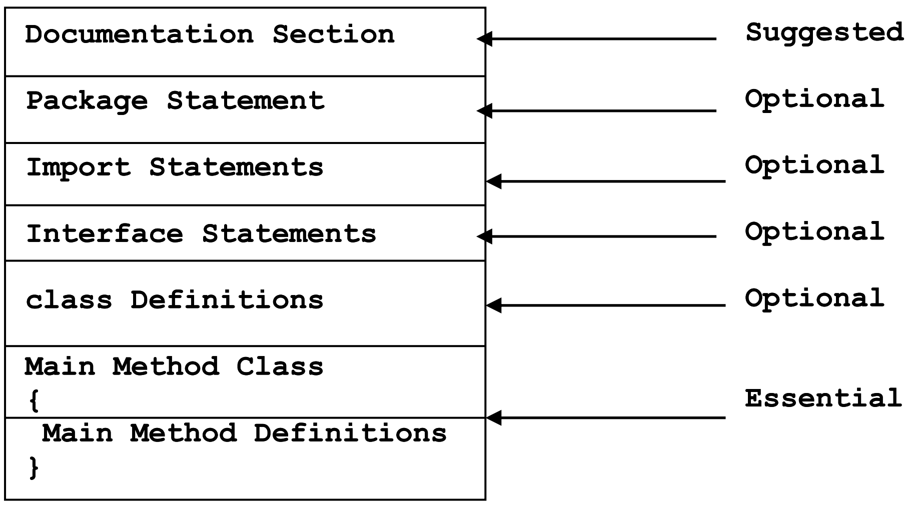
There are 2 types of JAVA programs can be developed.

1. Stand-alone applications
2. Applets/ Web applets

Stand-alone applications:
An application is a program that runs on your local stand-alone computer, under the operating system of that computer. Java application is like a program created using C and C++.
Ex: Adding of two numbers

Applets/Web applets:
An applet is an application designed to be transmitted over the Internet.
An Applet (small java program) located on distinct computer(Server) can be downloaded via Internet and executed on a local computer(Client) using JAVA Capable browser.
An applet cannot access the resources of the local computer.

<u>Java program structure:</u>
A java program may contain many classes, of which only one class should contains main() method.



Documentation section:
It consists of comment lines (program name, author, date,…….),
Ex:

```
/**
*
**/  known as documentation comment, which generated documentation automatically.
```

<u>Comments in java:</u>
Comments or any programming language makes us to understand the clarity or understandability of the program.
we have 3 types of comments in java
1) Multi line comment(from c) :
syntax :

```
/*
        comments
        comments
*/
```

ex:

```
/*
 @author skummitha
 This class is used for creating the resource in JSON data format.
 The @Path annotation identifies the URI path template to which the resource responds and
is specified at the class or method level of a resource.
```

```
                    */
```

2) Single line comment(from c++):
syntax :

```
                    // comment
                    ex:
                    // The Java class will be hosted at the URI path
```

3) Java Document comment(java):
javadoc comment
syntax :

```
          /**
                    comments
                    comments
          */
```

package statement:
This is the first statement in java file. This statement declares a package name and informs the compiler that the class defined here belong to this package.
Ex: - package student;

import statements:
This is the statement after the package statement. It is similar to # include in c/c++.
Ex: import java.lang.String

The above statement instructs the interpreter to load the String class from the lang package.

Note:
- import statement should be before the class definitions.
- A java file can contain N number of import statements.

interface statements:
An interface is like a class but includes a group of method declarations.
Note: Methods in interfaces are not defined just declared.

class definitions:
Java is a true oop, so classes are primary and essential elements of java program. A program can have multiple class definitions.

main method class:
Every stand-alone program required a main method as its starting point. As it is true oop the main method is kept in a class definition. Every stand-alone small java program should contain at least one class with main method definition.

Rules for writing JAVA programs:
1. Every statement ends with a semicolon
2. Source file name and class name must be the same.
3. It is a case-sensitive language.

4. Everything must be placed inside a class, this feature makes JAVA a true object oriented programming language.

Simple JAVA applications/programs:

Sequence: Executing all the statements one by one from Top to Bottom.

```
/** This is a simple Java program.
 Program name : HelloWorld.java */

class HelloWorld
{
         // program begins with a main()
        public static void main(String args[])
        {
               System.out.println("Hello World! Welcome to Core Java Online Training ");
        }
}
```

The name of the source file and name of the class name (which contains main method) must be the same, because in JAVA all code must reside inside a class.

Compiling the program:

```
C:> javac HelloWorld.java
```

Extension is compulsory at the time of compiling.

javac is a compiler, which creates a file called HelloWorld.class(contains the bytecode).

Note: when java source code is compiled each class is put into a separate .class file.

Executing the program:

```
C:> java HelloWorld
```

Java is interpreter which accepts .class file name as a command line argument.

That's why the name of the source code file and class name should be equal, which avoids the confusion.

Output:

```
Hello World! Welcome to Core Java Online Training
```

Important Note:

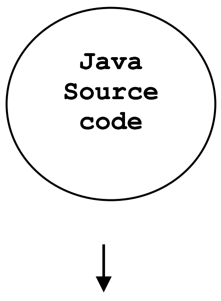
If the source code file name is Test.java and the class name is Hello, then to compile the program we have to give.

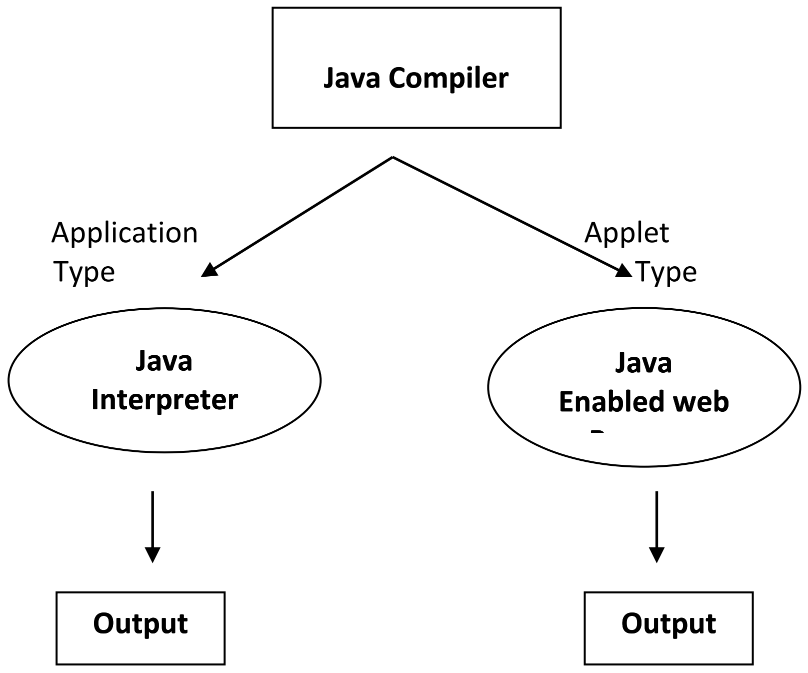
`C:> javac Test.java` (which creates a Hello.class file)

To execute the program

```
C:> java Hello
```

The following picture shows the execution of a java program/applet.





class HelloWorld:
The keyword class is used to declare a class and HelloWorld is the JAVA identifier ie name of the class.

public static void main(String args[])

public:
The keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes. The opposite of public is private.

static:
The keyword static allows main() to be called without creating any instance of that class.
This is necessary because main() is called by java interpreter before creating any object.

void:
The keyword void tells the compiler that main() does not return any value.

main():
main() is the method called when a java application begins.

String args[]:
Any information that you pass to a method is received by variables specified within the set of parenthesis that follow the name of the method. These variables are called parameters. Here args[] is the name of the parameter of string type.

No of ways to write a main method program
A) By changing sequence of the modifiers, method prototype is not changed.
```
ex : static public void main(String args[])
class HelloWorld {
      static public void main(String args[]) {
            System.out.println("Hello World- Modifiers");
      }
}
```

B) subscript notation in java array can be used after type, before variable or after variable.

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

class HelloWorld {
       static public void main(String[] args) {
              System.out.println("Hello World- Script Notation after args");
       }
}
```

C) You can provide var-args support to main method by passing 3 ellipses (dots)

```
public static void main(String... args)
class HelloWorld {
       static public void main(String... args) {
              System.out.println("Hello World- VAR -ARGS");
       }
}
```

D) Having semicolon at the end of class in java is optional.

Valid java main method signature

```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
public static void main(String... args)
static public void main(String[] args)
public static final void main(String[] args)
final public static void main(String[] args)
final strictfp public static void main(String[] args)
```

Invalid java main method signature

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

System.out.println:

`System.out.println()` --> displaying the data on the console line by line
`System.out.print()` --> displaying the data on the console in sample line
In the above 2 statements println() and print() methods are 2 predefined instance methods --> in predefined class "PrintStream"
PrintStream out = new PrintStream();
out.println();
out.print();

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

       class System {
              public static final java.io.PrintStream out
       }

       class PrintStream {
              public void println() {
              }
              public void print(){
        }
    }
```

Examples :

1) Only Message

```
    System.out.println("Hello World");
```

2) Only Data

```
    int a=10;
    int b=20;
    System.out.println(a);
    System.out.println(b);
```

3) Data + Message

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

```
class HelloWorld {
    public static void main(String args[]) {
        //Only Message
        System.out.println("Hello World");

        // Only Data
        int a=10,b=20;
        System.out.println(a);
        System.out.println(b);

        //Data+Message

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

Byte code:

The compiler converts the source code files into bytecode files. These codes are machine independent and therefore can be run on any machine. That is, a program compiled on IBM machine will run on Macintosh machine.
Java interpreter reads the bytecode files and translates them into machine code for the specific machine on which the java program is running.

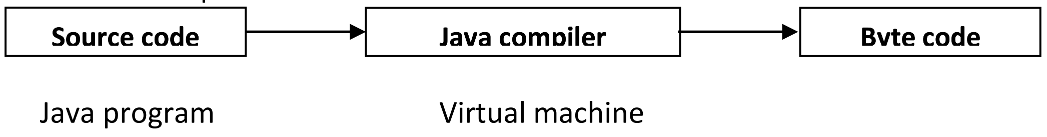
Java Virtual Machine:

Java compiler produces an intermediate code which is known as bytecode for a machine which does not exist. This machine is called the Java Virtual Machine and it exists only inside the computer memory. It is just like a computer with in the computer and does all major functions of a real computer.

To execute Java bytecode, the JVM uses a class loader to fetch the byte codes from a disk or from the network. Each class file is fed to a bytecode verifier that ensures the class is formatted correctly and that the class will not corrupt memory when it is executed.

The bytecode verification takes time to load a class, but it actually allows the program to run faster because the class verification is performed only once, but not throughout the program.

Process of compilation



The byte code/JVM code is not machine specific, machine code is generated by java interpreter.

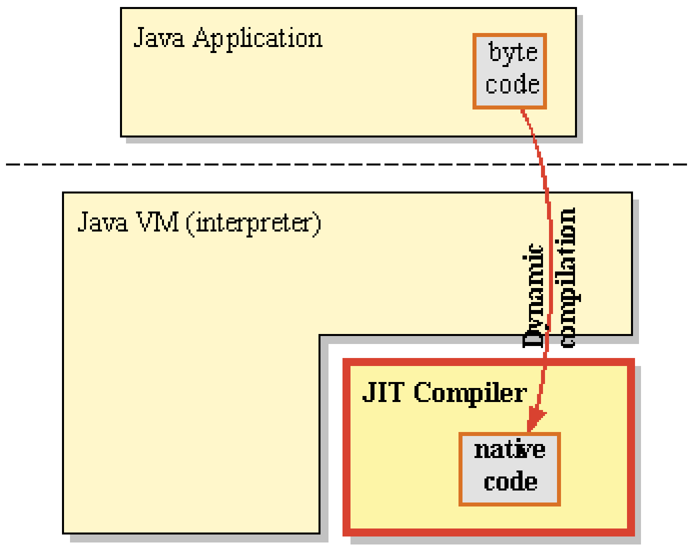
Process of Converting bytecode into machine code

$$\begin{tikzcd}\boxed{\texttt{Bytecode}} \arrow[r] & \boxed{\texttt{Java interperter}} \arrow[r] & \boxed{\texttt{Machine code}}\end{tikzcd}$$

Virtual machine   Real machine

Note: JVM is an interpreter for bytecode.
    ***Java interpreter is different from machine to machine

JIT Complier : Just in time compiler
At the time of executing Byte code it is again compiled by JIT (which is part of JVM). So it is known as Dynamic compilation.



Program to add two no's 10 and 20

```
/* Program name: Add2.java */
class Add2
{
        public static void main(String args[])
        {
                int a=10,b=20,c=0;
                c=a+b;
                System.out.println("The result is:"+c);
         }
}
```

Operators:
An operator is a symbol that takes two or more arguments and operates on them to produce a result.

The operators used in java are
1) Increment/Decrement
2) Assignment, Arithmetic, and Unary Operators
3) Equality, Relational, and Conditional Operators
4) Instanceof

5) Bitwise and Bit Shift Operators
6) Type cast
7) New
8)[] Operator
9) Operator precedence
10) Evaluation order of Java operands

1) Increment/Decrement :

Ex:-
```
	int x =10;
```
increment(++)
	pre increment
```
		int y = ++x;
```
	post increment
```
		int y = x++;
```

decrement(--)
```
	int x = 10;
```
	pre decrement
```
		int y = --x;
```
	post decrement
```
		int z = x--;
```

Expression	initial Value	final value of X	final value of y
y = ++x;	10	11	11
y = x++;	11	12	11
y = --x;	10	9	9
y = x--;	9	8	9

--> We can apply increment and decrement operators only for variables, but not for constant values.
--> We can not apply increment and decrement operators for the final variables.

Ex1:-

```
class IncreOperator1
{
	public static void main(String[] args)
	{
		int x =10;
		int y = ++x;
		System.out.println(" Y value = "+y);
	}
}
```

Ex2:-

```
class IncreOperator2
{
	public static void main(String[] args)
	{
		int x =10;
		int y = ++10;
		System.out.println(" Y value = "+y);
	}
}
```

```
CE: unexpected type
    int y = ++10;
   required: variable
  found:    value
```

Ex3:-

```
class IncreOperator3
{
	public static void main(String[] args)
	{
		final int x =10;
		int y = ++x;
		System.out.println(" Y value = "+y);
	}
}
```

```
CE: cannot assign a value to final variable x
```

--> Nesting of incement and decrement operators are not allowed otherwise we will get compile time error.

Ex3:-

```
class IncreOperator2
{
	public static void main(String[] args)
	{
		int x =10;
		int y = ++(++x);
		System.out.println(" Y value = "+y);
	}
}
```

```
CE: unexpected type
    int y = ++10;
   required: variable
  found:    value
```

--> We can apply increment and decrement operators for every "primitive"(byte, short, int, long, float, double, char, boolean) data types except "boolean"
https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html
http://www.scism.lsbu.ac.uk/jfl/Appa/appa4.html

Examples:
1) Valid
```
class IncreOperator3
{
	public static void main(String[] args)
	{
		double d1 =10.5;
		d1++;
		System.out.println(" Double value = "+d1);
	}
}
```
2) Valid
```
class IncreOperator3
{
	public static void main(String[] args)
	{
		char ch ='a'; // value of a = 97+1 = 98 = b
		ch++;
		System.out.println(" char value = "+ch);
	}
}
```

3) In valid

```
class IncreOperator3
{
	public static void main(String[] args)
	{
		boolean b = true;
		b++;
		System.out.println(" boolean value = "+b);
	}
}
```

CE: bad operand type boolean for unary operator '++'

Difference between x++ and x = x+1
==================================

When ever we are performing any arithematic operation between two variables x and y the result type is always max of that category data types

max(int, type of x, type of y)

Ex:
byte + byte = int
byte + short = int
int + long = long

long + float = float
double+char = double
char + char = int

Ex1:

```
class IncreOperator3
{
	public static void main(String[] args)
	{
		byte x = 10;
		byte y = 20;
		byte z = x+y;
		System.out.println(" Z value = "+z);
	}
}
```

```
CE:
 possible loss of precision
   d1 = d1+1;
        ^
  required: byte
  found:    int
```

Ex2:

```
class IncreOperator3
{
	public static void main(String[] args)
	{
		byte y = 20;
		y = y+1;
		System.out.println(" Y value = "+y);
	}
}
```

```
CE:
 possible loss of precision
   d1 = d1+1;
        ^
  required: byte
  found:    int
```

```
  class IncreOperator3
{
	public static void main(String[] args)
	{
		byte x = 10;
		byte y = 20;
		byte z = (byte)(x+y);
		System.out.println(" Z value = "+z);
	}
}
```

Note : In case of increment and decrement operators the required type casting automatically performed by the compiler.

2) Arithmetic operators:-
======================

+, *, -, /, %

The Java programming language provides operators that perform addition, subtraction, multiplication, and division. There's a good chance you'll recognize them by their counterparts in basic mathematics. The only symbol that might look new to you is "%", which divides one operand by another and returns the remainder as its result.

Operator	Description
+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

```
class ArithmeticDemo {

    public static void main (String[] args) {

        int result = 1 + 2;
        // result is now 3
        System.out.println("1 + 2 = " + result);
        int original_result = result;

        result = result - 1;
        // result is now 2
        System.out.println(original_result + " - 1 = " + result);
        original_result = result;

        result = result * 2;
        // result is now 4
        System.out.println(original_result + " * 2 = " + result);
        original_result = result;

        result = result / 2;
        // result is now 2
        System.out.println(original_result + " / 2 = " + result);
        original_result = result;

        result = result + 8;
        // result is now 10
        System.out.println(original_result + " + 8 = " + result);
        original_result = result;

        result = result % 7;
        // result is now 3
        System.out.println(original_result + " % 7 = " + result);
    }
}
```

This program prints the following:

```
1 + 2 = 3
3 - 1 = 2
2 * 2 = 4
4 / 2 = 2
2 + 8 = 10
10 % 7 = 3
```

--> You can also combine the arithmetic operators with the simple assignment operator to create compound assignments. For example, x+=1; and x=x+1; both increment the value of x by 1.

--> The + operator can also be used for concatenating (joining) two strings together, as shown in the following ConcatDemo program:

```
class ConcatDemo {
    public static void main(String[] args){
        String firstString = "This is";
        String secondString = " a concatenated string.";
        String thirdString = firstString+secondString;
        System.out.println(thirdString);
    }
}
```

--> By the end of this program, the variable thirdString contains "This is a concatenated string.", which gets printed to standard output.

The Unary Operators:-
===================

The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

The following program, UnaryDemo, tests the unary operators:

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

        int result = +1;
        // result is now 1
        System.out.println(result);

        result--;
        // result is now 0
        System.out.println(result);

        result++;
        // result is now 1
        System.out.println(result);

        result = -result;
        // result is now -1
        System.out.println(result);

        boolean success = false;
        // false
        System.out.println(success);
        // true
        System.out.println(!success);
    }
}
```

The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code result++; and ++result; will both end in result being incremented by one. The only difference is that the prefix version (++result) evaluates to the incremented value, whereas the postfix version (result++) evaluates to the original value. If you are just performing a simple increment/decrement, it doesn't really matter which version you choose. But if you use this operator in part of a larger expression, the one that you choose may make a significant difference.

The following program, PrePostDemo, illustrates the prefix/postfix unary increment operator:

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);
    }
}
```

Equality, Relational, and Conditional Operators:-
==

The Equality and Relational Operators:-

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

Operator	Description
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

The following program, ComparisonDemo, tests the comparison operators:

```
class ComparisonDemo {

   public static void main(String[] args){
      int value1 = 1;
      int value2 = 2;
      if(value1 == value2)
         System.out.println("value1 == value2");
      if(value1 != value2)
         System.out.println("value1 != value2");
      if(value1 > value2)
         System.out.println("value1 > value2");
      if(value1 < value2)
         System.out.println("value1 < value2");
      if(value1 <= value2)
         System.out.println("value1 <= value2");
   }
}
```

Output:

```
value1 != value2
value1 <  value2
value1 <= value2
```

The Conditional Operators:-

The && and || operators perform Conditional-AND and Conditional-OR operations on two boolean expressions. These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.

Operator	Description
========	===========
&&	Conditional-AND
||	Conditional-OR

The following program, ConditionalDemo1, tests these operators:

```
class ConditionalDemo1 {

   public static void main(String[] args){
      int value1 = 1;
      int value2 = 2;
      if((value1 == 1) && (value2 == 2))
         System.out.println("value1 is 1 AND value2 is 2");
      if((value1 == 1) || (value2 == 1))
         System.out.println("value1 is 1 OR value2 is 1");
   }
}
```

Another conditional operator is ?:, which can be thought of as shorthand for an if-then-else statement (discussed in the Control Flow Statements section of this lesson). This operator is also known as the ternary operator because it uses three operands. In the following example, this operator should be read as: "If someCondition is true, assign the value of value1 to result. Otherwise, assign the value of value2 to result."

The following program, ConditionalDemo2, tests the ?: operator:

```
class ConditionalDemo2 {

   public static void main(String[] args){
      int value1 = 1;
      int value2 = 2;
      int result;
      boolean someCondition = true;
      result = someCondition ? value1 : value2;

      System.out.println(result);
   }
}
```

Because someCondition is true, this program prints "1" to the screen. Use the ?: operator instead of an if-then-else statement if it makes your code more readable; for example, when the expressions are compact and without side-effects (such as assignments).

The Type Comparison Operator instanceof:-
==
--> The instanceof operator compares an object to a specified type. You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.

--> By using this operator we can check whether the given object is of particular type or not.

```
        HelloWorld hello = new HelloWorld();
```

Systax:-

```
hello instanceof HelloWorld  // here we can write interface name also.

Here
        hello is reference type
        HelloWorld is class or interface.
```

Ex:

```
        ComparisonDemo cdemo = new ComparisonDemo();

        System.out.println(cdemo instanceof ComparisonDemo); // true
        System.out.println(cdemo instanceof HelloWorld); // false
```

The following program, InstanceofDemo, defines a parent class (named Parent), a simple interface (named MyInterface), and a child class (named Child) that inherits from the parent and implements the interface.

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

    Parent obj1 = new Parent();
    Parent obj2 = new Child();

    System.out.println("obj1 instanceof Parent: "
      + (obj1 instanceof Parent));
    System.out.println("obj1 instanceof Child: "
      + (obj1 instanceof Child));
    System.out.println("obj1 instanceof MyInterface: "
      + (obj1 instanceof MyInterface));
    System.out.println("obj2 instanceof Parent: "
      + (obj2 instanceof Parent));
    System.out.println("obj2 instanceof Child: "
      + (obj2 instanceof Child));
    System.out.println("obj2 instanceof MyInterface: "
      + (obj2 instanceof MyInterface));
  }
}

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}
```

Output:

```
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof MyInterface: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

Note:- When using the instanceof operator, keep in mind that null is not an instance of anything.

Bitwise and Bit Shift Operators:-
================================

The Java programming language also provides operators that perform bitwise and bit shift operations on integral types. The operators discussed in this section are less commonly used. Therefore, their coverage is brief; the intent is to simply make you aware that these operators exist.

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

00000000 --> 11111111
00110011 --> 11001100

The bitwise & operator performs a bitwise AND operation.
 & --> AND --> of both arguments are true then result is true.
The bitwise | operator performs a bitwise inclusive OR operation.
 | --> OR --> if at least one is true then the result is true.
The bitwise ^ operator performs a bitwise exclusive OR operation.
 ^ --> XOR --> if both arguments are different then result is true.

 ex:

```
System.out.println(true & false)      ;
System.out.println(true | false)      ;
System.out.println(true ^ false)      ;
```

The signed left shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.

The following program, BitDemo, uses the bitwise AND operator to print the number "2" to standard output.

```
class BitDemo {
  public static void main(String[] args) {
    int bitmask = 0x000F;
    int val = 0x2222;
    // prints "2"
    System.out.println(val & bitmask);
  }
}
```

Summary of Operators:-
=====================

The following quick reference summarizes the operators supported by the Java programming language.

Simple Assignment Operator:-

=	Simple assignment operator

Arithmetic Operators:-

+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

Unary Operators:-

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Equality and Relational Operators:-

==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Conditional Operators:-

&&	Conditional-AND
||	Conditional-OR
?:	Ternary (shorthand for if-then-else statement)

Type Comparison Operator:-
----

instanceof	Compares an object to a specified type

Bitwise and Bit Shift Operators:-

~	Unary bitwise complement

<<   Signed left shift
>>   Signed right shift
>>>   Unsigned right shift
&   Bitwise AND
^   Bitwise exclusive OR
| Bitwise inclusive OR

Type cast operator:-
==================
There are 2 types of type castings

    1) Implicit type casting

    2) Explicit type casting

```
int i =130;
byte j = (byte)i;
```

1) Implicit type casting:-

--> Compiler is responsible for this implicit type casting

```
byte j =120;
int i = i;
```

--> It is required whenever we are tracing smaller data type value to the bigger data type variable is also known as widening or upcasting.
--> There is no loss of information in implicit type casting.
--> The following are various possible implicit promotions

```
byte --> short -
                                                              -
                                                                       -> int --> long --> float --> double
                                                              -
                              char  -
```

```
byte b = 2;
double d= b;
```

Ex1:-

```
        double d =10; // Compiler converts int to double type automatically.
        sop(d);
```

ex2:-

```
        int i ='a'; // compiler converts char to int type automatically.
        sop(i);
```

2) Explicit type casting:-

--> Programmer is responsible for explicit type casting.
--> It is required whenever we are trying to assign bigger data type value to the smaller data type variable.
--> Also known as "Narrowing" or "Down Casting"
--> There may be a chance of loss of information.
--> The following are various possible implicit promotions

```
int i = 12345678;
001110001  = 120
01110001 = 120
int 4 bytes
```

110010011111010110011001 = 12345678;
--> 2(0)* 2(1)+2(1)*2(0)+2(3)*2(0)
byte - 1 byte
00111001 = 126;

```
byte <-- short <-
                                              -
                                                    - int <-- long <-- float <-- double
                                              -
                  char  <-
```

Ex1:-

```
int i = 130;
byte b = i;
```

CE:

Solution:-

```
 byte b = (byte)i;
 sop(b);
```

--> Whenever we are assigning bigger data type value to the small data type by explicit type casting, the most significant bits will be lost.

7) New:-
======
--> We can use "new" operator for the creation of object.
--> There is no delete operator in java because it is the responsibility of garbage collector(Object destruction).
ex: Object o = new Object();

8) [] Operator:-
==============
This operator can be used for declaring and constructing arrays.

9) Precedence of java operators:-
===============================
1. unary operators : [], x++, x--, ++x, --x, ~,!,new, <type>
2. Arithmetic Operator : *, /,%, +, -
3. Shift Operator : >>, >>>, <<
4. Comparison operators : <,<=,>,>=, instanceof
5. Equality operators : ==, !=
6. Bit wise operators : &, ^, |
7. short circuit operators : &&, ||
8. Conditional operators : ?:
9. Assignment operator: =, +=, *=,...

10) Evaluation order of operands:-
================================
There is no precedence order for the operands but before applying any operator all the operand will be evaluated from left to right.

BODMAS

Flow Control:

Flow control will describe the order in which the statements will be executed at runtime.
Three types of flow controls are there in java, they are

1) Selection statements
```
if - else
switch(x)
```
2) Iterative statements
```
while()
do - while
for()
for each loop - introduced in 1.5
```
3) Transfer statements
```
break
continue
return
try - catch-finally
assert
```

1) Selection statements:

Branching of conditions. Statements are executed depending upon the conditions given by the user.

A) if - else

syntax:

```
	if(b) {
		execute statements under "if" block, if b is true;
	} else {
		execute statements under "else" block, if b is false;
	}
```
Note : The argument (b) to the if statement is always boolean type.

Example 1:
```
class SelectionStatements
{
	public static void main(String[] args)
	{
		int x=10;
		if (x)
		{
			System.out.println("Hello");
		}
		else
		{
			System.out.println("Hi");
		}
	}
}
```

```
CE: Incompatible Types
	found : int
	required : boolean
```

Example2:
```
class SelectionStatements
{
	public static void main(String[] args)
	{
		int x=10;
		if (x=20)
		{
			System.out.println("Hello");
		}
		else
		{
			System.out.println("Hi");
		}
	}
}

CE: Incompatible Types
	found : int
	required : boolean
```

Example3:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		int x=10;
		if (x==20)
		{
			System.out.println("Hello");
		}
		else
		{
			System.out.println("Hi");
		}
	}
}
```

o/p : Hi

Example4:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b=false)
		{
			System.out.println("Hello");
		}
		else
		{
			System.out.println("Hi");
		}
	}
}
```

O/P:

Example5:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b == true)
		{
			System.out.println("Hello");
		}
		else
		{
			System.out.println("Hi");
		}
	}
}
```

➔ for the if statement else part is optional

Example6:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b)
		{
			System.out.println("Hello");
		}

	}
}
```

→ Curly braces ({}) are optional.
→ Without curly braces only one statement is allowed to take under if and that statement should not be declarative statement.

Example6:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b)
			System.out.println("Hello");
	}
}
```

Example7:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b)
			int x=10;
	}
}
```

Example8:

```
class SelectionStatements
{
	public static void main(String[] args)
	{
		boolean b = true;
		if (b)
		{
			int x=10;
```

```
                System.out.println(x);
                System.out.println("X Value = "+x);
                System.out.println(x+" is X Value");
        }
    }
}
```

→ if (true);
Note: ";" is a valid java statement which is also known as empty statement.

B) switch() Statement:
→ If several options are available then it is not recommended to use "if - else ". we should go for switch statement.

Syntax:-
```
    switch(x)
    {
        case 1 : action1;
                break;
        case 2 : action2;
                break;
        case 3 : action3;
                break;
        deafult :
                default action;
    }
```

→The only allowed argument types to the switch statement are byte, short, int, char & corresponding wrapper classes(Byte, Short, Integer, Character) plus enum
→ Curly braces are mandatory for "switch", remaining all other cases are optional.
→ Inside a switch both case: & default are optional.
Ex:
```
    class SwitchStatement
    {
        public static void main(String []args)
        {
            int x = 10;
            switch(x)
            {
            }
        }
    }
```

→ Every case label should be constant expression.
eX1:
```
    class SwitchStatement
    {
        public static void main(String []args)
        {
            int x = 10;
            int y = 5;
            switch(x)
            {
                case 10 :
                    System.out.println("I am from case 10");
                    break;
                case y :
                    System.out.println("I am from case y");
                    break;
            }
        }
```

```
	}
```

-- In valid
CE(Compile Time Error) : Constant Expression required.

→ If we declare y as the "final" then we won't get any compile time error.

eX2:

```
	class SwitchStatement
	{
		public static void main(String []args)
		{
			int x = 10;
			final int y = 5;
			switch(x)
			{
				case 10 :
					System.out.println("I am from case 10");
					break;
				case y :
					System.out.println("I am from case y");
					break;
			}
		}
	}
```

eX3:

```
	class SwitchStatement
	{
		public static void main(String []args)
		{
			int x = 10;
			int y = 5;
			//x = y;
			switch(x)
			{
				case 10 :
					System.out.println("I am from case 10");
					break;
				case 5:
					System.out.println("I am from case 5");
					break;
			}
		}
	}
```

→ Switch argument can be expression also. but resultant expression should be of "int" type.

eX4:

```
	class SwitchStatement
	{
		public static void main(String []args)
		{
			int x = 4;
			switch(x+1) // expression is valid
			{
				case 11 :
					System.out.println("abc");
					break;
				case 5:
					System.out.println("I am from case 5");
					break;
```

```
                }
            }
        }
```

→ As the case label, we can take expression also but it should be constant expression.

eX5:

```
        class SwitchStatement
        {
                public static void main(String []args)
                {
                        int x = 4;
                        switch(x)
                        {
                                case 2+2 :  // expression is allowed, but only const expression.
                                        System.out.println("abc");
                                        break;
                                case 5:
                                        System.out.println("I am from case 5");
                                        break;
                        }
                }
        }
```

→ Case label should be within the range of argument type.

eX6:

```
        class SwitchStatement
        {
                public static void main(String []args)
                {
                        byte a = 10;
                        switch(a)
                        {
                                case 10 :
                                        System.out.println("abc");
                                        break;
                                case 100:
                                        System.out.println("I am from case 100");
                                        break;
                                case 1000:
                                        System.out.println("I am from case 1000");
                                        break;
                        }
                }
        }
 CE: ------------
 found : int
 required: byte
```

eX7:

```
        class SwitchStatement
        {
                public static void main(String []args)
                {
                        byte a = 10;
                        switch(a+1)  //Switch  argument  can  be  expression  also.  but  resultant
expression should be of "int" type. Here "a+1" returns int value.
                        {
                                case 10 :
                                        System.out.println("abc");
                                        break;
                                case 100:
```

```
                        System.out.println("I am from case 100");
                        break;
                  case 1000:
                        System.out.println("I am from case 1000");
                        break;
                  default :
                        System.out.println("I am from default");
            }
      }
}
```

Valid : Because expression contain integral constant it is allowed more than range.

→ Duplicate case labels are not allowed.

eX8:

```
class SwitchStatement
{
      public static void main(String []args)
      {
            byte a = 10;
            final byte b = 10;
            switch(a)
            {
                  case 10 :
                        System.out.println("abc");
                        break;
                  case 100:
                        System.out.println("I am from case 100");
                        break;
                  case b:
                        System.out.println("I am from case 1000");
                        break;
            }
      }
}
CE: Duplicate case label
```

Because case 10 is already there, again we are trying to define case 10 with the help of constant b value.

Case Label:

1) It should be constant value.
2) It can be constant expression also.
3) It should be within the range of arguments.
4) Duplicate case labels are not allowed.
5) Within the switch every statement should be under case or default. i.e independent statements are not allowed . Violation leads to compile time error.

eX9:

```
class SwitchStatement
{
      public static void main(String []args)
      {
            byte a = 10;
            switch(a)
            {
                  System.out.println("Hello");
            }
      }
}
```

CE: case, default or "}" is expected.

6) With in the switch statements, we can take default any where, but recommended to take always as last case.
7) Within the switch statement, if any case is matched from that statement onwards all the remaining statements will be executed until some break or end of the switch.
This is called "fall-through inside switch".

eX10:
```
	class SwitchStatement
	{
		public static void main(String []args)
		{
			byte a = define own value here;
			switch(a)
			{
				case 0 :
					System.out.println("0");
				case 1 :
					System.out.println("1");
					break;
				case 2 :
					System.out.println("2");
				default :
					System.out.println("default");

			}
		}
	}
o/p:

if a = 0
=====
0
1

if a = 1
=====
1

if a = 2
=====
2
default

if a = 3
======
default
```

eX11:
```
	class SwitchStatement
	{
		public static void main(String []args)
		{
			byte a = define own value here;
			switch(a)
			{
				default :
					System.out.println("default");
				case 0 :
					System.out.println("0");
					break;
				case 1 :
					System.out.println("1");
```

```
                                case 2 :
                                        System.out.println("2");
                        }
                }
        }
```
o/p:

```
if a = 0
========
0

if a = 1
=======
1
2
if a = 2
=======
2
if a = 3
=======
default
0
```

II) Iterative statements:

Iterative means executing set of statements repeatedly until a condition is satisfied.
Java's iteration statements are:

```
                while()
                do - while
                for()
                for each loop - introduced in 1.5
```

1) while loop:

If we don't know the no of iterations in advance then we should go for while loop.

syntax:

```
        while <condition>
          {
                stmt 1;                 }            Body of the loop
            stmt 2;                     }
         }
```

- Loop is pre tested.
- Control enters into the body of the loop if the condition is   satisfied.
- Loop terminates if the condition is failed.
- The curly braces are optional if the block contains only a single statement.

→ The argument to the while loop is always boolean.

Ex:

1) while(b) //valid if "b" is boolean type. ex b=true;
2) while(1) // In valid
CE: Incompatible types
        found : int
        required : boolean

→ Without curly braces only one statement is allowed to take under while and that statement should not be declarative statement.

Examples:

1)

```
class WhileStatement
{
	public static void main(String []args)
	{
		while(true)
		{
			System.out.println("Hello");
		}
		System.out.println("Hi"); //Un reachable statement
	}
}
```
CE: Un reachable statement

2)
```
class WhileStatement
{
	public static void main(String []args)
	{
		while(false)
		{
			System.out.println("Hello"); //Un reachable statement
		}
		System.out.println("Hi");
	}
}
```

CE: Un reachable statement

3)
```
class WhileStatement
{
	public static void main(String []args)
	{
		int a=10, b=5;
		while(a<b)
		{
			System.out.println("Hello");
		}
		System.out.println("Hi");
	}
}
```

4)

```
class WhileStatement
{
	public static void main(String []args)
	{
		int a=10, b=20;
		int i=0;
		while(a<b)
		{
			if (i<10) {
				System.out.println("Hello");
			} else {
				break;
			}
			i++;
		}
		System.out.println("Hi");
	}
}
```

5)
```
class WhileStatement
{
	public static void main(String []args)
	{
		final int a=10, b=20;
		while(a<b) // true ( Compiler evaluates it is true in case of final)
		{
			System.out.println("Hello");
		}
		System.out.println("Hi");
	}
}
```
CE: Unreachable statement

Note : - In the case of "if else" compiler won't check for unreachable statements
ex :
```
	if (true) {
		System.out.println("Hello");
	} else {
		System.out.println("Hi");
	}
```
Ex:
```
	if (false) {
		System.out.println("Hello");
	}
```

2) do - while:

➔ If the loop body required to execute at least once the we should go for do - while loop.

Syntax:
```
	do
	{
	 // body
	} while(b); //";" mandatory, b is boolean
```

→ Curly braces are optional without curly braces only 1 statement is allowed to take between do and while and that statement should not be declarative statement.

EX:

1)
```
	class DoWhileStatement
	{
		public static void main(String[] args)
		{
			do;
			while(true);
		}
	}
```

o/p: Valid

2)
```
	class DoWhileStatement
	{
		public static void main(String[] args)
		{
			do
				int i=10;
			while(true);
		}
	}
```

o/p: in Valid

3)

```
	class DoWhileStatement
	{
		public static void main(String[] args)
		{
			do
			while(true);
		}
	}
```

o/p: in Valid

4)

```
	class DoWhileStatement
	{
		public static void main(String[] args)
		{
			do
				while(true)
					System.out.println("Hello");
			while(false);
		}
	}
```

o/p: Valid

5)

```
	class DoWhileStatement
	{
		public static void main(String[] args)
		{
			do
			{
				System.out.println("Hello");
			} while(true);
			System.out.println("Hi");
		}
	}
```

CE: Unreachable statement

6)

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

```
                do
                {
                        System.out.println("Hello");
                } while(false);
                System.out.println("Hi");
        }
}
```

o/p : Hello
Hi

7)

```
class DoWhileStatement
{
        public static void main(String[] args)
        {
                int a=10, b=20;
                do
                {
                        System.out.println("Hello");

                } while(a<b);
                System.out.println("Hi");
        }
}
```

O/p : Valid

8)

```
class DoWhileStatement
{
        public static void main(String[] args)
        {
                final int a=10, b=20;
                do
                {
                        System.out.println("Hello");

                } while(a<b);
                System.out.println("Hi");
        }
}
```

op: In valid

9)

```
class DoWhileStatement
{
        public static void main(String[] args)
        {
                int a=10, b=20;
                int i=1;
                do
                {
                        System.out.println("Hello");
                        if(i == 100) {
                                b=6;
                        }

                i++;

                } while(a<b);
```

```
		}
	}
```

3) for():

The most commonly used loop is the **for** loop.

→ If we know the no of iterations in advance then we should go for "for loop".

Syntax:

```
	for (initialization; conditional expression; increment/decrement)
	{
		//Body
	}
```

A) Initialization Section:

We are not allowed to declare multiple variables of different data types. but we can declare any no of variables of same type.

Ex:

1) int a=10, b=20; // Valid
2) byte b=10, int i=10; //In Valid
3) int i=10, int j=20; // In Valid (Because the data type should be once)

--> In the initialization part we can take any valid java statement including System.out.println();

Ex:

```
	int i=0;
	for (System.out.println("Hello are U Sleeping"); i<3; i++)
	{
		System.out.println("No Boss U only Sleeping");
	}
```

B) Conditional Expression :

--> Here we can take only valid expression but the result should be of boolean type.

--> This part is optional and if we are not taking any expression, compiler will always place true.

Ex:

```
	class ForStatement
	{
		public static void main(String[] args)
		{
			for (int i=0; ; i++)
			{
				System.out.println("Hello");
			}
		}
	}
```

C) increment & decrement section:

--> Here we can take any valid java statement including System.out.println();

Ex: -

```
class ForStatement
	{
		public static void main(String[] args)
		{
			int i=0;
			for     (System.out.println("No     Boss     U     only     Sleeping");i<10;
System.out.println("Hello"))
			{
```

```
                    i++;
                }
            }
        }
```

--> All the three parts of for loop are independent of each other and optional.

Ex:

```
for ( ; ;); // Valid infinite loop.

        class ForStatement
        {
            public static void main(String[] args)
            {

                for (; ; )
                {
                    System.out.println("Hello");
                }
            }
        }
```

Ex1:

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

                for (int i=0; true; i++)
                {

                    System.out.println("Hello");
                }
                System.out.println("Hi"); // Unreachable statement

            }
        }
```

CE: Unreachable statement

Ex1:

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

                for (int i=0; false; i++)
                {
                    System.out.println("Hello"); // Unreachable statement
                }
                System.out.println("Hi");

            }
        }
```

CE: Unreachable statement

Ex3:

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

                for (int i=0; ; i++)
                {
                        System.out.println("Hello");
                }
                System.out.println("Hi"); // Unreachable statement

        }
}
```

CE: Unreachable statement

Ex4:

```
class ForStatement
{
        public static void main(String[] args)
        {
                int a=10, b=20;
                for (int i=0; a<b; i++)
                {
                        System.out.println("Hello");
                }
                System.out.println("Hi");

        }
}
```

o/p: valid

```
Hello
Hello
.
.
.
.
```

Ex5:

```
class ForStatement
{
        public static void main(String[] args)
        {
                final int a=10, b=20;
                for (int i=0; a<b; i++)
                {
                        System.out.println("Hello");
                }
                System.out.println("Hi"); //Unreachable statement
        }
}
```

CE: Unreachable statement

D) for each loop
Introduced in 1.5 version

--> Specially designed loop to retrieve the elements of arrays and collections.

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

a[0] = 10;
a[1] = 20;
a[2] = 30;

General for loop:

```
	class ForStatement
	{
		public static void main(String[] args)
		{
			int[] a = {10,20,30};
			for (int i=0; i<a.length; i++)
			{
				System.out.println(a[i]);
			}
		}
	}
```

o/p :
```
10
20
30
```

Enhanced For loop:-

```
	class ForStatement
	{
		public static void main(String[] args)
		{
			int a[] = {10,20,30};
			for (int x : a)
			{
				System.out.println(x);
			}
		}
	}
```

o/p :
```
10
20
30
```

--> Even though for each loop is more convenient it is applicable for only arrays and collections. It is not a general purpose loop.
--> For each loop is not applicable to retrieve a particular set of elements. If we want to retrieve all elements of collection or array then only we can use for each.

III) Transfer Statements:
There are 3 transfer statements
```
		break
		continue
		return
```

These are used to transfer the control from one part of the program to another part.

1) Break statement:
--> Inside switch to stop fall-through.(to quit the case)
--> Inside the loop to break based on some condition.
--> Inside labeled class to break the block execution.
Ex:-
```
	switch(x)
	{
		case 10 :
```

```
                        break;
                case 20 :
                        break;
                        .
                        .
                        .
                        .
        }
```

Ex1:

```
        for (int i=0;i<i+1;i++)
        {
                if(i==10)
                        break;
        }

        class BreakStatement
        {
                public static void main(String args[])
                {
                        for ( int i=0; i<10; i++)
                        {
                                System.out.println("Hello");
                                if(i==10)
                                {
                                        break;
                                }
                        }

                }
        }
```

Ex2:-

```
class BreakStatement
{
        public static void main(String args[])
        {
                int i=10;

                L1:
                {
                        System.out.println("Hello");
                        if(i==10)
                        {
                                break L1;
                        }
                        System.out.println("Hai");
                }
                System.out.println("After block");
        }
}
```

Note:- If we are using break anywhere else we will get compile time error.

Ex:

```
class BreakStatement
{
        public static void main (String []args)
        {
                int i=10;
```

```
		if (i>10)
		{
			break;
		}
		System.out.println("Hello");
	}
}
```
Invalid
CE: Break outside switch or loop

2) Continue Statement:
We should use continue statement inside a loop to skip the current iteration and continue for the next iteration.

Ex:

```
for (int i=0; i<=10; i++)
{
	if (i%2==0)
		Continue;
	System.out.println(i);
}
```

o/p :

```
1
2
3
5
7
9
```

Ex:
```
class ContinueStatement
{
	public static void main(String[] args)
	{
		for (int i=0; i<=10; i++)
		{
			if (i%2==0)
				continue;
			System.out.println(i);
			//System.out.println("Hello"+(4%2));
		}
	}
}
```

Note:- If we are using continue outside loop we will get compile time error.
Ex:1
```
class ContinueStatement
{
	public static void main(String[] args)
	{
		int i=10;
		if (i>10)
			continue;
		System.out.println(Hello World);
	}
}
```

CE: continue outside loop.

Arrays:

Array:

Group of similar elements referred with a single name.

```
Syntax:  Datatype <variable name[size]>
Ex:-    int arr[10] or int[10] arr
```

The array arr is initially set to null. new is a special operator that allocate memory.

```
Syntax:-
             <Variable name>= new datatype[size]
    Ex:-     arr=new int[10];
```

There are two types of arrays

1. Single dimensional
2. Multi dimensional

Single dimensional:

```
//Program name: Months.java
class Months
{
      public static void main(String args[])
      {
            int month_days[];
            month_days=new int[12];
            month_days[0]=31;
            month_days[1]=28;
            month_days[2]=31;
            month_days[3]=30;
            month_days[4]=31;
            month_days[5]=30;
            month_days[6]=31;
            month_days[7]=31;
            month_days[8]=30;
            month_days[9]=31;
            month_days[10]=30;
            month_days[11]=31;
            System.out.println("March has:"+month_days[2]+" Days");
      }
}
```

```
//Program name: Array.java
class Array
{
      public static void main(String args[])
      {
            int arr[]={10,20,30,40,50};
            int i;
            for(i=0;i<5;i++)
            {
                  System.out.println(arr[i]);
            }
      }
}
```

Note: If the array is initialized at the time of declaration then new key word is not required.

Multi dimensional arrays:

```
//Program name: TDarray.java
class TDarray
{
	public static void main(String args[])
	{
		double no[][]=new double[3][3];
		int i,j;

		for(i=0;i<3;i++)
		{
			for(j=0;j<3;j++)
			{
				System.out.print(no[i][j]+"  ");
			}
			System.out.println();
		}
	}
}
```

Declaring double dimensional array without specifying column size.

Note: The Column size can be varied according to user’s requirement.

```
//Program name: DTest.java
class DTest
{
	public static void main(String args[])
	{
		int Darr[][]=new int[4][];
		Darr[0]=new int[1]; // 0th row contains 1 column
		Darr[1]=new int[2]; // 1st row contains 2 columns...
		Darr[2]=new int[3];
		Darr[3]=new int[4];

		// filling the array elements
		int k=0;
		for(int i=0;i<4;i++)
		{
			for(int j=0;j<i+1;j++)
			{
				Darr[i][j]=k;
				k++;
			}
		}
		for(int i=0;i<4;i++)
		{
			for(int j=0;j<i+1;j++)
			System.out.print(Darr[i][j]+" ");
			System.out.println();
		}
	}
}
```

Why array index start with zero?

Ans1:

1) In my opinion, 0 is the lowest positive value that is why it starts from 0.

Reasons

1.In very earlier languages like Pascal, c, logic computer memory was very limited hardly in bytes, also their not os like windows or Linux

so programmers them selves had to locate memory loc to variables as we done even in 8085 kit

2.all computer data is digitally organized(ie in binary) even today

0- 0000

1-0001

2 0010
3-0011
...
3.so their was a physical existence of zeroth location in computers
4.for better memory loc calculation it was a trend to have arrays starting with zeroth loc, so is now

Accepting dynamic data to the java program:
--> In java programming we have 2 approaches to accept dynamic data they are

1) Accepting dynamic data from command prompt
2) Accepting dynamic data from keyboard.

1) Accepting dynamic data from command prompt:
--> Why main() takes String as a parameter?
--> main() - always takes String type as parameter
--> main() - takes array of params string
--> main() - java oop
To String type OOPS principle String class is used.
--> Since Java is pure OOPL - all command line arguments represents array of String Objects.
*--> Whatever the data we represent in java programming environment it is by default treated as String types.
--> The command line arguments which we are passing to every java program are send into main()
*--> In main() , to hold the command line arguments we always use String as a parameter, i.e array of String type.
--> Since java is pure Object oriented programming language all command line arguments represented has array of objects of String class.
*--> While we are using array of objects of String class, we should not specify the size of the array.

Command line Arguments:
The number of values which we pass from command prompt to the java program are known as command line arguments and whose position starts from 0,1,2,.....n-1.

--> Write a java program to print command line arguments

```
class PrintData
{
	public static void main(String k[]) // here String k[] = {"12", "Srinu", "30", "kailash", "true"};
	{
		System.out.println("Size of Input Arguments = "+k.length);
		for ( int i=0; i< k.length; i++)
		{
			System.out.println(k[i]);
		}
	}
}
```

Define a wrapper class and explain its use:
For each and every fundamental data type there exists a predefined class, which is known as wrapper class. In java programming we have 8 predefined wrapper classes since there exists 8 fundamental data types.

The purpose of wrapper classes is that which will convert String data into numerical data and numerical data into object data. and object data into numerical data.

Write a java program which will compute sum of 2 numbers by accepting data from command prompt.

```
class Sum
{
	int a,b,c;
	void assign(int n1, int n2)
	{
		a=n1;
		b=n2;
	}
	void add()
	{
		c = a+b;
	}
	void display()
	{
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+b);
		System.out.println("Value of c = "+c);
	}
}

class SumDemo
{
	public static void main(String[] args)
	{
		System.out.println("No of arguments = "+args.length);
		System.out.println("Begin");
		int n1 = Integer.parseInt(args[0]);
		int n2 = Integer.parseInt(args[1]);

		Sum so = new Sum();
		so.assign(n1,n2);
		System.out.println("Assigned values successfully...");
		so.add();
		System.out.println("Added Successfully...");
		so.display();
		System.out.println("All Values printed successfully...");
		System.out.println("End");
	}
}
```

In the above program, the command line arguments are passed to execution logic(SumDemo) of execution logic method(main()) and it is available in the form of String.

We convert the string data into the numerical data in execution logic class and it is passed to business logic class for its processing and finally the program flow will be terminated in execution logic method of execution logic class.

Write a java program for generating multiple vector table for given number, by accepting from command prompt.

```
class Mul
{
	int n;
	void set(int n1)
	{
		n=n1;
	}
	void table()
	{
		for (int i=1; i<=10; i++)
		{
			System.out.println(n+"*"+i+"="+(n*i));
		}
	}
}

class MulDemo
{
	public static void main(String[] args)
	{
		if(args.length == 0)
		{
			System.out.println("Please pass the value...");
		}
		else
		{
			int n1 = Integer.parseInt(args[0]);
			if (n1 <= 0)
			{
				System.out.println(n1+ " is invalid Input");
			} // Inter If
			else
			{
				Mul mo = new Mul();
				mo.set(n1);
				mo.table();
			} // Inter else
		} // Outer Else
	} // Main
} // mul demo
```

<u>Writing you own Java Methods:</u>

Write a java program it illustrate the concept of your own methods

```
class  MethodsBus
{
	int a; // Variable declaration
	String name = "Kailash"; // Variable declaration cum initilization
	static int noOfHits; // static Variable declaration
	static  String  lastRunTime="CURRENT_TIMESTAMP";  //  static  Variable  declaration  cum 
initilization
	int b = 20; // Variable declaration cum initilization
	final double PI = 3.14; // Creating final constant varibale

	int total(int a, int b)
	{
		int c = a+b;
		return c;
	}
	void total1(int a, int b)
	{
		int c = a+b;
		System.out.println("Inside Total1 Void "+c);
	}
	int total2()
	{
		int a = 10+20;
		return a;
	}
	void total3()
	{
		int a = 30+50;
		System.out.println("Inside Total3 Void "+a);
	}
}


class MethodsDemo
{
	public static void main(String[] args)
	{
		MethodsBus mbus = new MethodsBus();
		int retValue = mbus.total(1,2); // return type and input arguments
		System.out.println("Returned Value = "+retValue);

		mbus.total1(3,5); // Only input arguments , no return type.

		int t2mrv = mbus.total2(); // return type and no input arguments
		System.out.println("Returned Value from t2 method = "+t2mrv);

		mbus.total3(); // no input arguments , no return type.
	}
}
```

Constructors

Constructor:

A constructor is a special member method, which will be called automatically by the JVM when an object is created , for placing our own values instead of placing default values.

Advantages of Constructor:

1) It eliminates in placing default values.
2) It eliminates in calling ordinary methods.
3) The purpose of constructor is that to initialize the object.

Rules / Characteristics/ Properties of Constructors :

1) Constructor will be called automatically when an object is created.

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

	//Here Test() is a constructor , we can call it as class function name.

Example :

```
class Test
{
	Test(){
		System.out.println("I am from Constructor");
	}
	public static void main(String[] args)
	{
		Test t1 = new Test();
		System.out.println("Hello World!");
	}
}
```

2) Constructor name must be same to class name.
3) Constructor never return any value even "void" also.(If we write void before the constructor it is treated as ordinary method).
4) Constructor should not belongs to static,
	(Since constructors will be called each and every time when an object is created and they depend on object only but not class).

5) Constructors are not inherited,

For Example we have class A and class B

class A constructor can not use class B constructor to create an object.

```
Class A
{}
class B extends A
{
}
```

6) Constructors will not be inherited from one class to another.

-- (So to make private no other execution class is created, it is in the same class)

7) Constructors can be private provided an object of one class created in its own context/ scope of constructor con not be private provided on object of one class can be created
in the context of some other class.

Types of Constructors:

Based on initializing the object , in java we have 2 types of constructors, they are

1) Default /Parameter less/ no - argument constructor
2) Parameterized constructor

1) Default Constructor:

A default constructor is one which never takes any parameter values.

Syntax :

```
class <className>
{
      <className>()  // default constructor
      {
            Block of statements
      }
}
```

Write a java program which illustrate the concept of default constructor.

```
class TestDC
{
      int a,b;
      TestDC()
      {
            Syste.out.println("I am from default constructor");
            a=123;
            b = 456;
            Syste.out.println("Value of a = "+a);
            Syste.out.println("Value of b = "+b);
      }
}

class DCDemo
{
      public static void main(String args[])
      {
            TestDC dc = new TestDC();
      }
}
```

Rule 1 :

When we create an object with default constructor then defining the default constructor is optional.

If we are not defining default constructor then JVM will call System defined default constructor and it places default values for the data members of class.

If we define the default constructor then JVM will call Programmer defined default constructor for placing programmer defined values for data members of the class.

2) Parameterized Constructor:

A parameterized constructor is one which always takes parameters.

when ever we want to create multiple objects with different values of same type of class. Then we must use the concept of parameterized constructors.

Syntax:

```
class <className>
{
	<className>(list of formal parameter)
	{
		// Block of statements
	}
}
```

Write a java program which illustrate the concept of parameterized constructor.

```
class TestPC
{
	int a, b;
	TestPC(int x, int y)
	{
		System.out.println("I am from Parameterized constructor");
		a=x;
		b=y;
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+b);
	}
}

class ParamsConst
{
	ParamsConst()
	{
		System.out.println("I am from ParamsConst() begin");
		TestPC pc1 = new TestPC(10,20);
		TestPC pc2 = new TestPC(100,200);
		TestPC pc3 = new TestPC(1000,2000);
		System.out.println("I am from ParamsConst() End");
	}
}

class PCDemo
{
	public static void main(String aregs[])
	{
		ParamsConst pconst = new ParamsConst();
	}
}
```

Rule2:

we create an object with parameterized constructor then defining the parameterized constructor is mandatory for the java programmer. otherwise compile time error will come.

Object Parameterized Constructor:

It is one which always takes object as parameter

Object parameterized constructors are used for copying the content of one object into another object where both the objects must belongs to same type .

Examples:

```
class Test
{
	int a, b;
	Test(){}
	Test(int x, int y) {}
}

class TestOPC
{
	int c,d,e;
	TestOPC()
	{
		c = 30;
		d = 40;
		e= 50;
	}
}
class TestPC
{
	int a, b;
	TestPC(TestOPC x)
	{
		System.out.println(" I am from Object Parameterized Constructor");
		System.out.println("Value of c = "+x.c);
		System.out.println("Value of d = "+x.d);
		System.out.println("Value of e = "+x.e);
	}
	void add(int a,  int b){
		System.out.println("Sum of A and B = "+(a+b));
	}
}
class PCDemo
{
	public static void main(String aregs[])
	{
		TestOPC opc = new TestOPC();
		TestPC pc2 = new TestPC(opc);
	}
}
```

Javap:

To see the profile of any class in java we use a tool called "javap".

syntax: `javap className`

```
Ex: javap PCDemo
```

In any of the class, we may have two types of constructor they are one single default constructor and "n" number of parameterized constructor.

Each and every class is by default containing single default constructor and it may belongs to system defined or programmer defined constructor.

"This" Keyword:

'This' is a keyword or implicit object created by JVM for two purposes, they are

1) It always contains address of the current class object or it points to current class object.

2) When ever formal parameters and data members of the class are similar then JVM gets an ambiguity. In order to differentiate the data members of the class with formal parameters
the data members of class must be preceded by a keyword 'This'

syntax : `This.data members of their class`

Write a java program which illustrate the concept of 'This' Keyword.

```
class ThisSample
{
	int a,b;
	ThisSample(int a, int b)
	{
		System.out.println("I am from duble parameeterized constructor");
		this.a = a;
		this.b = b;
		this.a = this.a+1;
		this.b = this.b +1;
	}
	void disp()
	{
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+
		);
	}
}
class ThisDemo
{
	public static void main(String k[])
	{
		ThisSample s1 = new ThisSample(10,20);
		ThisSample s2 = new ThisSample(100,200);
		s1.disp();
		s2.disp();
	}
}
```

Note : s1.disp()

Display is a member method called with respect to s1, in display method definition the values of s1 are refereed a,b directly or this.a , this.b;

In general which ever object is used for calling the function or method, with in the method definition object values are referred as directly or this.value1 and this.value2.... this.valuen

Rule3:
When ever we create multiple objects with both default and parameter in constructors it is mandatory for the java programmer to define both default and parameterized constructor.

Overloaded Constructor:

A constructor is said to be overloaded if and only if constructor is same but its signature is different signature represents the following,

No of parameters
Type of parameters
Order of parameters.

At least one of the above must be differentiate.

Ex:
```
Test t1 = new Test(100,200);
 Test t1 = new Test(100,200,300);
 Test t1 = new Test(10.5f,20.7f);
 Test t1 = new Test(20,40.3f);
```

Object Parameter Constructor:

It is one which always takes object as a parameter, object parameterized constructors are used for copying the content of one object into another object where both objects must belongs to same type.

Function this() and this(...) :

The above functions are used for calling current class constructors to each other.

this():

It is used for calling current class default constructor from current class parameterized constructor.

this(...): // Here "..." represents parameters or arguments

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

Rule4:

When ever we use either this() or this(...) is the current class constructors. they must be used always as the first statements, other wise we will get compile time error.

Write a java program which illustrate the concept of this() and this(...).

```
class SampleTest
{
	int a,b;
	SampleTest() // (3)
	{
		System.out.println("I am from default constructor");
		a =1;
		b=2;
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+b);
	}
	SampleTest(int x) // (1)
	{
		this(10,20);
		System.out.println("I am from single parameeterized constructor");
		a =b =x;
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+b);
	}
	SampleTest(int a, int b) // (2)
	{
		this();
		System.out.println("I am from double parameeterized constructor");
		this.a =a;
		this.b=b;
		System.out.println("Value of a = "+this.a);
		System.out.println("Value of b = "+this.b);

	}
}

class ThisOdemo
 {
	public static void main(String args[])
	{
		SampleTest st = new SampleTest(100); // The control goes to (1)
	}
 }
```

In the above program the constructors are calling in the order 1,2,3 and they are executing in the order 3,2,1.

In general the constructors are calling in java, is increasing order(bottom to Top) and they are executing in decreasing order(top to bottom).

***) Write a java program for addition of two objects by accepting the data from the command prompt.;**

```
 class OTest
 {
      int a,b;
      OTest(){}
      OTest(int a , int b)
      {
            this.a = a;
            this.b = b;
      }
      void display()
      {
            System.out.println("Value of a = "+a);
            System.out.println("Value of b = "+b);
      }
      OTest sum(OTest o1)
       {
            OTest o2 = new OTest();
            o2.a = this.a+o1.a;
            o2.b = this.b+o1.b;
            return o2;
       }
}

class OTestDemo
{
      public static void main(String k[])
      {
            if (k.length != 4)
            {
                  System.out.println("Invalid Input");
            }
            int n1 = Integer.parseInt(k[0]);
            int n2 = Integer.parseInt(k[1]);
            int n3 = Integer.parseInt(k[2]);
            int n4 = Integer.parseInt(k[3]);

            OTest ot1 = new OTest(n1,n2); // 10, 20
            OTest ot2  = new OTest(n3,n4); // 30, 40

            OTest ot3 = new OTest();

            ot3 = ot1.sum(ot2); // we can't use operators in objects like ot1+ot2. // ot3
constains sum of 2 onjects a = 40, b = 60.

            System.out.println("ot1 Values");
            ot1.display();

            System.out.println("ot2 Values");
            ot2.display();

            System.out.println("ot3 Values");
            ot3.display();

      }
}
```

Note : A method is not only returning fundamental values , predefined class names, but also returns "User Defined class Names".

```
int a =20;
int b= 30;
int c = a+b;

int add( int a, int b)
{
      int c = a+b;
}
```

<u>Conversions In Java:</u>

Conversion of java makes us to understand , converting the data from one type to another type.

In java programming we have 6 types of conversions

They are

1) Converting String data into Fundamental or Numerical data.
2) Converting Fundamental /Numerical Data into String data.
3) Converting Fundamental /Numerical Data into Object data.
4) Converting Object data into Fundamental or Numerical data.
5) Converting Object data into String data.
6) Converting String Data into Object data.

1) Converting String data into Fundamental or Numerical data:

In order to convert the String data into fundamental/ Numerical data we have must use the following predefined generalized methods,
Which are present in each and every wrapper class.

The following table use the fundamental data type , whose corresponding wrapper class name and conversion method from String data numerical/ fundamental data.

Fundamental Data type	Wrapper class Name	String to Fundamental/Numerical Data type
byte	Byte	public static byte parseByte(String)
short	Short	public static short parseShort(String)
int	Integer	public static long parseInt(String)
long	Long	public static float 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)

Syntax:

```
Wrapper class Name
       └─>  public static XXX parseXXX(String)
```

// Here XXX is any fundamental data type

Examples

1)

```
Integer --> public static int parseInt(String);

String s = "10";
int  i1 = Integer.parseInt(s);
```

2)

```
        Float --> public static float parseFloat(String);

        String s = "10.75f";
        float f1 = Float.parseFloat(s);
```

3)

```
        String b = "true";
        boolean b = Boolean.parseBoolean(b);
```

4)

```
         String c = "K";
        char chr = Character.parseChar(c);
```

2) Converting Fundamental /Numerical Data into String data:

In order to convert fundamental data into String data we must use the following predefined generalized static methods which is present in String class.

Hence the predefined class "String" contains 8 predefined overloaded static methods.

Example :

int i = 20;
String s = String.valueOf(i);

3) Converting Fundamental /Numerical Data into Object data:

In order to convert numerical or fundamental data into Object data we use the constructors of 8 wrapper classes,
and they are given in following table.

Fundamental Data type	**Wrapper class Name**	Constructors
byte	Byte	Byte(byte)
short	Short	Short(shot)
int	Integer	Integer(int)
long	Long	Long(long)
float	Float	Float(float)
double	Double	Double(double)
char	Character	Character(char)
boolean	Boolean	Boolean(boolean)

In general each and every wrapper class con their parameterized constructors which will take fundamental data type value as a parameter and whose generalized notation given bellow

Wrapper class Name

↳ Wrapper class Name(XXX)

Here XXX represents fundamental data type value

Example:

```
int a = 10;
Integer intObj = new Integer(a);
double d = 10.99;
Double douObje = new Double(d);
intObj+douObje // Error, but we can add a+d
                                because a and d are fundamental data types.
```

Since Object data can not be modified(so more security)

4) Converting Object data into Fundamental or Numerical data:

In order to convert Object data into fundamental data we must use the "instance method" which is present in various wrapper classes,
and whose details are given in the following table
--> In general each and every wrapper class contains the following generalized predefined instance methods, which will convert object data into fundamental data.

Wrapper class Name
$\downarrow$
public XXX XXXValue(); // Here XXX represents fundamental data type.

Fundamental Data type	Wrapper class Name	Constructors
byte	Byte	public byte byteValue()
short	Short	public short shortValue()
int	Integer	public int intValue()
long	Long	public long longValue()
float	Float	public float floatValue()
double	Double	public double doubleValue()
char	Character	public char charValue()
boolean	Boolean	public boolean booleanValue()

Examples:

```
int a =10;
Integer inObje = new Integer(a);
int x = inObje.intValue();
Float fObje = new Float(10.75f);
float f = fObje.floatValue();
```

5) Converting Object data into String data:

In order to convert object data to string data we have following predefined instance methods , which is present in each and every wrapper class.

Wrapper class Name
public String toString();// This is instance method, it will called with the help of object.

```
Ex:    Double do1 = new Double(10.47);
       String s1 = d01.toString();
       Integer io = new Integer(3000);
       String s2 = io.toString();
```

6) Converting String Data into Object data:

In order to convert String data into object data each and every wrapper class contains a parameterized constructors,
which is taking as a parameter and whose details are given bellow,

Fundamental Data type	Wrapper class Name	Constructor Name
byte	Byte	Byte(String)
short	Short	Short(String)
int	Integer	Integer(String)
long	Long	Long(String)
float	Float	Float(String)
double	Double	Double(String)
char	Character	Character(String)
boolean	Boolean	Boolean(String)

In general each and every wrapper class contains the following generalized parameterized constructor by taking String as a parameter.

WrapperClassName
WrapperClassName(String)

```
Ex:     String s ="4000";
        Integer io = new Integer(s); // String data to object data
        String s1 = "true";
        Boolean b = new Boolean(s1);
```

Examples Overview:

```
String s2 = "20";
int x = Integer.parseInt(s2); // String data to Fundamental data
String s3 = String.valueOf(x); // Fundamental data to String data
Integer io = new Integer(s3); // like "10" String to Object data
int y = io.intValue(); // Object data to Fundamental data
Integer io1 = new Integer(y); // Fundamental data to Object data.
```

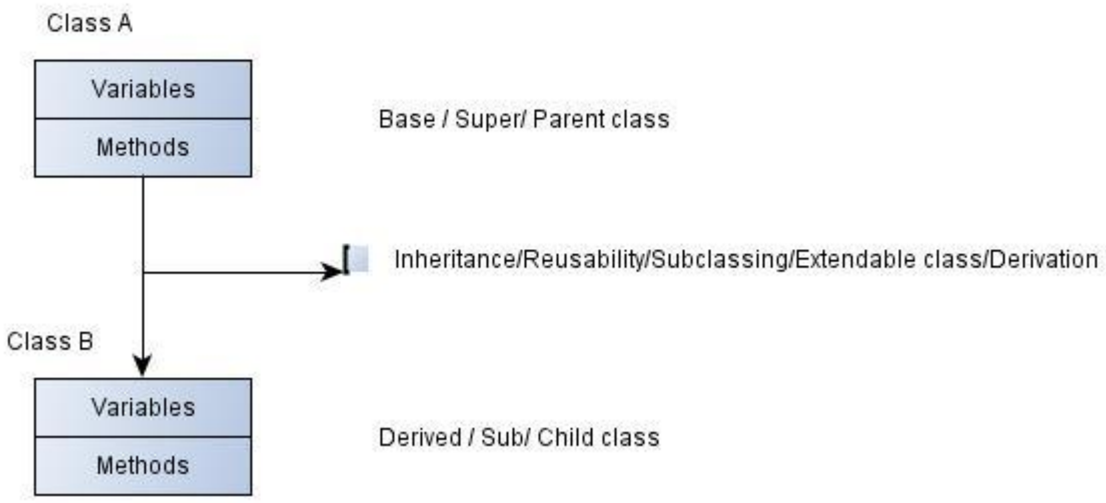
Inheritance

def: Inheritance is a mechanism of getting the Variables and methods from one class to another class.
--> Typically class contains Variables + methods.
--> The class which is giving Variables and methods is known as Base/Super/Parent class.
--> The class which is taking Variables and methods from Base class is known as Derived /Sub/ Child class.

Definition of derived class:

A derived class is one which always contains some of the features on its own(Variables + Methods) and some of the features from base class.

The process of inheritance is also known as Re-usability or extendibility or Sub classing or Derivation(new class from exiting class)


Class A
Variables
Methods
Base / Super/ Parent class
Inheritance/Reusability/Subclassing/Extendable class/Derivation
Class B
Variables
Methods
Derived / Sub/ Child class

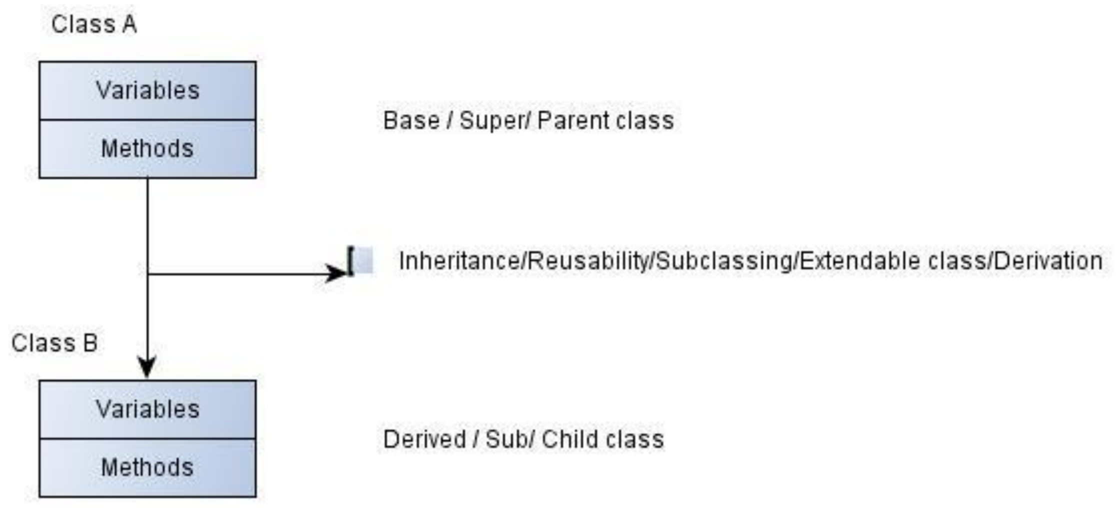
Advantages of inheritance /Re-usability:
1) Application development time is less.
2) Memory space is less, hence we get higher performance and low memory cost.
3) Redundancy of the code is reduced, hence we get less execution time and consistent result.

Reusable techniques / Inheritance types:

Based on inheriting features from one class to another class , in java we have 5 types of inheritances, they are

1) Single Inheritance:

It is one in which there exits single base class and a single derived class.

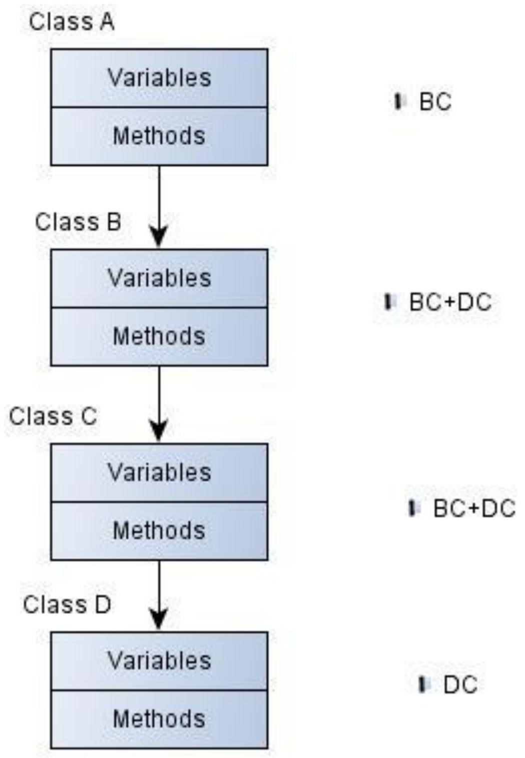


Class A
Variables
Methods
Base / Super/ Parent class
Inheritance/Reusability/Subclassing/Extendable class/Derivation
Class B
Variables
Methods
Derived / Sub/ Child class

Inheritance Path : Class A --> Class B

2) Multi Level inheritance:

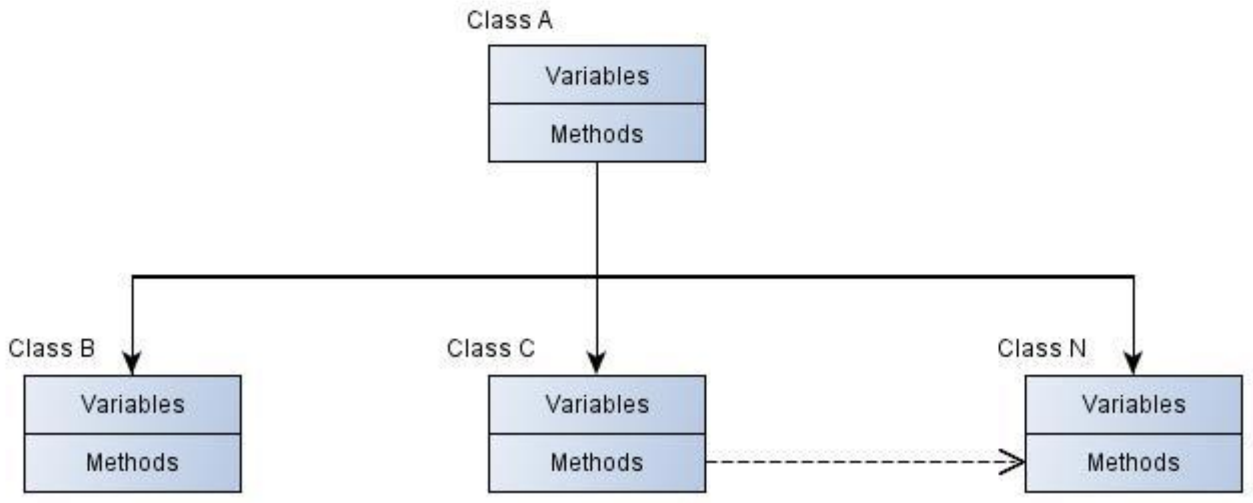
It is one in which there exists single base class, single derived class and "multiple intermediate base classes.(It is one, in one context it acts as a base class and in another context it acts as derived class).



Inheritance Path : ClassA --> ClassB --> ClassC --> ClassD

3) Hierarchical Inheritance:

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

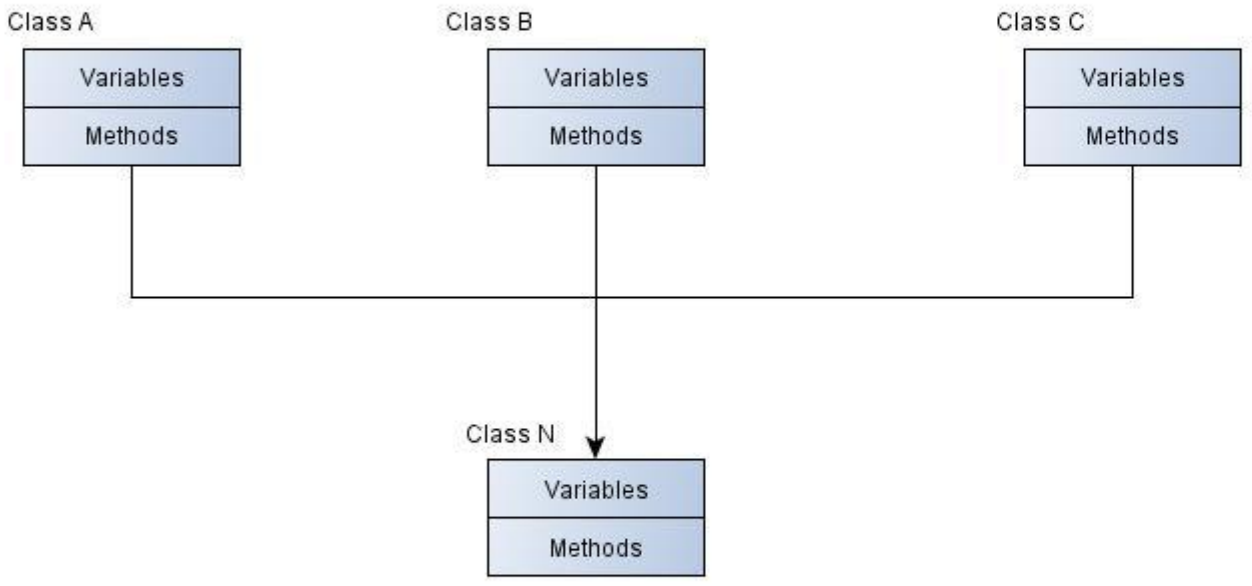


Inheritance Path :

ClassA --> ClassB
ClassA --> ClassC
ClassA --> ClassD
ClassA --> ClassN

4) Multiple Inheritance:

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



Inheritance Path :

ClassA --> ClassN
ClassB --> ClassN
ClassC--> ClassN

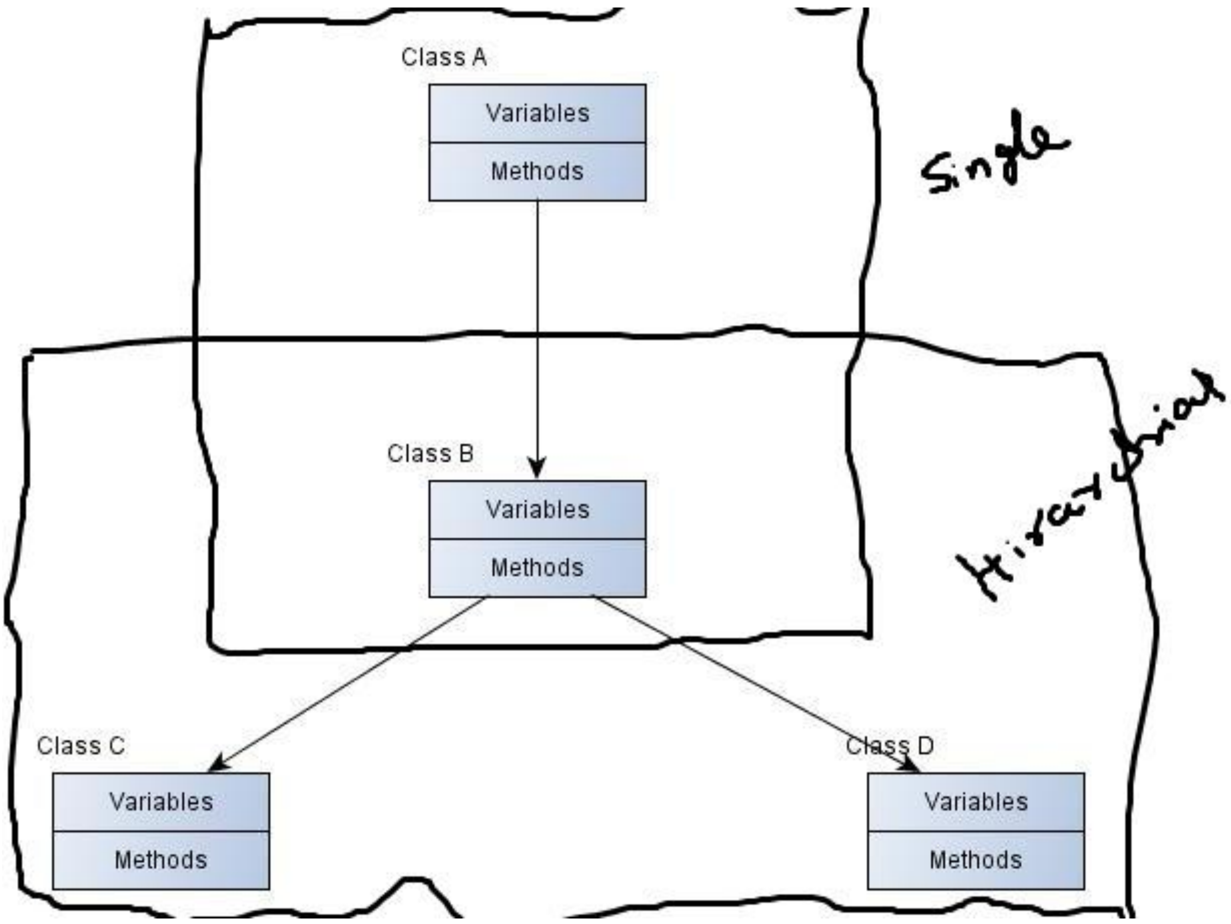
In java multiple inheritance is not supported through the concept of classes but it can be supported through the concept of 'interfaces'.

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 is not supported in java, through classes concept.
But it can be supported through interfaces concept.

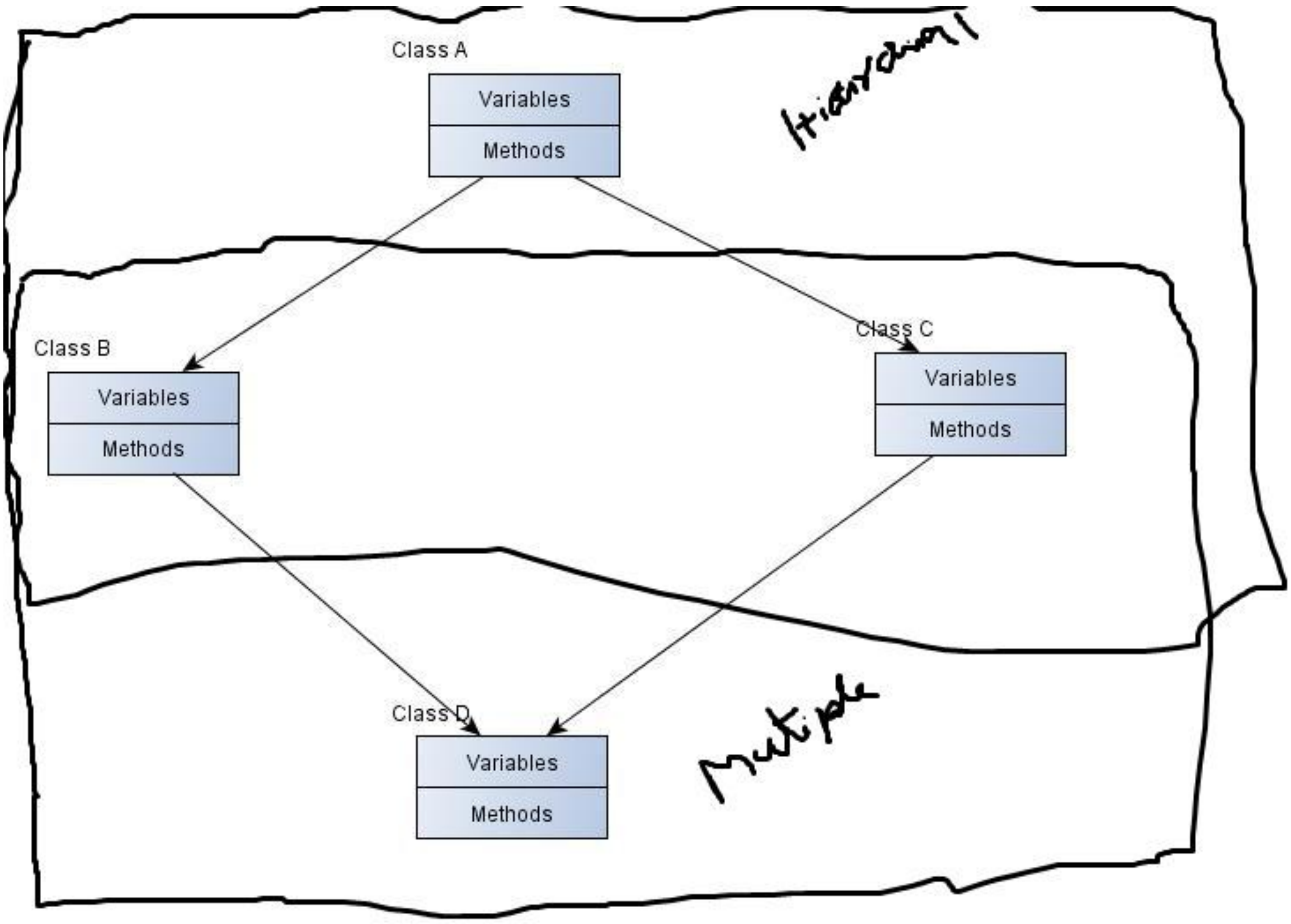
Example1:



Inheritance Path :

ClassA --> ClassB --> ClassC
ClassA --> ClassB --> ClassD

Example2:



// In valid this through classes but valid through interfaces

Syntax:
syntax for inheriting base class features into derived classes.

```
class <ClassName2> extends <ClassName1>
{
      Variable declaration;
      method definition;
}
```

--> In the above syntax
--> <CLassName1> and <ClassName2> represents name of the base class and derived class respectively.
--> 'extends' is a keyword which is used for inheriting the features of base class into derived class. plus it improves the functionality of derived class.
The functionality of derived class
-->One derived class can extends only one base class.
-->Constructors of base class can not be inherited into derived class.
--> If we don't want to give the complete features of base class to derived class, then the base class definition must be made as 'final'. Hence final classes can not be inherited or reused in derived classes.
--> i.e final classes never participates in Inheritance process.

Example :
```
final class A
{
      //Variables
      //Methods
}
```

```
	class B extends A  // Error because final classes are not supported inheritance process
	{
		// Block of statements
	}
```
--> Once the base class is final then its variables and methods are automatically belongs to final.
--> If we don't want to give the some of the features of base class to the derived class then those features must be made as 'private'.
Hence private features of base class can not be inherited into derived classes.
--> When ever we develop any inheritance application, it is always recommended to create an object of bottom most derived class. since bottom most class contains all the features of its super classes.
--> When ever we create an object of bottom most derived class, we get the memory space for base class members/features first and then we will get the memory space for derived class members/features.
--> For all the classes in java, there exists an implicit super class known as java.lang.Object, since it provides garbage collector program to its sub classes.

Write a java program which illustrate the concept of inheritance.

```
// Base Class
class C1
{
	int a,b;
	void display()
	{
		System.out.println(" I am from display method of base class ... ");
	}
} // C1-Base class

// Derived class

class C2 extends C1
{
	int c,d;
	void display1()
	{
		System.out.println(" I am from display1 method of derived class ...");
		a=10;
		b=20;
		c=30;
		d=40;
		System.out.println("Value of a = "+a);
		System.out.println("Value of b = "+b);
		System.out.println("Value of c = "+c);
		System.out.println("Value of d = "+d);
	}
}

// Executable class
class Idemo1
{
	public static void main(String args[])
	{
		C2 do = new C2();
		do.display();
		do.display1();
	}
}
```

Note :
--> In Inheritance application development, a base class contains final features and private features.

--> A derived class can not use/access private features of vase class.
Where as the derived class can use final features of base class and we can not change the content/ value in derived class.

"Super" Keyword:

In order to differentiate the base class features with derived class features we use a keyword 'Super'.
The keyword Super is placing an important role in 3 places in java

1) Super at variable level.
2) Super at method level
3) Super at constructor level.

1) Super at variable level:

Whenever we inherit the base class variables into derived class, there is a possibility that base class variables are similar to derived class variables then JVM gets an ambiguity.
--> In order to differentiate the base class variables with derived class variables in the context derived class. the base class members must be preceded by keyword 'super'.

Syntax:

super.baseClassVariableName;

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

```
class BaseClass
{
	int a;
}

class DerivedClass extends BaseClass
{
	int a,c;
	void set(int x, int y)
	{
		super.a = x; //  here if we miss super then the output is default value in base
clas variable
		a=y;
	}

	void add()
	{
		c = super.a+a;
	}

	void display()
	{
		System.out.println("Value of base class variable a = "+super.a);
		System.out.println("Value of  a = "+a);
		System.out.println("Value of c = "+c);
	}
}

class Idemo2
{
	public static void main(String [] aaas)
	{
		int x = Integer.parseInt(aaas[0]);
		int y = Integer.parseInt(aaas[1]);
		DerivedClass dc = new DerivedClass();
			dc.set(x,y);
```

```
                dc.add();
                dc.display();
        }
}
```

2) Super at method level:

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

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

Syntax:

super.BaseClassMethodName();

Ex) Write a java program which illustrate the concept of super keyword at method level.

```
        class BaseClass1
        {
                void display()
                {
                        System.out.println("I am from base class display() ..");
                }
        }

        class DerivedClass1 extends BaseClass1
        {
                void display()
                {
                        super.display();
                        System.out.println("I am from derived class display()...");
                }
        }

        class Idemo3
        {
                public static void main(String args[])
                {
                        DerivedClass1 dc1 = new DerivedClass1();
                        dc1.display();
                }
        }
```

Note : In the above program, if we eliminate 'super' keyword before display() method in DerivedClass1, we get a runtime error known as java.lang.StackOverFlowError
--> An error at runtime is known as Exception.

Method Overriding:

Method Overriding = method Heading is Same + Method body is different.
or
--> Method overriding is a process 'redefining' the original method into derived class(es) for doing different operations.

Ex:- Write a java program which illustrate the concept of super keyword in method level.

```
class BaseClass2
{
        void operation(int x, int y)
        {
                int z = x+y;
                System.out.println("Sum in  Base class = "+z);
        }
}

class DerivedClass2 extends BaseClass2
{
        void operation(int x, int y)
        {
                //super.operation(x,y);
                int z = x*y;
                System.out.println(" Mul in Derived class = "+z);
        }
}
class Idemo4
{
        public static void main(String args[])
        {
                int x = Integer.parseInt(args[0]);
                int y = Integer.parseInt(args[1]);

                DerivedClass2 dc2 = new DerivedClass2();
                dc2.operation(x,y);
        }
}
```

Question:

Q) How do you call the base class method from derived class method, when they are same?
Ans: super.BaseClassMethodName;
Q) How do you call the original method() from overridden method?
Ans : super.originalmthodName;

What is the difference between method overloading and methods overriding.

Method Overloading	Method Overriding

3) Super at constructor level:

Whenever we develop any inheritance application we always use to create an object of bottom most derived class for obtaining more number of features from base classes.

When we create an object of derived class , we get the memory space for base class members first later we get for derived class members. In any inheritance application development
base class members has to be initialized and later derived class members must be initialized.

Let us consider the following hierarchy

```
TOP                                                        TOP
e|              C1   C1 o1 = new C1();                      /\
x|               |              /\                         c |
e|              \/               |                         a |
c|              C2   C2 o2 = new C2();                      l |
u|               |              /\                         l |
t|              \/               |                         i |
i|              C3     C3 o3 = new C3();                   n |
      o|                                              g |
      n|                                                |
       |                                                |
      \/
    Bottom                                          Bottom
```

--> In the above hierarchy c2 default constructor is automatically calling c2 default constructor.
--> C2 default constructor is in terms calling C1 default constructor.
--> C1 default constructor is executing first later C2 default and finally C3 default constructor.
--> Hence in any inheritance application, the way of calling the constructors is from bottom to top and the way of executing is from top to bottom. Since the base class members has to be initialized first and later the derived class members.

Ex) Write a java program which illustrate the concept of calling the constructors is from bottom to top and the execution is from top to bottom.

```
class BC1
{
	BC1() // (3)
	{
		System.out.println("I am from BC1 default Constructor");
	}
}
class IBC1 extends BC1
{
	IBC1() // (2)
	{
		System.out.println("I am from IBC1 default Constructor");
	}
}

class DC1 extends IBC1
{
	DC1()  // (1)
	{
		System.out.println("I am from DC1 default Constructor");
	}
}

class Idemo5
{
	public static void main(String args[])
	{
		DC1 dco = new DC1();  // Control goes to (1)
	}
}
```

Note: In the above programs Constructors are calling in the order of (1), (2), (3) and they are executing in the order of (3), (2), (1).
In general constructors are calling in increasing order and executing in decreasing order.

super(), super(...):

The above functions are used for calling super class constructors from derived class constructor.

super():

It is used for calling super class default constructor from derived class constructors.

super(...):

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

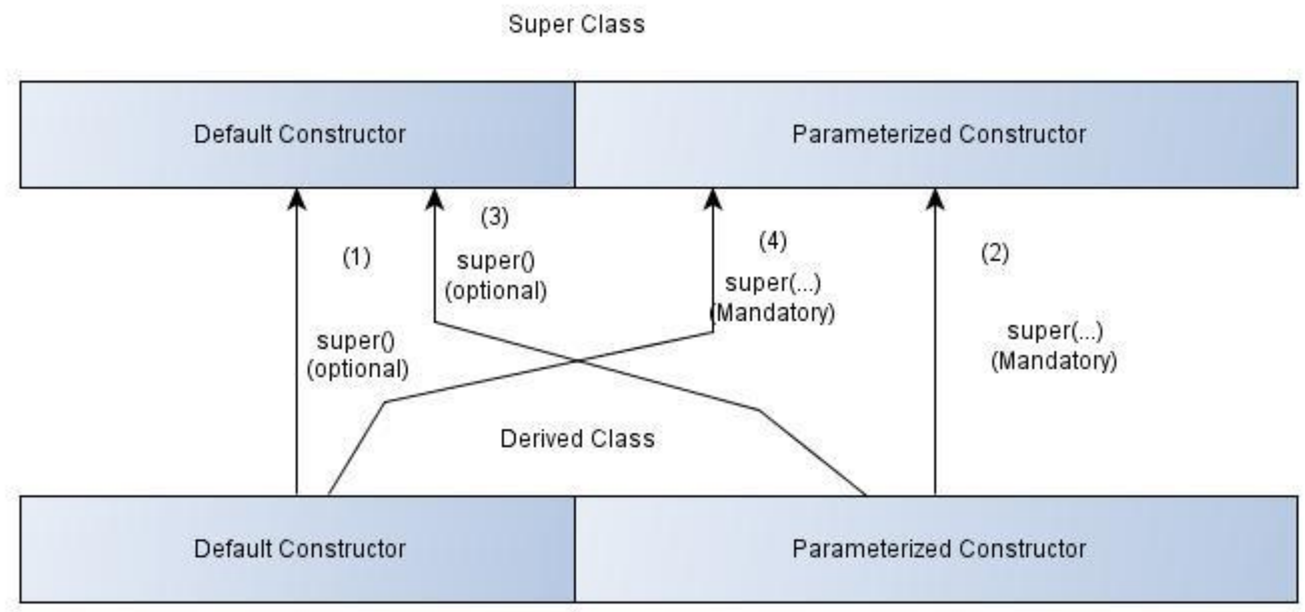
Rule :

When ever we use either super() or super(...) in the derived class constructors, they must be used always as first statement other wise we get compile time error.

Q) What is the difference between this(), this(...) and super(), super(...)?
ans : this(), this(...) are used for calling current class constructors where as super(), super(...) are used for calling Base/Super class constructors.

No of ways super(), super(...) in Inheritance application:

The super functions can be applied in four ways and they are shown in the following diagram.



Rule(1) and (3) :

When ever we want to call default constructor of super class from derived class constructors, we user super(). using super() in the context of derived class constructors is optional

Rule(2) and (4):

When ever we want to call parameterized constructor of super class from derived class constructors we use super(...). Using super(...) in the content of derived class constructors

is mandatory(Since the super class can contain n number of parameterized constructors).

Example for Rule(1):

```
class BCR1
{
	private int a;
	BCR1() //(2)
	{
		System.out.println("I am from base class default constructor");
		a=10;
		System.out.println("Value of a = "+a);
	}
}

class DCR1 extends BCR1
{
	int b;
	DCR1()
	{
		super(); // optional
		System.out.prinltn("I am from deried class default constructor");
		b=20;
		System.out.println(" Valur of b from dc = "+b);
	}
}

class Idemo6
{
	public static void main(String arg[])
	{
		DCR1 dc = new DCR1();
	}
}
```

*** Example of Rule2:**

```
class BCR2
{
	private int a;
	BCR2(int k) //(2)
	{
		System.out.println("I am from base class parametrized constructor");
		this.a=k;
		System.out.println("Value of a = "+this.a);
	}
}

class DCR2 extends BCR2
{
	int b;
	DCR2(int x,int y)
	{
		super(x);  // Mandatory
		System.out.prinltn("I am from deried class parametrized constructor");
		this.b=y;
		System.out.println(" Valur of b from dc = "+this.b);
	}
}

class Idemo7
{
	public static void main(String arg[])
	{
		DCR2 dc = new DCR2(5,9);
	}
}
```

Example of Rule3:

```
class BCR3
{
	private int a;
	BCR3() //(2)
	{
		System.out.println("I am from base class default constructor");
		a=10;
		System.out.println("Value of a = "+a);
	}
}

class DCR3 extends BCR3
{
	int b;
	DCR3(int b)
	{
		super();  // optional
		System.out.prinltn("I am from deried class parametrized constructor");
		this.b=b;
		System.out.println(" Valur of b from dc = "+this.b);
	}
}

class Idemo8
{
	public static void main(String arg[])
	{
		DCR3 dc = new DCR3(1);
	}
}
```

Example of Rule4:

```
class BCR4
{
	private int a;
	BCR4(int a) //(2)
	{
		System.out.println("I am from base class parametrized constructor");
		this.a=a;
		System.out.println("Value of a = "+this.a);
	}
}

class DCR4 extends BCR4
{
	int b;
	DCR4()
	{
		super(2); // mandatory
		System.out.prinltn("I am from deried class default constructor");
		b=20;
		System.out.println(" Valur of b from dc = "+b);
	}
}

class Idemo9
{
	public static void main(String arg[])
	{
		DCR4 dc = new DCR4();
	}
}
```

***) Write a java program which illustrate the concept of this(), this(...) and super(), super(...).**

```
class Abc
{
	Abc() // (6)
	{
		System.out.println("I am from Abc class default constructor");
	}
	Abc(int a) // (5)
	{
		this();
		System.out.println("I am from Abc class parametrized constructor");
	}
}

class Def extends Abc
{
	Def() // (3)
	{
		this(100);
		System.out.println("I am from Def class default constructor");
	}
	Def(int b) //(4)
	{
		super(1121);
		System.out.println("I am from Def class parametrized constructor");
	}
}

class Ghi extends Def
{
	Ghi() // (1)
	{
		this(10);
		System.out.println("I am from Ghi class default constructor");
	}
	Ghi(int b) //(2)
	{
		super();
		System.out.println("I am from Ghi class parametrized constructor");
	}
}

class Exe
{
	public static void main(String args[])
	{
		Ghi obj = new Ghi();  // goes to (1)
	}
}
```

Polymorphism:
============
def:- The process of representing 'one form' in multiple forms is known as polymorphism.
The principle of polymorphism is implemented in java by the concept of 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.

Polymorphisms are divided into two types
they are
	1) Static or compile time Polymorphism.
	2) Runtime or Dynamic Polymorphism

1) Static or compile time Polymorphism:

A static polymorphism is one in which methods are binded with an object at compile time.

The disadvantage of static polymorphism is that "utilization of the resource(memory space)" are very poor(in c++ static polymorphism can be implemented by function overloading and operator overloading).

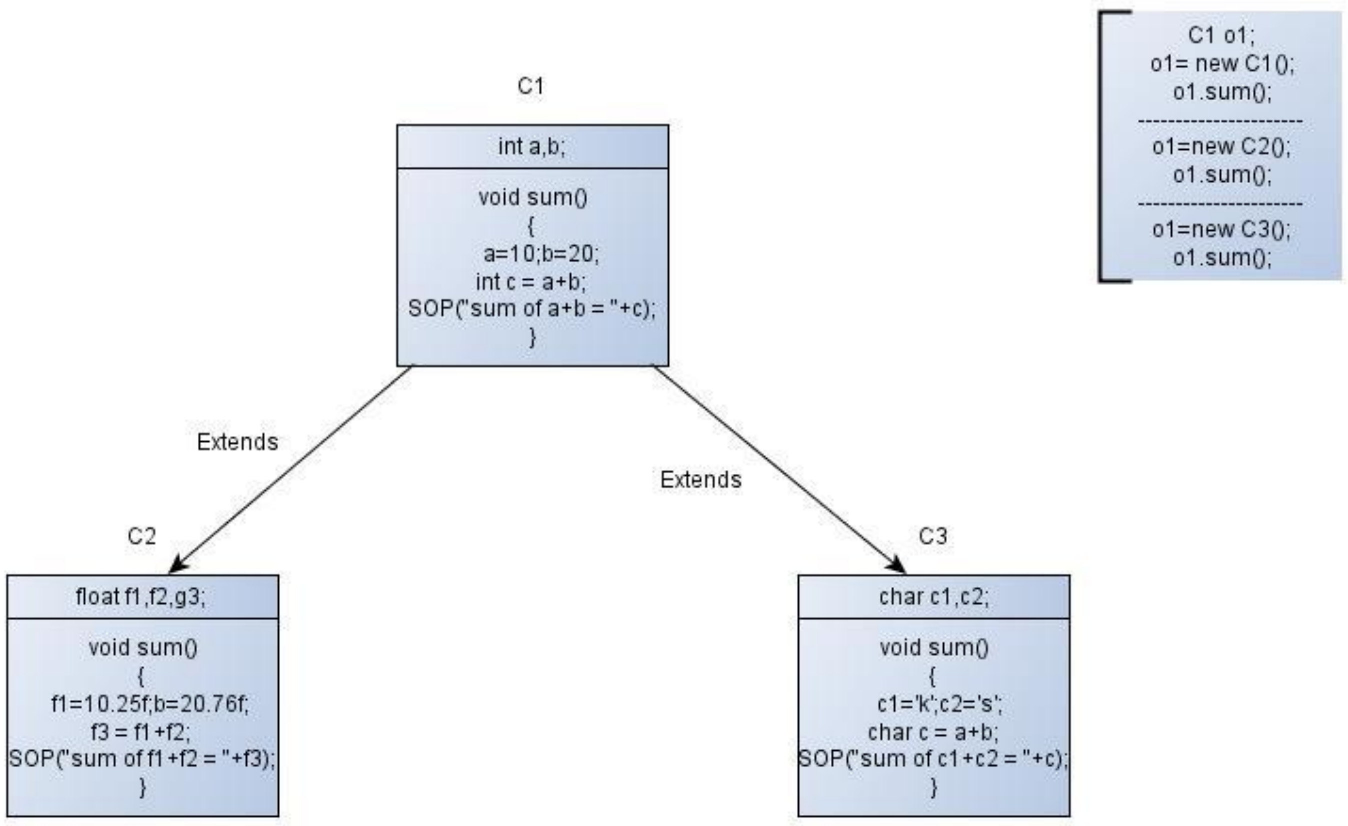
In Java we never follow static polymorphism that is java never allows static polymorphism. hence java always follows runtime polymorphism.

2) Runtime or Dynamic Polymorphism:

A Runtime polymorphism is one in which methods are binded with an object at runtime(Runtime polymorphism implemented in java by the concept of Abstract classes) where as in c++ it is implemented by 'Virtual function'

Advantage: The advantage of Runtime polymorphism is that Utilization of the resources effectively . hence all the real time applications in java follows only runtime polymorphism but not static polymorphism.

Ex: Let us consider the following hierarchy chat



Note: In order to develop any application we follow the concept of polymorphism and to execute the polymorphism application we follow another OOPs(Object Oriented Programming) principle called dynamic binding but not 'plain old logic of java'.

Dynamic Binding:

It is always says that don't create an object(s) of derived class(es) but create an object of single base class.

Dynamic binding always used for executing polymorphic application.

The advantage of dynamic binding id that 'utilization of the resources are very effective".

def:- Dynamic Binding is a process of binding an appropriate version(overridden methods) of the derived classes which is inherited from base class
with base class object.

Why we are using dynamic binding to execute polymorphic applications, JVM internally considers the following 2 points

1) What type of Object?
2) What type of Address object contains.

*--> Let us consider the following code segment to execute the problem which is illustrated in the above diagram.

(1) C1 o1;
(2) o1 = new C1();
(3) o1.sum();
(4) o1 = new C2();
(5) o1.sum();
(6) o1 = new C3();
(7) o1.sum();

--> In the above statements
In statement numbers (2), (4), (6) the object o1 contains address of different classes(C1, C2, C3) and hence Object o1 is known as polymorphic Object.
--> In the above diagram the methods on ia actually in one form (original form) and further it is implemented or defined in multiple forms. hence
sum() method is known as polymorphic method.
--> In the statement numbers (3), (5), (7) , o1.sum(); is actually ine one form but it gives different sum()s depends on type of address presenting in o1 object

Hence the statement o1.sum() is known as polymorphic statement.
" All the real time applications must be developed by polymorphism along with method overriding and that application is executed with dynamic binding"

Abstract Class:
--> Each and very program in java can be written with the concept of classes.
--> In java programming we have 2 types of classes they are
1) Concrete classes
2) Abstract Classes

--> A concrete class is one which contains fully defined methods. defined methods of a class are also known as implemented methods or concrete
Methods.
Ex:
```
class C1
{
      void f1()
      {
            System.out.println("F1 Implemented");
      }
      void f2()
      {
            System.out.println("F2 Implemented");
      }
}
```
The class C1 is called concrete and who's object can be created directly.
```
C1 o1 = new C1(); // Direct creation
      o1.f1();
      o1.f2();
```

--> An 'Abstract' class is one which contains some defined methods and some undefined methods.
Once if a class belongs to an abstract then who's object can not be created directly 'but we can create Indirectly'.

Def of Abstract method:

An abstract method is one which does not contain any body /definition . but there exists only prototype declaration.

--> In order to make undefined methods as abstract we use a keyword called 'abstract'.

Syntax for abstract methods:

abstract return type methodName(list of formal parameters);

ex: abstract void f1(int a);
abstract void sum();
abstract void op(int a, int b);

*--> abstract methods always makes us to understand the following

1) what a method can do? but
2) It never say 'How a method can be done?'

--> If a class contains any abstract methods then that class is known as abstract class. In order to make the class as abstract ,
we use a keyword 'abstract' before the class definition.

Syntax for Abstract classes:

```
abstract class <ClassName>
{
      // variables
      abstract return type methodName(list of formal prams);
      // we may have some implemented methods too
}

Ex: abstract class Operation
{
      abstract void op(int a, int b);
}
```

--> In above class operation is called abstract and who's object can not be created directly but we can create its object indirectly.

Operation opobj = new Operation() // In valid

Importance of abstract classes:

--> when ever a java application contains commonly reused variables and methods, it is highly recommended to keep then into abstract classes but
not in concrete classes.

--> abstract classes in java are always making use of polymorphism along with method overriding(for development), along with
dynamic binding(for execution). Abstract classes always improves the performance of an application by making use of 'less amount of memory space'.

--> Abstract class features are always reusable , since abstract classes always contain 'common reusable features'

--> Abstract class definitions are not belongs to 'final' since they are always reusable.

def: An object of abstract class can not be created directly but we can create indirectly.

An Object of abstract class = an Object of its subclass
or
An object of abstract class = An object of that class which extends an Abstract class.

Ex: Operation op; // Object declaration
op = new iSum();// indirect referencing of a abstract class

Note :- An object of abstract class can be declared but it can not be 'referenced' directly. And it can be referenced indirectly

Q) Write a java program to compute sum of 2 integers and floats by using abstract classes.

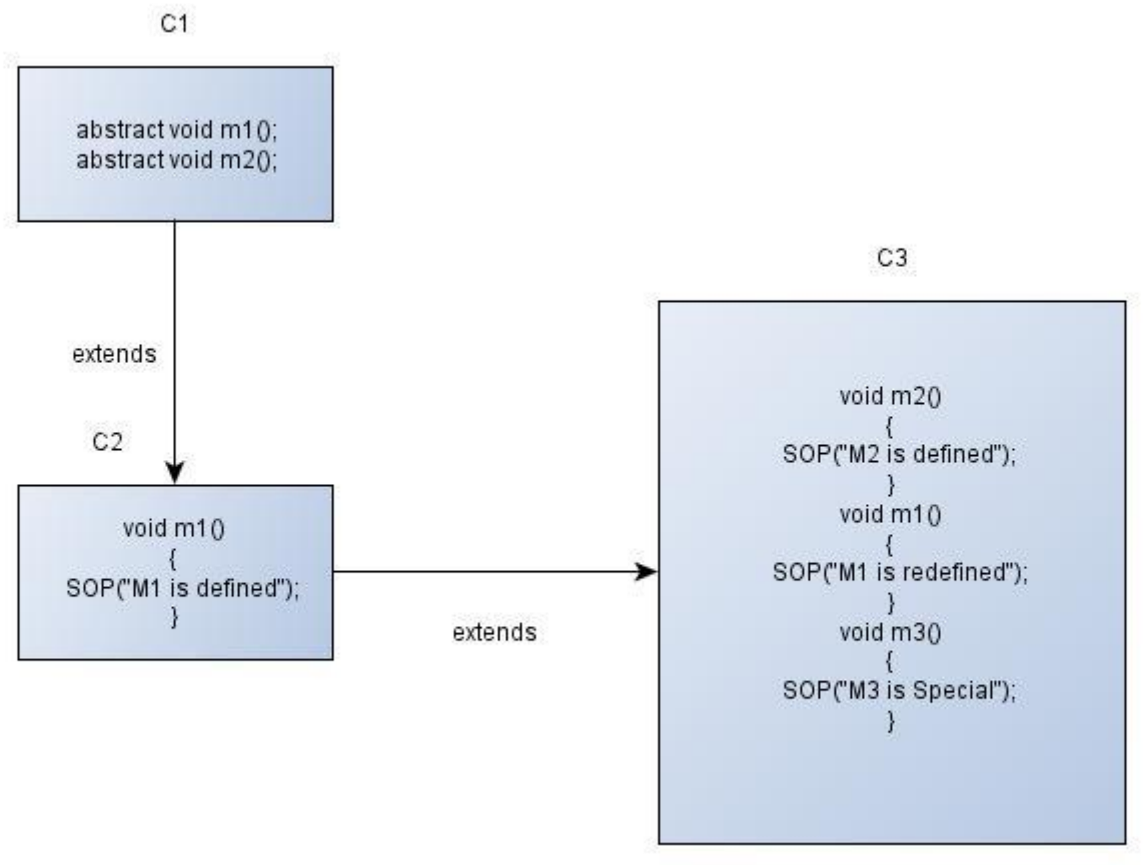
```
abstract class SOperation
{
	abstract void sum();
	void mul()
	{
		System.out.println("I am in mul method of abstract");
	}
}

class Isum extends SOperation
{
	int a,b,c;
	void sum()
	{
		a = 10;
		b = 20;
		c = a+b;
		System.out.println("Sum of Integers = "+c);
	}
}

class Fsum extends SOperation
{
	int f1,f2,f3;
	void sum()
	{
		f1 = 10.25f;
		f2 = 0.95f;
		f3 = f1+f2;
		System.out.println("Sum of floats = "+f3);
	}
}

	class AbstractDemo
	{
		public static void main(String args[])
		{
			SOperation op;

			op = new Isum();
			op.sum();
			op.mul();

			op = new Fsum();
			op.sum();
			op.mul();
		}
	}
```

Note : - abstract methods should not be static.

Abstract base classes and Abstract derived classes:
--> An abstract base class is one which contains physical representation of abstract methods.

--> An abstract derived class is one which contains logical representation of abstract methods. which are inherited from abstract base class.
Q) When a derived class becomes abstract?
Ans: When ever a derived class inherits 'n' number of abstract methods from abstract base class and if a derived class is not defining at least one
abstract method, then that derived class is known as abstract derived class. And who's definition must be made abstract by using abstract keyword.
--> If the derived class define all 'n' number of abstract methods then the derived class is known as concrete sub class.
--> Either with respect to abstract base class or with respect to abstract derived class we can not create and object directly but we can create indirectly.
--> Abstract base classes and abstract derived classes are always reusable by sub classes.
--> Either an object of ABC(Abstract Base Class) or an object of ADC(Abstract Derived Class) contains details about those methods which are available
in the smae class but these objects does not contains details about those methods which are specially defined in derived classes.

***) Write a java program for implementing the bellow referred diagram.**



```
abstract class C1 // ABC - Abstract Base Class
{
	abstract void m1();
	abstract void m2();
}

abstract class C2 extends C1 // ADC - Abstract Derived Class
{
	void m1()
	{
		System.out.println("m1-defined in C2");
	}
	void m4()
	{
		System.out.println("m4-Special in C2");
	}
}

class C3 extends C2 // Concrete class
{
	void m2()
	{
		System.out.println("m2-defined in C3");
	}
	void m1()
	{
		super.m1(); // will call super.m1() class from derived class C2
		System.out.println("m1-redefined in C3");
	}

	void m3()
	{
		System.out.println("m3-Special in C3");
	}
}

class AbstractDemo1
{
	public static void main(String args[])
	{
		System.out.println(" With respect to C3(CC Concrete Class) class");
		C3 obj3 = new C3();
		obj3.m1();
		obj3.m2();
		obj3.m3();
		obj3.m4();
		System.out.println(" With respect to C2(ADC Abstract Derived Class) class");
		//C2 obj2 = new C2(); // invalid since C2 is abstract
		C2 obj2 = new C3();
		obj2.m1();
		obj2.m2();
		//obj2.m3();  // Invalid , since m3() does not exists in c2 class.
		obj2.m4();

		System.out.println(" With respect to C1(ABC Abstract Base Class) class");
		//C1 obj1 = new C1(); // invalid since C1 is abstract
		C1 obj1 = new C3();
		obj1.m1();
		obj1.m2();
		//obj1.m3();  // Invalid , since m3() does not exists in c1 class.
		//obj1.m4(); // Invalid , since m4() does not exists in c1 class.
	}
}
```

Q) Can we make a concrete class as an abstract class.
Ans : A concrete class can be made as abstract class in the following situations

1) If a concrete class contains the defined methods with 'null' body. then calling the null body method with respect to concrete class object is of no use and we never get any result, hence an object of the class which contains null body method can not be created. Hence make this class definition has abstract.

```
	Ex: abstract class Father
		{
			void goc(){} // Null body
		}
```

Def of null body method:

A 'null' body method is one which does not contains any block of statements.

2) If a concrete class contains set of defined methods with block of statements and if they are not belongs to any use, then such type of class definition is recommended to make as abstract.

```
            Ex: abstract class SOperation
                        {
                              void sum()
                              {
                                    System.out.println("I  am  from  SOperation  sum  I  am  not
doing any sum");
                                    System.out.println("Please override me");
                              }
                        }
```

***) Write a java program which illustrate the concept of concrete abstract class.**

```
            abstract class Father
                  {
                        void goc(){} // Null body
                  }
                  class Son extends Father
                  {
                        void goc()
                        {
                              System.out.println("Going to college for 6 days");
                        }
                  }

                  class AbstractDemo2
                  {
                        public static void main(String args[])
                        {
                              // Father fo = new  Father(); // invalid since it is abstract
class
                              Father fo1 = new Son();
                              fo1.goc();
                              }
                  }
```

Interfaces:

The concept of interfaces is used for developing user defined defined/programmer defined data types.

--> If a derived class wants to inherit multiple abstract methods from multiple abstract classes, then our derived class needs to extend multiple abstract class names which invalid in java. since java does not support the concept of multiple inheritance either through concrete classes are through abstract classes, but the concept of 'interfaces'. i.e one class can obtain the features from multiple interfaces at a time.

--> Hence in java programming scope of interfaces is more than abstract classes.

def--> An interface is a construct which is containing commonly reusable variables and commonly reusable methods.

An interface is a construct which is containing
public static final XXX variables; (XXX represents data type , variable name and variable value) and
public abstract methods();
or
An interface is collection of purly undefined or abstract methods.

Syntax for defining an interface:

```
interface <interfaceName>
{
       variable declaration cum initialization;
       methods declaration;
}
```

--> 'interface' is a keyword which is used for developing user defined defined/programmer defined data types.
--> <interfaceName> represents a java valid identifier/ variable name treated as a name of interface.
--> Interface names are always used for creating objects and object of an interface can not be created directly but we can create indirectly since interface is containing purely abstract methods.
--> Variable declaration in the interface represents the type of variables we use, all the variables of an interface must be initialized (other wise we will get compile time error), since they ment for commonly reusable. Hence all the variables of an interface are by default belongs to

'public static final XXX variables;'

All the variables of an interface must be accessed with respect to interface name , since they belongs to static.
--> Methods declaration represents abstract methods which are used for performing common operation. In order to make undefined methods of interface as abstract explicitly, we need not to use 'abstract' keyword and methods of interface are universally reusable and hence we need to write 'public' keyword explicitly.

'All the methods of interface are by default belongs to public abstract methods();'

--> Finally the definition of interface contains universal commonly reusable features.
Ex: define an interface i1 with commonly reused variables a,b with 10 and 20 respectively and commonly reused methods m1 and m2.

```
interface i1
{
       int a= 10;
       int b = 20;
       void m1();
       void m2();
}
```

After compiling the above program the variables a,b are belongs to 'public static final' and methods m1 and m2 belongs to public abstract.

Consequences :
--> interface definition is by default abstract.
--> All the features of interfaces are commonly reusable and hence interface should not belongs to 'final'.
--> The definition of interfaces should not contain any constructors since,

i) The variables of interface are already initialized (note : constructors are used for placing our own values).

ii) Generally constructors are containing definitions and defined things are not allowed in interfaces.

--> If a program is containing only interfaces with out having any class, such type of of program can not be executed but it can be compiled, if an interface program compiled (javac i1.java) the we get an intermediate file with an extension .class'(i1.class).
--> Interface definitions does not contain main() method, since main() must be defined and it belongs to specific purpose.

Inheriting the features of interfaces into classes:
In java programming we have 3 syntaxes (ways) to reuse the features of interfaces.

Syntax1:-

```
	[abstract] class <className> implements <interface1>, <interface2>, ......<interfacen>
			{
				variable declaration;
				method definition/declaration;
				}
```

--> Here <className> represents name of the derived class and <interface1>, <interface2>, ...<interfacen> represents list of interface names.
--> 'implements' is a keyword used for inheriting the features of interface(s) and it improved the functionality of derived classes.
--> The keyword 'extends' is always used for extending the functionality of only one base class , where as the implements keyword implementing the functionality of multiple base interfaces, that is once class can extends only one class, where as one class can implements more than one interface.
*--> If a derived class is inheriting 'n' number of abstract methods form interface(s) and if the derived class is not defining at least one abstract method the derived class is known as abstract and whose definition must be made as abstract by using abstract keyword.

Ex)

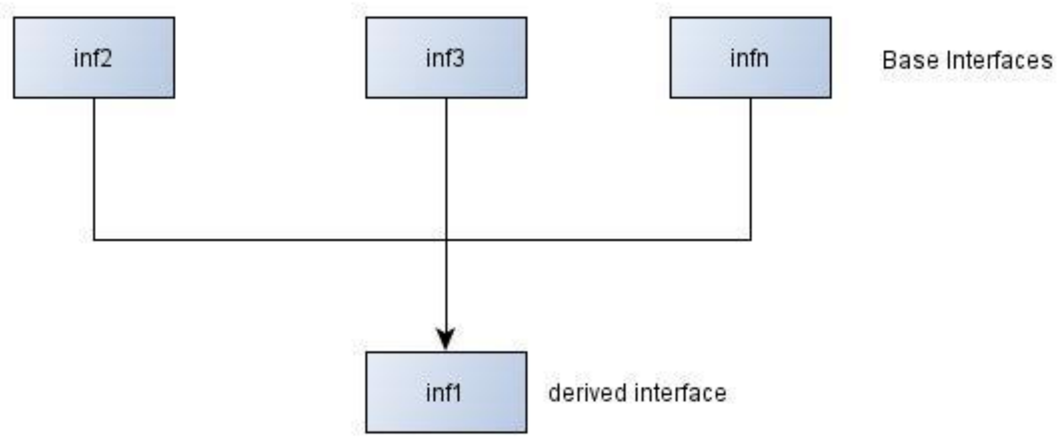
```
	interface I1
	{
		void m1();
	}

	class C1 implements I1
	{
		public void m1()
		{
			System.out.println("M1 is defined");
		}
	}

	class Idemo
	{
		public static void main(String args[])
		{
			System.out.println("With respect to C1");
			C1 o1 = new C1();
			o1.m1();

			System.out.println("With respect to I1");
			//I1 o2 = new I1(); // error invalid

			I1 o2;
			o2 = new C1();
			o2.m1();
		}
	}
```

--> An object of an interface can not be created directly but we can create indirectly.
An Object of interface = An object of its sub class
	or
An object of interface = An object of that class which is implemented an interface.

Syntax2(Interface Inheritance):
Interface inheritance is the process of obtaining the features of interface(s) to another interface.

syntax:
```
interface <inf1> extends <inf2>,<inf2>....<intn>
		{
```

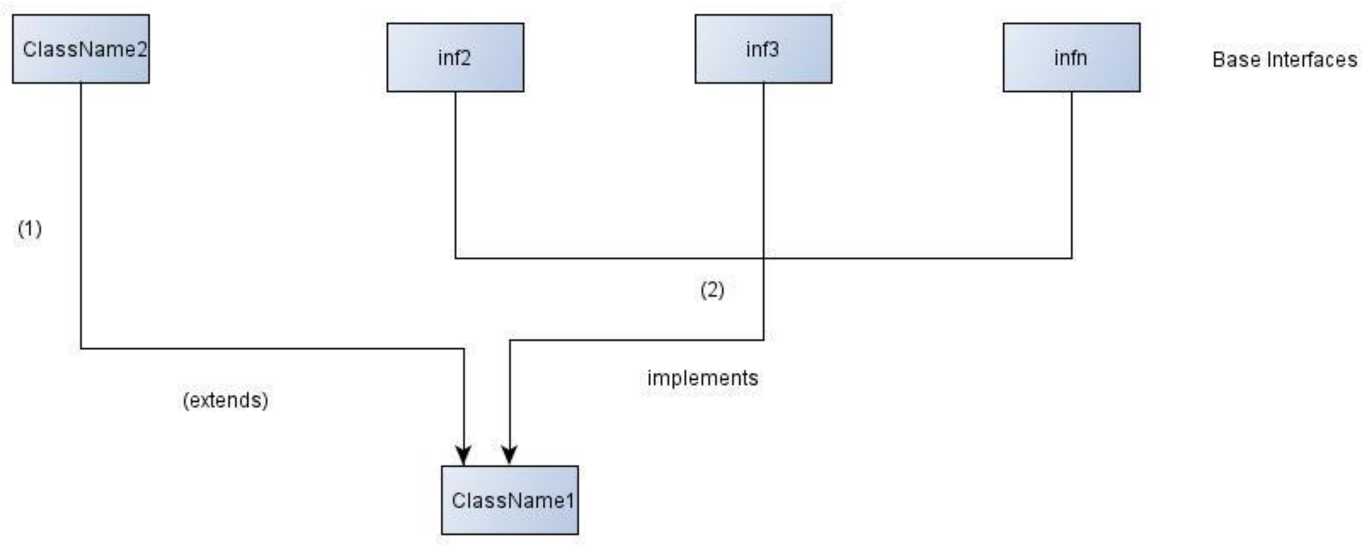
```
                        variable declaration cum initialization;
                        methods declaration;
                  }
```



--> In the above syntax <inf1> represents name of the derived interface and <inf2>, <inf3>, ... <infn> represents list of base interface names.
--> Here extends keyword is used for inheriting the features of interface(s) to another interface and it improves the functionality of derived interface.
****--> One class can 'extends' only one class where as one 'interface' can 'extends' either one or more than one 'interface'.

Syntaxt3:

```
                            (1)                 (2)
[abstract] class <className1> extends <className2> implements <inf1>,<inf2>,...<infn>
            {
                  variable declaration;
                  methods definition/declaration;
            }
```



--> In the above syntax <className1> represents name of derived class.
--> <className2> represents name of base class.
--> <inf1><inf2>,...<infn> represents list of base interfaces.
--> When ever we use both 'extends' and 'implements' keywords in a single java program, it is mandatory for the java programmer to use 'extends' keyword
first and later we must use 'implements' keyword(if the order is reversed we get compile time error).

```
ex: class C1 extends C2 implements i1, i2, i3
{
       ===
       ===
}
```

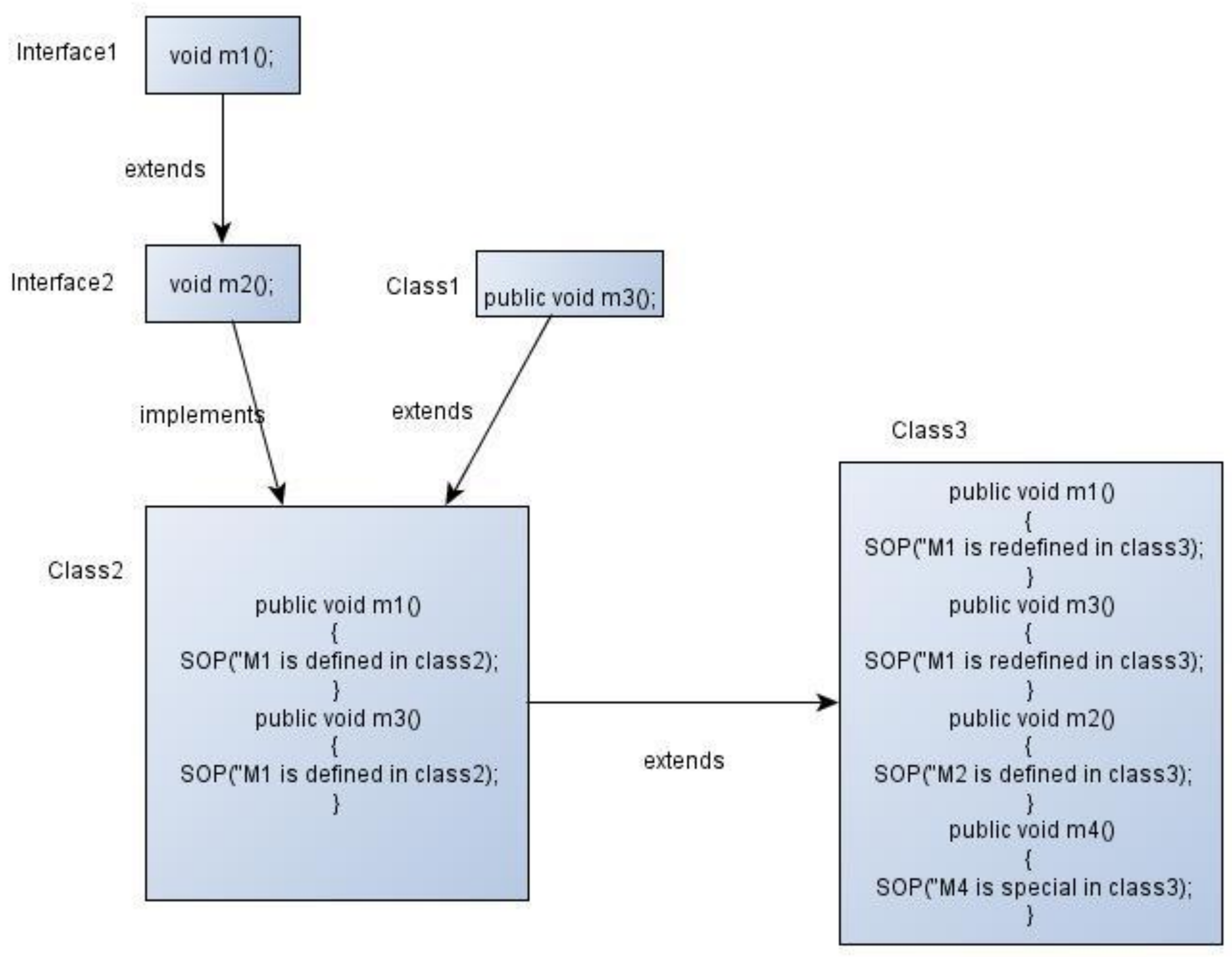
FAQ:

--> One class can extends only one class
--> One interface can extends one or more interfaces.
--> One class can extends one class and implements multiple interface.
--> One interface can not extends a class , or once class can not extends an interface.
--> Top most base class by default extends 'Object' class.
we have a classes C1, C2, C3,

Object (Object is predefined class provided by sun micro systems , every class in java by default extends Object class)
$\downarrow$
C1
$\downarrow$
C2
$\downarrow$
C3

---> Top most base interface can not extends predefined 'Object' class.(Invalid)
--> For each and every class in java there exits an implicit Super class known as java.lang.Object class but for each and every interface in java there is no super class available.

Ex) Implements the following diagram by writing the java program



```java
interface Interface1
{
	void m1();
}

interface Interface2 extends Interface1
{
	void m2();
}

abstract class Class1
{
	abstract void m3();
}

abstract class Class2 extends Class1 implements Interface2
{
	public void m1()
	{
		System.out.println("M1 is defined in Class2");
	}
	public void m3()
	{
		System.out.println("M3 is defined in Class2");
	}
}

class Class3 extends Class2
{
	public void m1()
	{
		System.out.println("M1 is re-defined in Class3");
	}
	public void m3()
	{
		System.out.println("M3 is re-defined in Class3");
	}
	public void m2()
	{
		System.out.println("M2is defined in Class3");
	}
	public void m4()
	{
		System.out.println("M4 is special Method in Class3");
	}
}

class Idemo2
{
	public static void main(String args[])
	{
		System.out.println("***********************");
		System.out.println("With respect to Class3");
		Class3 cobj3 = new Class3();
		cobj3.m1();
		cobj3.m2();
		cobj3.m3();
		cobj3.m4();
		System.out.println("***********************");
		System.out.println("With respect to Class2");
		Class2 cobj2 = new Class3();
		cobj2.m1();
		cobj2.m2();
		cobj2.m3();
		//cobj2.m4();
		System.out.println("***********************");
		System.out.println("With respect to Class1");
		Class1 cobj1 = new Class3();
		//cobj1.m1();
		//cobj1.m2();
		cobj1.m3();
		//cobj1.m4();
		System.out.println("***********************");
		System.out.println("With respect to Interface2");
		Interface2 iobj2 = new Class3();
		iobj2.m1();
		iobj2.m2();
		//iobj2.m3();
		//iobj2.m4();
		System.out.println("***********************");
		System.out.println("With respect to Interface1");
		Interface1 iobj1 = new Class3();
		iobj1.m1();
		//iobj1.m2();
		//iobj1.m3();
		//iobj1.m4();

	}
}
```

PACKAGE:

A package is a collection of classes, interfaces and sub-packages. A sub-package in turns divides into classes, interfaces, sub-sub-packages, etc.

Learning about JAVA is nothing but learning about various packages. By default one predefined package is imported for each and every JAVA program and whose name is java.lang.*.

Whenever we develop any java program, it may contain many number of user defined classes and user defined interfaces. If we are not using any package name to place user defined classes and interfaces, JVM will assume its own package called NONAME package.
In java we have two types of packages they are

1. predefined or built-in or core packages.
2. user or secondary or custom defined packages.

PREDEFINED PACKAGES:
Predefined packages are those which are developed by SUN micro systems and supplied as a part of JDK (Java Development Kit) to simplify the task of java programmer.

NOTE: Core packages of java starts with java. (For example: java.lang.*) and Advanced packages of java starts with javax. (For example: java.sql.*)

TYPES of predefined packages:
As a part of J2SE we have nine predefined packages which are given in the following table:

Package name	Package description
java.lang.*	This package is used for achieving the language functionalities such as conversion of data from string to fundamental data, displaying the result on to the console, obtaining the garbage collector. This is the package which is by default imported for each and every java program.
java.io.*	This package is used for developing file handling applications, such as, opening the file in read or write mode, reading or writing the data, etc.
java.awt.* (abstract windowing toolkit)	This package is used for developing GUI (Graphic Unit Interface) components such as buttons, check boxes, scroll boxes, etc.
java.awt.event.*	Event is the sub package of awt package. This package is used for providing the functionality to GUI components, such as, when button is clicked or when check box is checked, when scroll box is adjusted either vertically or horizontally.
java.applet.*	This package is used for developing browser oriented applications. In other words this package is used for developing distributed programs. An applet is a java program which runs in the context of www or browser.
java.net.*	This package is used for developing client server applications.
java.util.*	This package is used for developing quality or reliable applications in java or J2EE. This package contains various classes and interfaces which improves the performance of J2ME applications. This package is also known as collection framework (collection framework is the standardized mechanism of grouping of similar or different type of objects into single object. This single object is known as collection object).
java.text.*	This package is used for formatting date and time on day to day business operations.
java.lang.reflect.*	Reflect is the sub package of lang package. This package is basically used to study runtime information about the class or interface. Runtime information represents data members of the class or interface, Constructors of the class, types of methods of the class or

	interface.
java.sql.*	This package is used for retrieving the data from data base and performing various operations on data base.

USER DEFINED PACKAGES:

A user defined package is one which is developed by java programmers to simplify the task of the java programmers to keep set of classes, interfaces and sub packages which are commonly used. Any class or interface is commonly used by many java programmers that class or interface must be placed in packages.

Syntax:

```
	package pack1[.pack2[.pack3……[.packn]…..]];
```

Here, package is a keyword which is used for creating user defined packages, pack1 represents upper package and pack2 to packn represents sub packages.

For example:

package p1;--> statement-1
package p1.p2; --> statement-2
The statements 1 and 2 are called package statements.

RULE:

Whenever we create user defined package statement as a part of java program, we must use package statement as a first executable statement.

NOTE: Whenever we develop any JAVA program it contains 'n' number of classes and interfaces. **Each and every** class and interface which are developed by the programmer **must belong to a package** (according to industry standards). If the **programmer is not keeping** the set of classes and interfaces in a package, **JVM will assume its own package** called **NONAME** package.

NONAME package **will exist only for a limited span of time until the program is completing.**

STEPS for developing a PACKAGE:

1. Choose the appropriate package name, the package name must be a JAVA valid variable name and we showed ensure the package statement must be first executable statement.
2. Choose the appropriate class name or interface name and whose modifier must be public.
3. The modifier of Constructors of a class must be public.
4. The modifier of the methods of class name or interface name must be public.
5. At any point of time we should place either a class or an interface in a package and give the file name as class name or interface name with extension .java

For example:

```
// Test.java
package tp;
public class Test
{
	public Test ()
	{
		System.out.println ("TEST - DEFAULT CONSTRUCTOR");
```

```
        }
        public void show ()
        {
                System.out.println ("TEST - SHOW");
        }
}
```

```
//ITest.java
package tp;
public interface ITest
{
        void disp ();
}
```

Syntax for compiling a package:

```
        javac –d . filename.java
```

For example:

```
        javac –d . Test.java
```

Here, -d is an option or switch which gives an indication to JVM saying that go to Test.java program take the package name and that package name is created as directory automatically provides no errors are present in Test.java. When Test.java is not containing any errors we get Test. class file and it will be copied automatically into current directory which is created recently i.e., tp (package name). The above program cannot be executed since it doesn't contain any main method.

How to use PACKAGE CLASSES and INTERFACES in another java program:

In order to refer package classes and interfaces in JAVA we have two approaches, they are using import statement and using fully qualified name approach.

Using import statement:
Import is a keyword which is used to import either single class or interface or set of classes and interfaces all at once.

Syntax -1:

```
        Import pack1 [.pack2 [.………[.packn]]].*;
```

For example:

```
        Import p1.*; ---1
        Import p1.p2.*; ---2
        Import p1.p2.p3.*; ---3
```

When statement 1 is executing we can import or we can access all the classes and interfaces of package p1 only but not its sub packages p2 and p3 classes and interfaces.

When statement 2 is executing we can import as the classes and interfaces of package p2 only but not p1 and p3 classes and interfaces.

When statement 3 is executing we can import as the classes and interfaces of package p3 only but not p1 and p2 classes and interfaces.

Syntax-2:

```
        Import pack1 [.pack2 [.…………[.packn]]].class name/interface name;
```

For example:

```
        Import p1.c1; ---4
        Import p1.p2.c3; ---5
```

When statement 4 is executing can import c1 class of package p1 only but not other classes and interfaces of p1 package, p2 package and p3 package.

Write a JAVA program which illustrates the usage of package classes?
Answer:
Import approach:
```
import tp.Test;
class PackDemo
{
	public static void main (String [] args)
	{
		Test t1=new Test ();
		t1.show ();
	}
};
```

When we compile the above program we get the following error "package tp does not exist". To avoid the above error we must set the classpath as., SET CLASSPATH = %CLASSPATH%;.;

This is the alternate technique for import statement:
```
		p1.c2 o2=new p1.c2 ();
		p1.p2.p3.c4 o4=new p1.p2.p3.c4 ();
		p1.p2.i3 o3=new p1.p2.p3.c4 ();
```

Fully qualified approach:
```
class PackDemo
{
	public static void main (String [] args)
	{
		tp.Test t1=new tp.Test ();
		t1.show ();
	}
};
```

NOTE:
1. Whenever we develop user defined packages, to use the classes and interfaces of user defined packages in some other program, we must set classpath before there usage.
2. In order to set the classpath for predefined packages we must use the following statement:
D:\core\set classpath=C: \Program Files\Java\jdk1.5.0\lib\rt.jar;.; [rt.jar contains all the .class files for the predefined classes which are supplied as a port of predefined packages by the SUN micro systems.]
When two classes or an interface belongs to the same package and if they want to refer those classes or interfaces need not to be referred by its package name.

For example:
```
// I1.java
// javac -d . I1.java
package ip;
public interface I1
{
	public void f1 ();
	public void f2 ();
};

// C01.java
// javac -d . C01.java
package cp;
public abstract class C01 implements ip.I1
```

```
{
	public void f1 ()
	{
		System.out.println ("F1 - C01");
	}
};

// C02.java
// javac -d . C02.java
package cp;
public class C02 extends C01
{
	public void f2 ()
	{
		System.out.println ("F2 - C02");
	}
	public void f1 ()
	{
		super. f1 ();
		System.out.println ("F1 - C02 - OVER RIDDEN");
	}
};


// PackDemo.java
// javac PackDemo.java
import ip.I1;
class PackDemo
{
	public static void main (String [] args)
	{
		System.out.println ("CP.C02");
		cp.C02 o2=new cp.C02 ();
		o2.f1 ();
		o2.f2 ();
		System.out.println ("CP.C01");
		cp.C01 o1=new cp.C02 ();
		o1.f1 ();
		o1.f2 ();
		I1 io;
		io=o1; //new cp.C02 ();
		io.f1 ();
		io.f2 ();
	}
};
```

In order to run the above program we must run with respect to package name.
1 javac –d . PackDemo1.java
2 java mp.PackDemo1 or java mp/PackDemo1

ACCESS SPECIFIERS in java:

In order to use the data from one package to another package or within the package, we have to use the concept of access specifiers. In JAVA we have four types of access specifiers. They are private, default (not a keyword), protected and public.

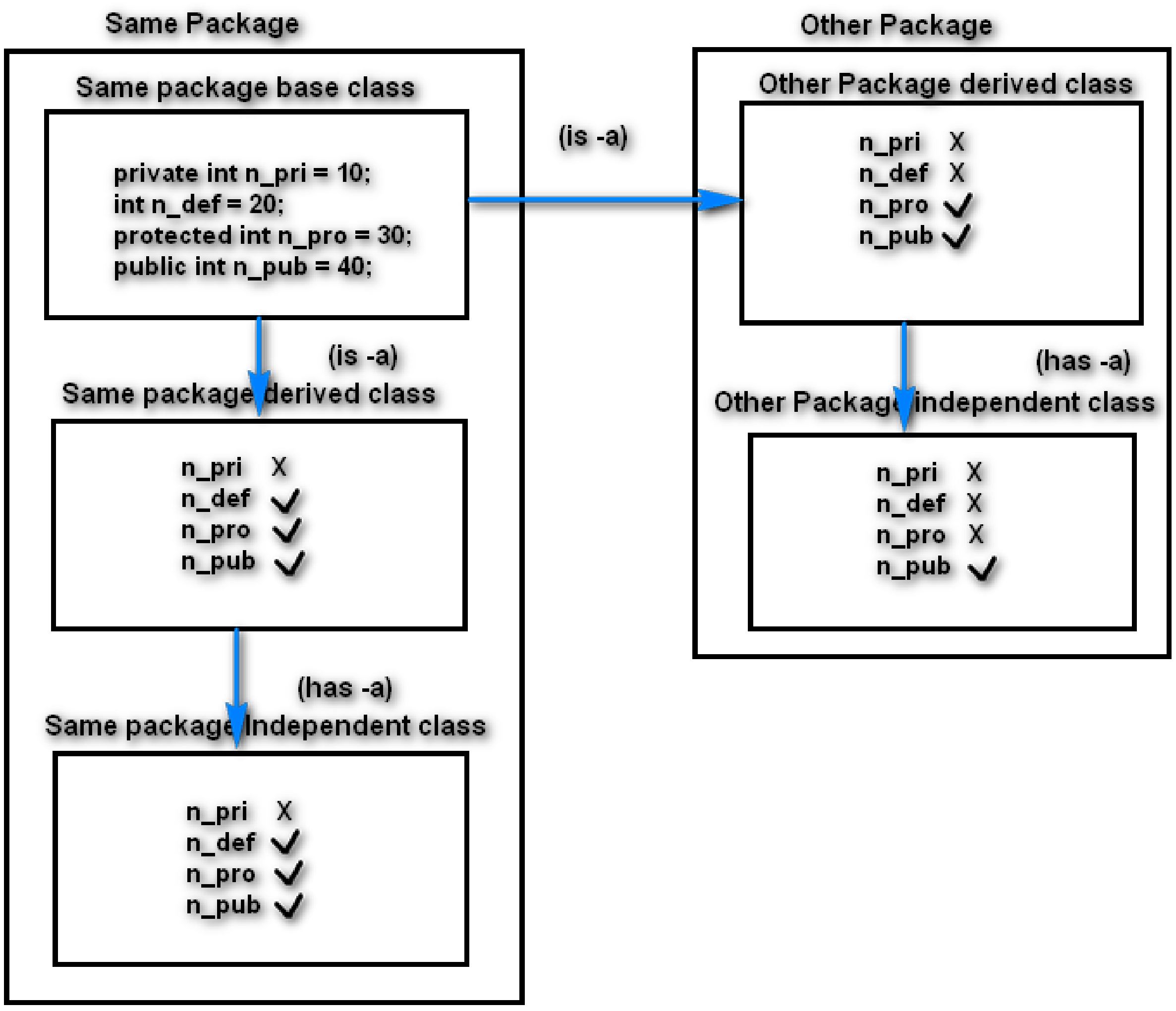
Access specifiers makes us to understand how to access the class, interface to interface and interfaces to class) and across the package (class to class, interface to interface and interfaces to class). In other words access specifiers represent the visibility of data or accessibility of data.

Syntax for declaring a variable along with access specifiers:

```
	[Access specifiers] [Static] [Final] data type v1 [=val1], v2 [=val2] …vn [=valn];
```

For example:

```
Public static final int a=10;
Protected int d;
Int x; [If we are not using any access specifier it is by default treated as default access
specifier]
private int e;
```



	Private	default	protected	public
Same package base class	√	√	√	√
Same package derived class	×	√	√	√
Same package independent class	×	√	√	√
Other package derived class	×	×	√	√
Other package independent class	×	×	×	√

NOTE:

1. Private access specifier is also known as native access specifier.
2. Default access specifier is also known as package access specifier.
3. Protected access specifier is also known as inherited access specifier.
4. Public access specifier is also known as universal access specifier.

Write a JAVA program which illustrates the concept of access rules?
Answer:

```
// Sbc.java
// javac -d . 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_PRIVATE = "+N_PRI);
		System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
		System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
		System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
	}
};

// Sdc.java
// javac -d . Sdc.java
package sp;
public class Sdc extends Sbc //(is-a relation & within only)
{
	public Sdc()
	{
		// System.out.println ("VALUE OF N_PRIVATE = "+N_PRI);
		System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
		System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
		System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
	}
};


// Sic.java
// javac -d . Sic.java
package sp;
public class Sic
{
	Sbc so=new Sbc(); // (has-a relation & within only)
	public Sic()
	{
		// System.out.println ("VALUE OF N_PRIVATE = "+so.N_PRI);
		System.out.println ("VALUE OF N_DEFAULT = "+so.N_DEF);
		System.out.println ("VALUE OF N_PROTECTED = "+so.N_PRO);
		System.out.println ("VALUE OF N_PUBLIC = "+so.N_PUB);
	}
};


// Odc.java
// javac -d . Odc.java
package op;
public class Odc extends sp.Sbc // (is-a relation & across)
{
	public Odc ()
```

```
        {
                // System.out.println ("VALUE OF N_PRIVATE = "+N_PRI);
                // System.out.println ("VALUE OF N_DEFAULT = "+N_DEF);
                System.out.println ("VALUE OF N_PROTECTED = "+N_PRO);
                System.out.println ("VALUE OF N_PUBLIC = "+N_PUB);
        }
};

// Oic.java
// javac -d . Oic.java
package op;
public class Oic
{
        sp.Sbc so=new sp.Sbc (); // (has-a relation & across)
        public Oic ()
        {
                // System.out.println ("VALUE OF N_PRIVATE = "+so.N_PRI);
                // System.out.println ("VALUE OF N_DEFAULT = "+so.N_DEF);
                // System.out.println ("VALUE OF N_PROTECTED = "+so.N_PRO);
                System.out.println ("VALUE OF N_PUBLIC = "+so.N_PUB);
        }
};


// ASDemo.java
// javac ASDemo.java

import sp.Sbc;
import sp.Sdc;
import sp.Sic;
class ASDemo
{
        public static void main (String [] args)
        {
                // import approach
                System.out.println ("WITH RESPECT TO SAME PACKAGE BASE CLASS");
                Sbc so1=new Sbc();
                System.out.println ("WITH RESPECT TO SAME PACKAGE DERIVED CLASS");
                Sdc so2=new Sdc();
                System.out.println ("WITH RESPECT TO SAME PACKAGE INDEPENDENT CLASS");
                Sic so3=new Sic();
                //fully qualified name approach
                System.out.println ("WITH RESPECT TO OTHER PACKAGE DERIVED CLASS");
                op.Odc oo1=new op.Odc();
                System.out.println ("WITH RESPECT TO OTHER PACKAGE INDEPENDENT CLASS");
                op.Oic oo2=new op.Oic();
        }
};
```

Nameless object approach:

Sometimes there is no necessity for the JAVA programmer to create an object with some name. In such situations we can use the concept of nameless object.

For example:

```
// named object approach
Test t1=new Test ();
t1.display ();
```

To convert the above statements into nameless object approach follow the following statements.

For example:

```
// nameless object approach
new Test ().display ();
```

EXCEPTIONAL HANDLING:

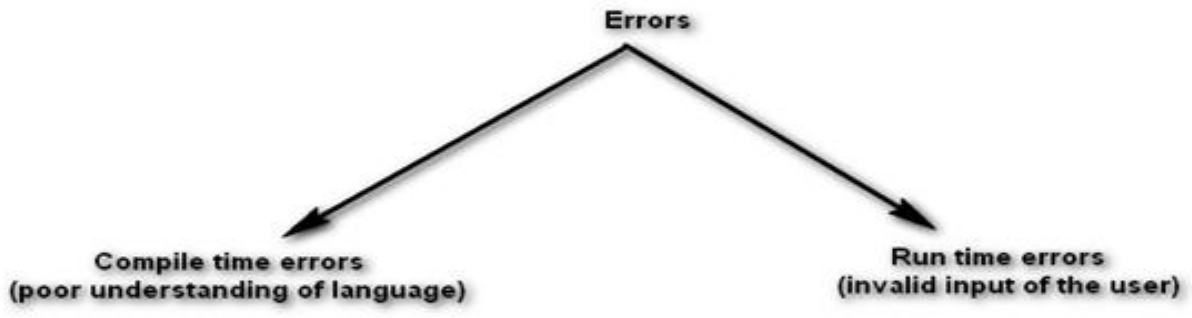
Whenever we develop any project in real time it should work in all circumstances (mean in any operation either in error or error free). Every technology or every programming language, if we use for implementing real time applications and if the end user commits a mistake then by default that language or technology displays system error messages which are nothing but run time errors.

- Run time errors in JAVA are known as exceptions.
- System error messages are those which are unable to understand by end user or client.
- User friendly messages are those which are understandable by end user or client.

Exceptional handling is a mechanism of converting system error messages into user friendly messages.

Errors are of two types. They are **compile time errors and run time errors**. **Compile time errors** are those which are occurring because of poor understanding of the language.
Run time errors are those which are occurring in a program when the user inputs invalid data.

The run time errors must be always converted by the JAVA programmer into user friendly messages by using the concept of exceptional handling.



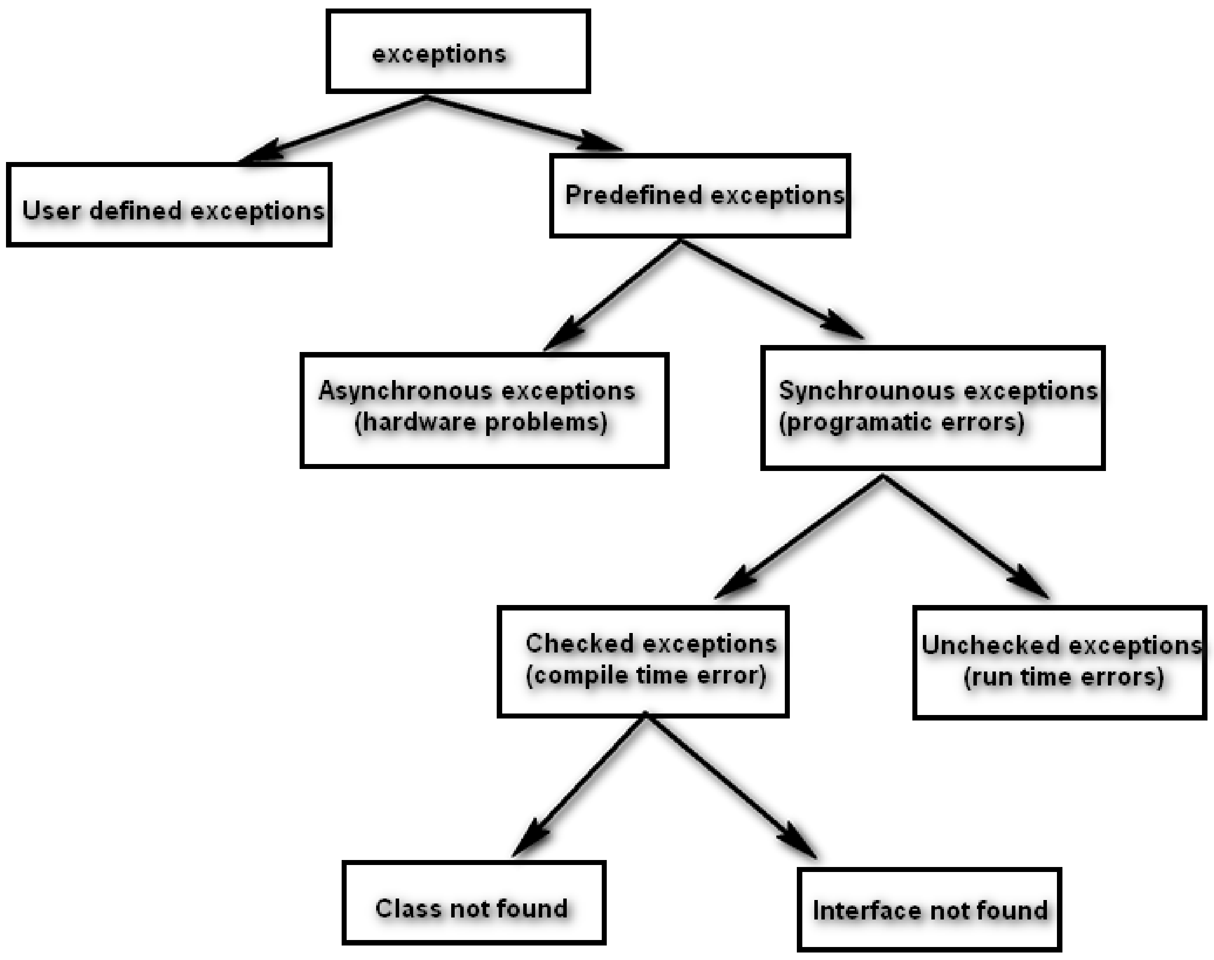
In JAVA run time environment, to perform any kind of operation or task or an action that will be performed with respect to either class or an interface.

Whenever we pass invalid data as an input to the JAVA program then JAVA run time environment displays an error message, that error message is treated as a class.

Types of exceptions:

In JAVA we have two types of exceptions they are **predefined** exceptions and **user** or **custom** defined exceptions.

1. **Predefined exceptions** are those which are developed by SUN micro system and supplied as a part of JDK to deal with universal problems. Some of the universal problems are dividing by zero, invalid format of the number, invalid bounce of the array, etc


exceptions
User defined exceptions
Predefined exceptions
Asynchronous exceptions (hardware problems)
Synchrounous exceptions (programatic errors)
Checked exceptions (compile time error)
Unchecked exceptions (run time errors)
Class not found
Interface not found

Predefined exceptions are divided into two types. They are **asynchronous exceptions** and **synchronous exceptions**.

Asynchronous exceptions are those which are always deals with hardware problems. In order to deal with asynchronous exceptions there is a predefined class called java.lang.Error. java.lang.Error class is the super class for all asynchronous exceptions.

Synchronous exceptions are one which always deals with programmatic errors. In order to deal with synchronous exceptions we must use a predefined class called java.lang.Exception class.

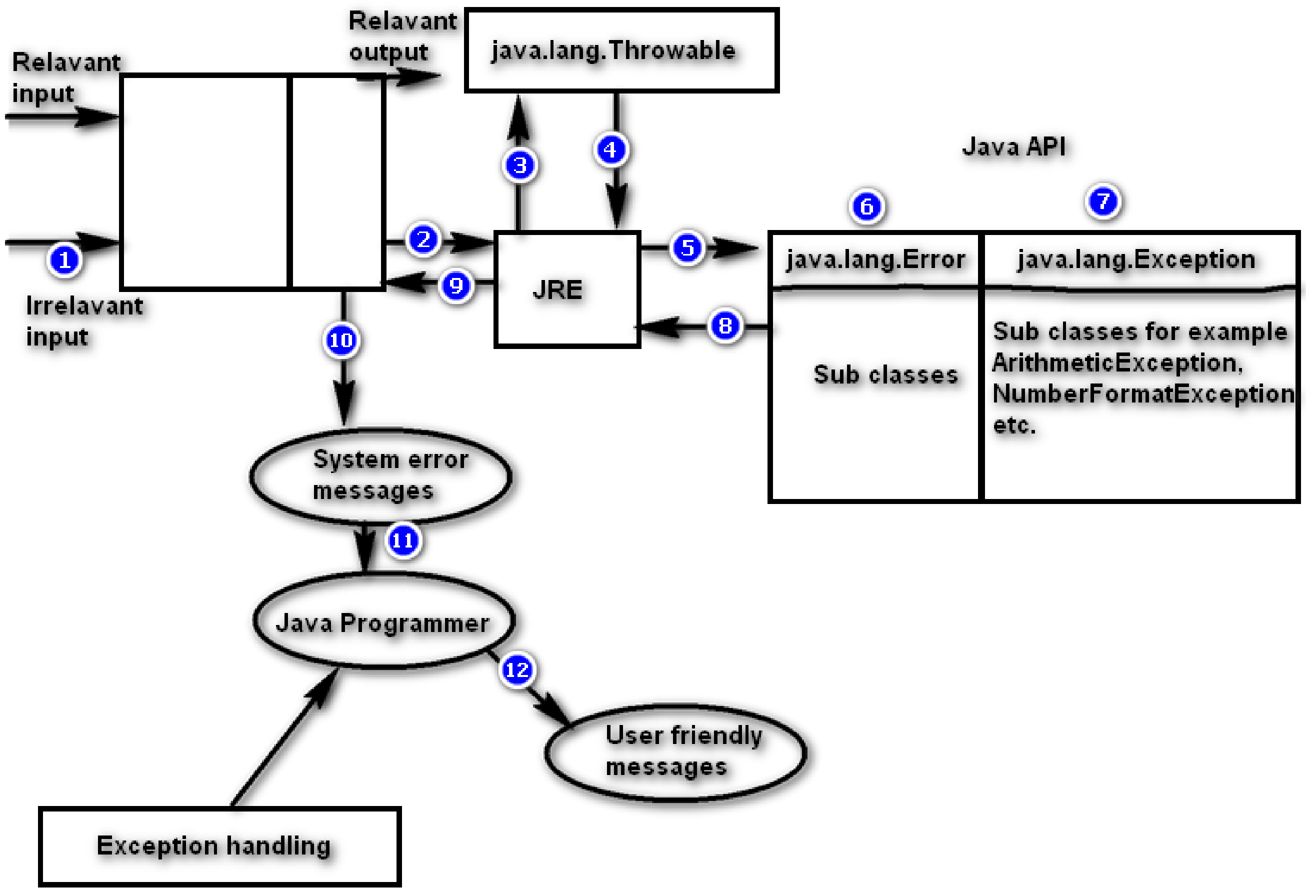
java.lang.Exception is the super class for all synchronous exceptions. Synchronous exceptions are divided into two types. They are checked exceptions and unchecked exceptions.

A checked exception is one which always deals with compile time errors regarding class not found and interface not found.

Unchecked exceptions are those which are always deals with programmatic run time errors such as ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException, etc

An exception is an object which 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 message.

How an EXCEPTION OCCURS in a java RUN TIME ENVIRONMENT:

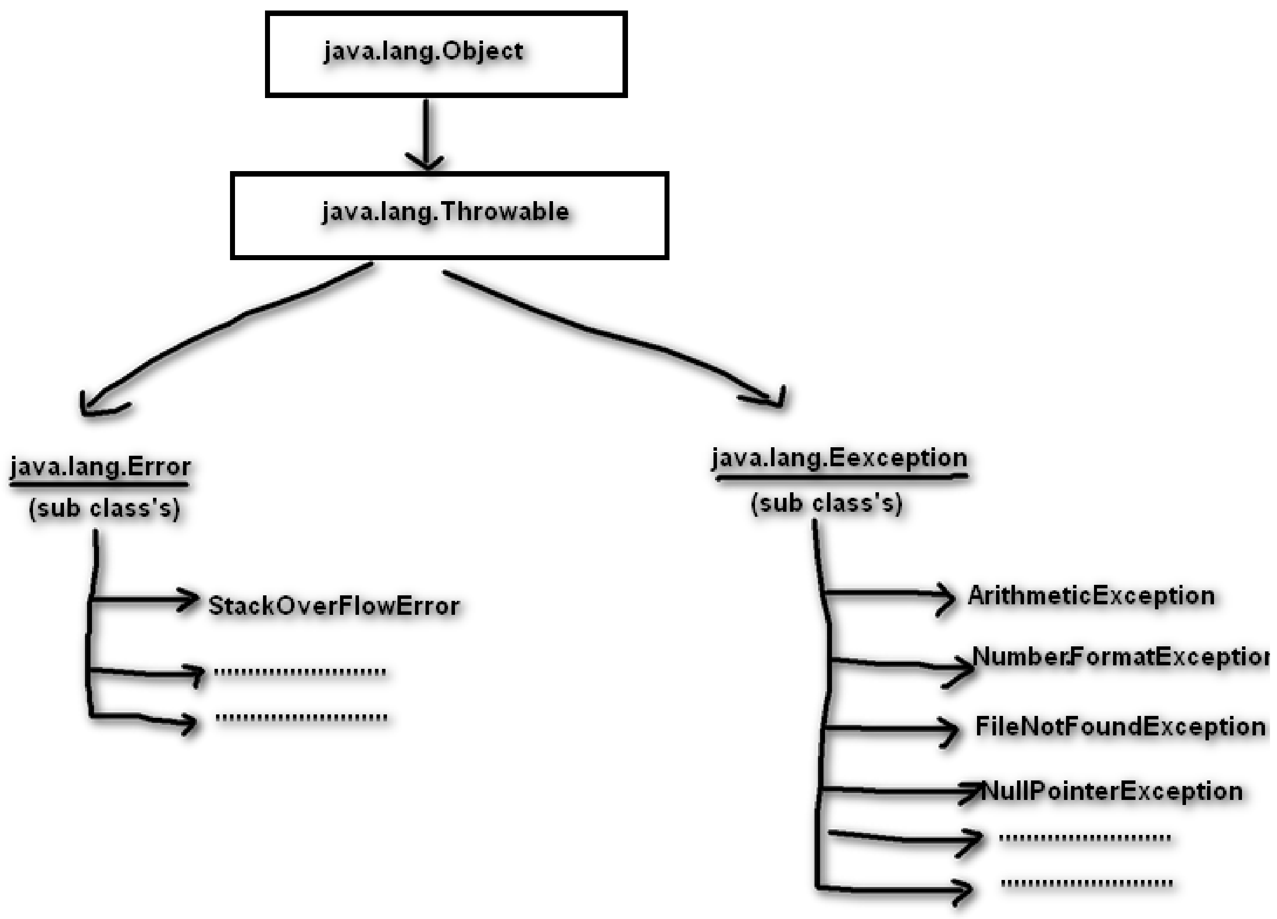


- Whenever we pass irrelevant input to a JAVA program, JVM cannot process the irrelevant input.
- Since JVM is unable to process by user input, hence it can contact to JRE for getting an appropriate exception class.
- JRE contacts to java.lang.Throwable for finding what type of exception it is.
- java.lang.Throwable decides what type of exception it is and pass the message to JRE.
- JRE pass the type of exception to JAVA API.
- From the JAVA API either java.lang.Error class or java.lang.Exception class will found an appropriate sub class exception.
- Either java.lang.Error class or java.lang.Exception class gives an appropriate exception class to JRE.
- JRE will give an appropriate exception class to JVM.
- JVM will create an object of appropriate exception class which is obtained from JRE and it generates system error message.
- In order to make the program very strong (robust), JAVA programmer must convert the system error messages into user friendly messages by using the concept of exceptional handling.

User friendly messages are understand by normal user effectively hence our program is robust.

HIERARCHY chart of EXCEPTIONAL HANDLING:

java.class.Object → super class for all JAVA class’s.



Syntax for exceptional handling:

In order to handle he exceptions in JAVA we must use the following keywords. They are try, catch, finally, throws and throw.

```
try
{
      Block of statements which are to be monitored by JVM at run time (or problematic
errors);
}
catch (Type_of_exception1 object1)
{
      Block of statements which provides user friendly messages;
}
catch (Type_of_exception2 object2)
{
      Block of statements which provides user friendly messages;
}
.
.
.
catch (Type_of_exception3 object3)
{
      Block of statements which provides user friendly messages;
}
finally
{
      Block of statements which releases the resources;
}
```

Try block:

1. This is the block in which we write the block of statements which are to be monitored by JVM at run time i.e., try block must contain those statements which causes problems at run time.
2. If any exception is taking place the control will be jumped automatically to appropriate catch block.
3. If any exception is taking place in try block execution will be terminated and the rest of the statements in try block will not be executed at all and the control will go to catch block.
4. For every try block we must have at least one catch block. It is highly recommended to write ‘n’ number of catch’s for ‘n’ number of problematic statements.

Catch block:

1. This is used for providing user friendly messages by catching system error messages.
2. In the catch we must declare an object of the appropriate execution class and it will be internally referenced JVM whenever the appropriate situation taking place.
3. If we write ‘n’ number of catch’s as a part of JAVA program then only one catch will be executing at any point.
4. After executing appropriate catch block even if we use return statement in the catch block the control never goes to try block.

Finally block:

1. This is the block which is executing compulsory whether the exception is taking place or not.
2. This block contains same statements which releases the resources which are obtained in try block (resources are opening files, opening databases, etc.).
3. Writing the finally block is optional.

For example:

```
class Ex1
{
	public static void main (String [] args)
	{
		try
		{
			String s1=args[0];
			String s2=args[1];
			int n1=Integer.parseInt (s1);
			int n2=Integer.parseInt (s2);
			int n3=n1/n2;
			System.out.println ("DIVISION VALUE = "+n3);
		}
		catch (ArithmeticException Ae)
		{
			System.out.println ("DONT ENTER ZERO FOR DENOMINATOR...");
		}
		catch (NumberFormatException Nfe)
		{
			System.out.println ("PASS ONLY INTEGER VALUES...");
		}
		catch (ArrayIndexOutOfBoundsException Aioobe)
		{
			System.out.println ("PASS DATA FROM COMMAND PROMPT...");
		}
		finally
		{
			System.out.println ("I AM FROM FINALLY...");
		}
	}
};
```

Throws block: This is the keyword which gives an indication to the calling function to keep the called function under try and catch blocks.

Syntax:
```
<Return type> method name (number of parameters if any) throws type of exception1,type of exception2,………type of exception;
```

Write a JAVA program which illustrates the concept of throws keyword?
Answer:
(CALLED FUNCTION)
```
package ep;
public class Ex2
{
	public void div (String s1, String s2) throws ArithmeticException, NumberFormatException
	{
		int n1=Integer.parseInt (s1);
		int n2=Integer.parseInt (s2);
		int n3=n1/n2;
		System.out.println ("DIVISOIN = "+n3);
	}
};
```

(CALLING FUNCTION)
```
import ep.Ex2;
class Ex3
{
	public static void main (String [] args)
	{
		try
		{
			String s1=args [0];
			String s2=args [1];
			Ex2 eo=new Ex2 ();
			eo.div (s1,s2);
		}
		catch (ArithmeticException Ae)
		{
			System.out.println ("DONT ENTER ZERO FOR DENOMINATOR");
		}
		catch (NumberFormatException Nfe)
		{
			System.out.println ("PASS INTEGER VALUES ONLY");
		}
		catch (ArrayIndexOutOfBoundsException Aioobe)
		{
			System.out.println ("PASS VALUES FROM COMMAND PROMPT");
		}
	}
};
```

Number of ways to find details of the exception:

In JAVA there are three ways to find the details of the exception. They are using an object of java.lang.Exception class, using public void printStackTrace method and using public string getMessage method.

i. Using an object of java.lang.Exception: An object of Exception class prints the name of the exception and nature of the message.

For example:
```
try
{
	int x=Integer.parseInt ("10x");
}
```

```
catch (Exception e)
{
      System.out.println (e); // java.lang.NumberFormatException : for input string 10x
}                               name of the exception             nature of the message
```

ii. Using printStackTrace method: This is the method which is defined in java.lang.Throwable class and it is inherited into java.lang.Error class and java.lang.Exception class. This method will display name of the exception, nature of the message and line number where the exception has taken place.

For example:

```
      try
      {
            ......;
            int x=10/0;
            ......;
      }
      catch (Exception e)
      {
      e.printStackTrace (); // java.lang.ArithmeticException : / by zero :  at line no: 4
      }                  name of the exception          nature of the message     line number
```

iii. Using getMessage method: This is also a method which is defined in java.lang.Throwable class and it is inherited into both Error and Exception classes. This method will display only nature of the message.

For example:

```
      try
      {
      ......;
      int x=10/0;
      ......;
      }
      catch (Exception e)
      {
      System.out.println (e.getMessage ()); // / by zero
      }                               nature of the message
```

USER or CUSTOM DEFINED EXCEPTIONS:

User defined exceptions are those which are developed by JAVA programmer as a part of application development for dealing with specific problems such as negative salaries, negative ages, etc.

Guide lines for developing user defined exceptions:

• Choose the appropriate package name.
• Choose the appropriate user defined class.
• Every user defined class which we have choose in step 2 must extends either java.lang.Exception or java.lang.RunTimeException class.
• Every user defined sub-class Exception must contain a parameterized Constructor by taking string as a parameter.
• Every user defined sub-class exception parameterized Constructor must called parameterized Constructor of either java.lang.Exception or java.lang.RunTimeException class by using super (string parameter always).

For example:

```
      package na; // step1
      public class Nage extends Exception // step2 & step3
      {
            public Nage (String s) // step4
            {
                  super (s);// step5
            }
      };
```

Write a JAVA program which illustrates the concept of user defined exceptions?

Answer:

```
package na;
public class Nage extends Exception
{
        public Nage (String s)
        {
                super (s);
        }
};

package ap;
import na.Nage;
public class Age
{
        public void decide (String s) throws NumberFormatException, Nage
        {
                int ag= Integer.parseInt (s);
                if (ag<=0)
                {
                        Nage na=new Nage ("U HAV ENTERED INVALID AGE..!");
                        throw (na);
                }
                else
                {
                        System.out.println ("OK, U HAV ENTERED VALID AGE..!");
                }
        }
};


import na.Nage;
import ap.Age;
class CDemo
{
        public static void main(String[] args)
        {
                try
                {
                        String s1=args [0];
                        ap.Age Ao=new ap.Age ();
                        Ao.decide (s1);
                }
                catch (Nage na)
                {
                        System.out.println (na);
                }
                catch (NumberFormatException nfe)
                {
                        System.out.println ("PASS ONLY INTEGER VALUES..!");
                }
                catch (ArithmeticException ae)
                {
                        System.out.println ("PASS INTEGER VALUES ONLY..!");
                }
                catch (ArrayIndexOutOfBoundsException aioobe)
                {
                        System.out.println ("PASS VALUES THROUGH COMMAND PROMPT..!");
                }
                catch (Exception e)
                {
                        System.out.println (e);
                }
        }
};
```

NOTE: Main method should not throw any exception since the main method is called by JVM and JVM cannot provide user friendly message.

<u>IO STREAMS:</u>

Generally, in JAVA programming we write two types of applications or programs. They are

1. Volatile or non-persistent program
2. Non-volatile or persistent program.

• A volatile program is one whose result is always stored in main memory of the computer i.e., RAM. Whatever the data which is stored in the main memory of the computer that data will be temporary i.e., the data which is available in main memory is volatile.

• A non-volatile program is one whose result is stored in secondary storage devices i.e., hard disk, magnetic tapes, etc. the data which is stored in secondary storage devices is permanent. To store the data we have two approaches. They are using files and using database.

If we store the data permanently in the form of a file, then the data of that file can be modified by any unauthorized user. Hence, industry always recommends not storing the data in the files. Since, files do not provide any security. Hence, industry always recommends storing the data in the form of database. Since, most of the popular databases provide security in the form of user names and passwords.

In order to store the data permanently in the form of files we must use or import a package called java.io.*

• Collection of records is known as file. A record is a collection of field values.

• A stream is a flow of data between primary memory and secondary memory either locally (within the system) or globally (across the network) or [A stream is a flow of bytes between primary memory and secondary memory either locally (within the system) or globally (across the network)] or [A stream is a flow of bits between primary memory and secondary memory either locally (within the system) or globally(across the network)].

Types of operations or modes on files:

On files we perform two types of operations they are

1. Read mode
2. Write mode

The following diagram will illustrate how to open the file either in read mode or in write mode or in both modes:

Types of streams in JAVA:

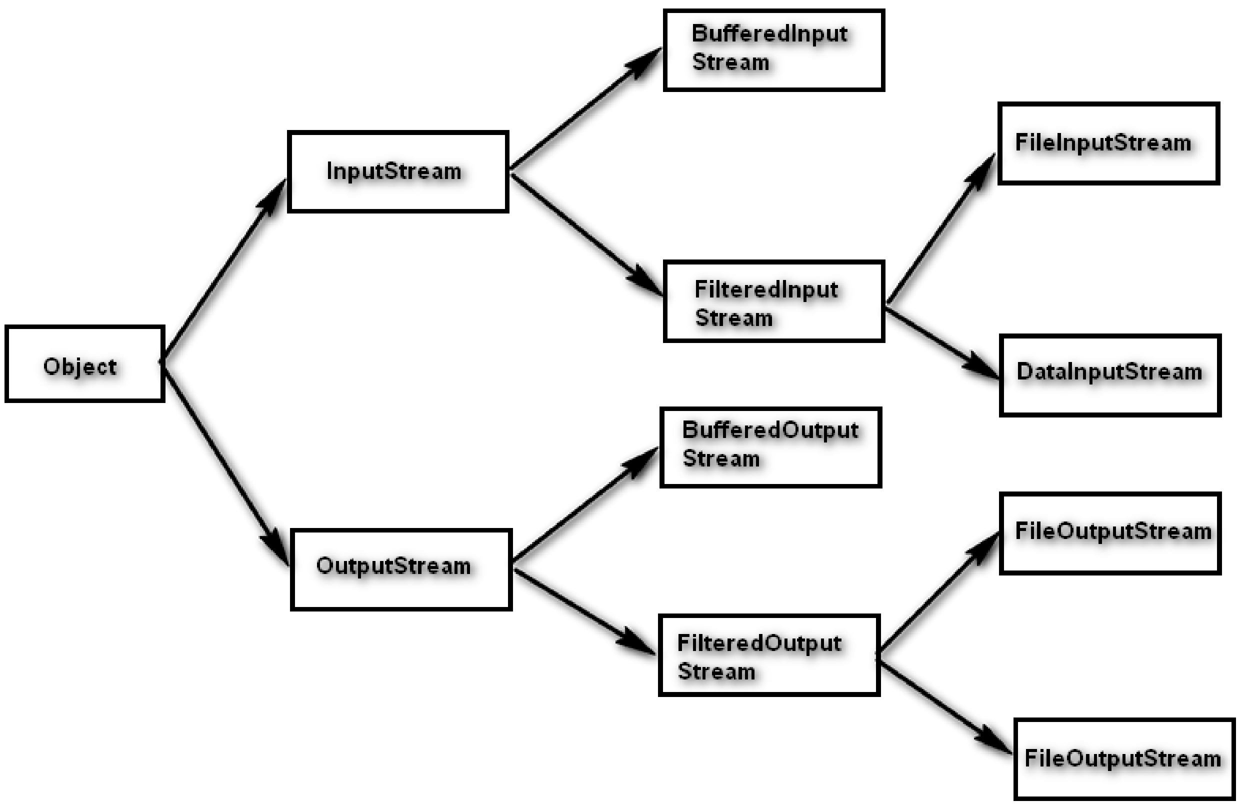
Based on transferring the data between primary memory to secondary memory and secondary memory to primary memory. In JAVA we have two types of streams they are

Byte streams

Char streams.

Byte streams are those in which the data will be transferred one byte at a time between primary memory to secondary memory and secondary memory to primary memory either locally or globally. java.io.* package contains some set of classes and interfaces which will transfer one byte at a time.

Hierarchy chart for byte streams:-



InputStream class:
This is an abstract class; hence we cannot create an object of this class directly. This class is basically used to open the file in read mode. In this class we have the following methods:

```
1. public int read ();
2. public int length (); // total size of the file
3. public int available (); // available number of bytes only
4. public void close ();
```

In JAVA end of file (EOF) is indicated by -1.

OutputStream class:
This is also an abstract class; hence we cannot create an object of this class directly. This class is used for opening the file in write mode. In this class we have the following methods:

```
1. public void write (int);
2. public int length ();
3. public void available ();
4. public void close ();
```

FileInputStream class:
This is the concrete (which we can create an object or it contains all defined methods) subclass of all InputStream class. This class is always used to open the file in read mode. Opening the file in read mode is nothing but creating an object of FileInputStream class.

Constructor:

```
FileInputStream (String fname) throws FileNotFoundException
```

For example:

```
FileInputStream fis=new FileInputStream (“abc.txt”);
```

If the file name abc.txt does not exist then an object of FileInputStream fis is null and hence we get FileNotFoundException. If the file name abc.txt is existing then the file abc.txt will be opened successfully in read mode and an object fis will point to beginning of the file.

FileOutputStream class:
This is the concrete sub-class of all OutputStream classes. This class is always used for opening the file in write mode is nothing but creating an object of FileOutputStream class.
Constructors:

```
1. FileOutputStream (String fname);
2. FileOutputStream (String fname, boolean flag);
```

If the flag is true the data will be appended to the existing file else if flag is false the data will be overlapped with the existing file data.
If the file is opened in write mode then the object of FileOutputStream will point to that file which is opened in write mode. If the file is unable to open in write mode an object of FileOutputStream contains null.

Write the following JAVA programs:

1) create a file that should contain 1 to 10 numbers.
2) read the data from the file which is created above.

Answer:
1)
```
import java.io.*;
class Fos
{
	public static void main (String [] args)
	{
		try
		{
			String fname=args [0];
			FileOutputStream fos=new FileOutputStream (fname, true);// mean append mode
			for (int i=1; i<=10; i++)
			{
			fos.write (i);
			}
			fos.close ();
		}
		catch (IOException ioe)
		{
			System.out.println (ioe);

		}
		catch (ArrayIndexOutOfBoundsException aiobe)
		{
			System.out.println ("PLEASE PASS THE FILE NAME..!");
		}
		System.out.println ("DATA WRITTEN..!");
	}
};
```

2)
```
import java.io.*;
class Fis
{
	public static void main(String[] args)
	{
		try
		{
			String fname=args [0];
			FileInputStream fis=new FileInputStream (fname);
```

```
                    int i;
                    while ((i=fis.read ())!=-1)
                    {
                    System.out.println (i);
                    }
                    fis.close ();
              }
              catch (FileNotFoundException fnfe)
              {
                    System.out.println ("FILE DOES NOT EXIST..!");
              }
              catch (IOException ioe)
              {
                    System.out.println (ioe);
              }
              catch (ArrayIndexOutOfBoundsException aiobe)
              {
                    System.out.println ("PLEASE PASS THE FILE NAME..!");
              }
       }
};
```

DataInputStream class:
This is used for two purposes. They are reading the data from input device like keyboard and reading the data from remote machine like server.
In order to perform the above operations we must create an object of DataInputStream class.
Constructors:

```
       DataInputStream (InputStream);
```

For example:

```
       DataInputStream dis=new DataInputStream (System.in);
```

An object of InputStream called 'in' is created as a static data member in System class.

Instance methods:

```
       1. public byte readByte ();
       2. public char readChar ();
       3. public short readShort ();
       4. public int readInt ();
       5. public long readLong ();

       6. public float readFloat ();
       7. public double readDouble ();
       8. public boolean readBoolean ();
       9. public String readLine ();
```

DataOutputStream class:
This is used for displaying the data onto the console and also used for writing the data to remote machine.

Constructor:

```
       DataOutputStream (OutputStream);
```

For example:

```
       DataOutputStream dos=new DataOutputStream (System.out);
```

Instance methods:
Old methods:

```
       1. public void writeByte (byte);
       2. public void writeChar (char);
       3. public void writeShort (short);
       4. public void writeInt (int);
```

```
    5. public void writeLong (long);
    6. public void writeFloat (float);
    7. public void writeDouble (double);
    8. public void writeBoolean (boolean);
    9. public void writeLine (String);
```

New methods:

```
    1. public void write (byte);
    2. public void write (char);
    3. public void write (short);
    4. public void write (int);
    5. public void write (long);
    6. public void write (float);
    7. public void write (double);
    8. public void write (boolean);
    9. public void write (String);
```

Write a JAVA program to read two numbers from keyboard and display their product (using older methods)?

Answer:

```
import java.io.*;
class DataRead
{
	public static void main (String [] args)
	{
		try
		{
			DataInputStream dis=new DataInputStream (System.in);
			System.out.println ("Enter first number : ");
			String s1=dis.readLine ();
			System.out.println ("Enter second number : ");
			String s2=dis.readLine ();
			int n1=Integer.parseInt (s1);
			int n2=Integer.parseInt (s2);
			int n3=n1*n2;
			System.out.println ("Product = "+n3);
		}
		catch (FileNotFoundException fnfe)
		{
			System.out.println ("FILE DOES NOT EXISTS");
		}
		catch (IOException ioe)
		{
			System.out.println (ioe);
		}
		catch (NumberFormatException nfe)
		{
			System.out.println ("PASS INTEGER VALUES ONLY");
		}
	}
};
```

Implement copy command of DOS?

Answer:

```
import java.io.*;
class Xcopy
{
	public static void main(String[] args)
	{
		FileInputStream fis=null;
		FileOutputStream fos=null;
		if (args.length!=2)
		{
```

```
                System.out.println ("INVALID ARGUMENTS..!");
            }
            else
            {
                try
                {
                    fis=new FileInputStream (args [0]);
                    fos=new FileOutputStream (args [1], true);
                    int i;
                    do
                    {
                        i=fis.read ();
                        char ch=(char)i;// type casting int value into char value
                        fos.write (ch);
                    }
                    while (i!=-1);
                }
                catch (FileNotFoundException fnfe)
                {
                    System.out.println (args [0]+"DOES NOT EXIST..!");
                }
                catch (IOException ioe)
                {
                    System.out.println (ioe);
                }
                catch (Exception e)
                {
                    e.printStackTrace ();
                }
                finally
                {
                    try
                    {
                        if (fis!=null)// file is closed when it is opened
                        {
                            fis.close ();
                        }
                        if (fos!=null)
                        {
                            fos.close ();
                        }
                    }
                    catch (IOException ioe)
                    {
                        ioe.printStackTrace ();
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace ();
                    }
                }// finally
            }// else
        }//main
};
```

Implement type command of DOS?

Answer:

```
import java.io.*;
class DosType
{
        public static void main (String [] args)
        {
            FileInputStream fis=null;
            if (args.length!=2)
            {
                System.out.println ("INVALID ARGS");
                }
                else
                {
                        try
                        {
                                fis=new FileInputStream (args [0]);
                                int i;
                                do
                                {
                                        i=fis.read ();
                                        char ch=(char) i;
                                        System.out.println (ch);
                                }
                                while (i!=-1);
                        }
                        catch (FileNotFoundException fnfe)
                        {
                                System.out.println (args [0]+" DOES NOT EXIT");
                        }
                        catch (Exception e)
                        {
                                e.printStackTrace ();
                        }
                        finally
                        {
                                try
                                {
                                        if (fis!=null)
                                        {
                                                fis.close ();
                                        }
                                }
                                catch (Exception e)
                                {
                                        e.printStackTrace ();
                                }
                        }// finally
                }// else
        }// main
};// DosType
```

Serialization: Serialization is the process of saving the state of the object permanently in the form of a file.

Steps for developing serialization sub-class:

1. Choose the appropriate package name.
2. Choose the appropriate class name whose object is participating in serialization.
3. Whichever class we have chosen in step 2 that must implements a predefined interface called java.io.Serializable (this interface does not contain any abstract methods and such type of interface is known as marked or tagged interface).
4. Choose set of properties or data members of the class.
5. Define set of setter methods (these are called mutators or modifiers).
6. Define set of getter methods (these are also known as inspectors).

Write a JAVA program for student serializable sub-class?

Answer:

```
package sp;
import java.io.*;
public class Student implements Serializable
{
        int stno;
        String sname;
```

```
	float marks;

	public void setSno (int stno)
	{
		this.stno=stno;
	}
	public void setSno (String sname)
	{
		this.sname=sname;
	}
	public void setSno (float marks)
	{
		this.marks=marks;
	}
	public int getStno ()
	{
		return (stno);
	}
	public String getSname ()
	{
		return (sname);
	}
	public float getMarks ()
	{
		return (marks);
	}
};
```

SERIALIZATION PROCESS:

1. Create an object of serializable sub-class.

   For example:

   ```
   Student so=new Student ();
   ```

2. Accept the data either from keyboard or from command prompt.
3. Call set of set methods to set the values for the serializable sub-class (Student) object.

   For example:

   ```
   so.setSno (sno);
   ```

4. Choose the file name and open it into write mode with the help of FileOutputStream class.
5. Since an object of FileOutputStream class cannot write the entire object at a line to the file. In order to write the entire object at a time to the file we must create an object of ObjectOutputStream class and it contains the following Constructor:

   ```
   ObjectOutputStream (FileOutputStream);
   ```

   For example:

   ```
   ObjectOutputStream oos=new ObjectOutputStream (fos);
   ```

The object oos is pointing to object fos, hence such type of streams are called chained or
sequenced stream.

6. ObjectOutputStream class contains the following instance method, which will write the entire object at a time to the file.

For example:

```
	ObjectOutpurStream.writeObject (so);
```

Write a JAVA program to save or serialize student data?

Answer:

```
import java.io.*;
class StudentData
{
```

```
	public static void main (String [] args)
	{
		try
		{
			sp.Student so=new sp.Student ();
			DataInputStream dis=new DataInputStream (System.in);
			System.out.println ("ENTER STUDENT NUMBER : ");
			int stno=Integer.parseInt (dis.readLine ());
			System.out.println ("ENTER STUDENT NAME : ");
			String sname=dis.readLine ();
			System.out.println ("ENTER STUDENT MARKS : ");
			float marks=Float.parseFloat (dis.readLine ());
			so.setStno (stno);
			so.setSname (sname);
			so.setMarks (marks);
			System.out.println ("ENTER THE FILE NAME TO WRITE THE DATA");
			String fname=dis.readLine ();
			FileOutputStream fos=new FileOutputStream (fname);
			ObjectOutputStream oos=new ObjectOutputStream (fos);
			oos.writeObject (so);
			System.out.println ("STUDENT DATA IS SERIALIZED");
			fos.close ();
			oos.close ();
		}
		catch (IOException ioe)
		{
			System.out.println ("PROBLEM IN CREATING THE FILE");
		}
		catch (Exception e)
		{
			System.out.println (e);
		}
	}
};
```

DESERIALIZATION PROCESS: De-serialization is a process of reducing the data from the file in the form of object.

Steps for developing deserialization process:

1. Create an object of that class which was serialized.

	For example:
```
Student so=new Student ();
```

2. Choose the file name and open it into read mode with the help of FileInputStream class.

	For example:
```
FileInputStream fis=new FileInputStream (“Student”);
```

3. Create an object of ObjectInputStream class. The Constructor of ObjectInputStream class is taking an object of FileInputStream class.

	For example:
```
ObjectInputStream ois=new ObjectInputStream (fis);
```

4. ObjectInputStream class contains the following method which will read the entire object or record.
```
public Object readObject ();
```
	For example:
```
Object obj=ois.readObject ();
```

5. Typecast an object of java.lang.Object class into appropriate Serializable sub-class object for calling get methods which are specially defined in Serializable sub-class.

	For example:

```
        So= (Student) obj;
```

6. Apply set of get methods for printing the data of Serializable sub-class object.

    For example:
    ```
    int stno=so.getStno ();
    String sname=so.getSname ();
    flaot marks=so.getMarks ();
    ```
7. Close the chained stream.

    For example:
    ```
    fis.close ();
    ois.close ();
    ```

Write a JAVA program to retrieve or de-serialize student data?

Answer:

```
import java.io.*;
import sp.Student;
class DeSerial
{
	public static void main (String [] args)
	{
		try
		{
			Student so=new Student ();
			DataInputStream dis=new DataInputStream (System.in);
			System.out.println ("ENTER FILE NAME TO READ");
			String fname=dis.readLine ();
			FileInputStream fis=new FileInputStream (fname);
			ObjectInputStream ois=new ObjectInputStream (fis);
			Object obj=dis.readObject ();
			so=(Object) obj;
			System.out.println ("STUDENT NUMBER "+so.getStno ());
			System.out.println ("STUDENT NAME "+so.getSname ());
			System.out.println ("STUDENT MARKS "+so.getMarks ());
			fis.close ();
			ois.close ();
		}
		catch (FileNotFoundException fnfe)
		{
			System.out.println ("FILE DOES NOT EXISTS");
		}
		catch (Exception e)
		{
			System.out.println (e);
		}
	}
};
```

Types of serialization:

In java we have four types of serialization; they are

1. Complete serialization,
2. Selective serialization,
3. Manual serialization and
4. Automatic serialization.

• Complete serialization is one in which all the data members of the class participates in serialization process.

• Selective serialization is one in which few data members of the class or selected members of the class are participating in serialization process.

In order to avoid the variable from the serialization process, make that variable declaration as transient i.e., transient variables never participate in serialization process.

• Manual serialization is one in which the user defined classes always implements java.io.Serializable interface.
• Automatic serialization is one in which object of sub-class of Serializable sub-class participates in serialization process.

```
class RegStudent extends Student
{
………………..
………………..
};
```

Buffered Streams:
Buffered streams are basically used to reduce physical number of read and write operation i.e., buffered streams always increases the performance of ordinary streams.
In byte streams we have two buffered streams, they are

1. BufferedInputStream and
2. BufferedOutputStream.

BufferedInputStream:
BufferedInputStream class is used for reducing number of physical read operation. When we create an object of BufferedInputStream, we get a temporary peace of memory space whose default size is 1024 bytes and it can be increased by multiples of 2.

Constructor:
```
public BufferedInputStream (FileInputStream);
```
For example:
```
FileInputStream fis=new FileInputStream (“abc.dat”);
BufferedInputStream bis=new BufferedInputStream (fis);
```

BufferedOutputStream:
BufferedOutputStream class is used for reducing number of physical write operation when we create an object of BufferedOutputStream, we get a temporary peace of memory space whose default size is 1024 bytes and it can be increased by multiple of 2.

Constructor:
```
public BufferedOutputStream (FileOutputStream);
```
For example:
```
FileOutputStream fos=new FileOutputStream (“abc.dat”);
BufferedOutputStream bos=new BufferedOutputStream (fos);
```

- Byte streams will transfer one byte of data and they will be implemented in system level applications such as flow of data in electronic circuits, development of ftp protocol etc.
- Character streams are those in which 2 bytes of data will be transferred and it can be implemented in higher level applications such as internet applications, development of protocols etc., like http etc.

<u>MULTI THREADING:</u>

Thread:
1. A flow of control is known as thread.
2. If a program contains multiple flow of controls for achieving concurrent execution then that program is known as multi threaded program.
3. A program is said to be a multi threaded program if and only if in which there exist ‘n’ number of sub-programs there exist a separate flow of control. All such flow of controls are executing concurrently such

flow of controls are known as threads and such type of applications or programs is called multi threaded programs.
The languages like C, C++ comes under single threaded modeling languages, since there exist single flow of controls where as the languages like JAVA, DOT NET are treated as multi threaded modeling languages, since there is a possibility of creating multiple flow of controls.
When we write any JAVA program there exist two threads they are fore ground thread and back ground thread.
Fore ground threads are those which are executing user defined sub-programs where as back ground threads are those which are monitoring the status of fore ground thread. There is a possibility of creating 'n' number of fore ground threads and always there exist single back ground thread.

Multi threading is the specialized form of multi tasking of operating system.

In information technology we can develop two types of applications. They are

1. Process based applications and
2. Thread based applications.

Process Based Applications	Thread Based Applications
1. It is the one in which there exist single flow of control.	1. It is the one in which there exist multiple flow of controls.
2. All C, C++ applications comes under it.	2. All JAVA, DOT NET applications comes under it.
3. Context switch is more (context switch is the concept of operating system and it says switching the control from one address page to another address page).	3. Context switch is very less.
4. For each and every sub-program there exist separate address pages.	4. Irrespective of 'n' number of subprograms there exist single address page.
5. These are treated as heavy weight components.	5. These are treated as light weight components.
6. In this we can achieve only sequential execution and they are not recommending for developing internet applications.	6. In thread based applications we can achieve both sequential and concurrent execution and they are always recommended for developing interact applications.

States of a thread:
When we write any multi threading applications, there exist 'n' numbers of threads. All the threads will under go different types of states. In JAVA for a thread we have five states. They are

new, ready, running, waiting and halted or dead state.

New state: It is one in which the thread about to enter into main memory.
Ready state: It is one in which the thread is entered into memory space allocated and it is waiting for CPU for executing.
Running state: A state is said to be a running state if and only if the thread is under the control of CPU.
Waiting state: It is one in which the thread is waiting because of the following factors:

- For the repeating CPU burst time of the thread (CPU burst time is an amount of the required by the thread by the CPU).
- Make the thread to sleep for sleep for some specified amount of time.
- Make the thread to suspend.
- Make the thread to wait for a period of time.

- Make the thread to wait without specifying waiting time.

Halted state: It is one in which the thread has completed its total execution.
As long as the thread is in new and halted states whose execution status is false where as when the thread is in ready, running and waiting states that execution states is true.

Creating a thread:
In order to create a thread in JAVA we have two ways. They are

1. by using java.lang.Thread class and
2. by using java.lang.Runnable interface.

In multi threading we get only one exception known as java.lang.InterruptedException.

Using java.lang.Thread:
Creating a flow of control in JAVA is nothing but creating an object of java.lang.Thread class.
An object of Thread class can be created in three ways. They are:

i) Directly Thread t=new Thread ();
ii) Using factory method Thread t1=Thread.currentThread ();
iii) Using sub-class of Thread class

```
class C1 extends Thread
{
……………………;
……………………;
};

C1 o1=new C1 ();
Thread t1=new C1 ();
Here, C1 is the sub-class of Thread class.
```

Thread API:

```
public static final int MAX_PRIORITY (10);
public static final int MIN_PRIORITY (1);
public static final int NORM_PRIORITY (5);
```

The above data members are used for setting the priority to threads are created. By default, whenever a thread is created whose default priority NORM_PRIORITY.

Constructors:
i) **Thread ():** With this Constructor we can create an object of the Thread class whose default thread name is Thread-0.

For example:

```
Thread t=new Thread ();
System.out.println (t.getName ());// Thread-0
```

ii) **Thread (String):** This Constructor is used for creating a thread and we can give the user specified thread name.

For example:

```
Thread t=new Thread (“JAVA”);
t.setName (“JAVA”);
t.setPriority (Thread.MAX_PRIORITY);
```

iii) **Thread (Runnable):** This Constructor is used for converting Runnable object into Thread object for entering into run method of Runnable interface by making use of start method of Thread class without giving thread name.

iv) **Thread (Runnable, String):** This Constructor is similar to above Constructor but we give thread name through this Constructor.

Instance methods:

```
        public final void setName (String);
        public final String getName ();
```

The above two methods are used for setting the name of the thread and getting the name from the thread respectively.

For example:

```
        Thread t1=new Thread ();
        T1.setName ("JAVA");
        String tp=t1.getName ();
        System.out.println (tp);// JAVA

        public final void setPriority (int);
        public final int getPriority ();
```

The above two methods are used for setting the priority to the thread and getting the priority of the thread respectively.

For example:

```
        Thread t1=new Thread ();
        Int pri=t1.getPriority ();
        System.out.println (pri);// 5  by default
        t1.setPriority (Thread.MAX_PRIORITY);
        pri=t1.getPriority ();
        System.out.println (pri); // 10
```

public void run ():
Any JAVA programmer want to define a logic for the thread that logic must be defined only run () method. When the thread is started, the JVM looks for the appropriate run () method for executing the logic of the thread. Thread class is a concrete class and it contains all defined methods and all these methods are being to final except run () method. run () method is by default contains a definition with null body. Since we are providing the logic for the thread in run () method. Hence it must be overridden by extending Thread class into our own class.

For example:

```
        class C1 extends Thread
        {
                public void run ()
                {
                        ………………………;
                        ………………………;
                }
        };
```

public final void start ():
This is the method which is used for making the Thread to start to execute the thread logic. The method start is internally calling the method run ().

For example:

```
        Thread t1=new Thread ();
        t1.start ();
        Thread t2=Thread.currentThread ();
        t2.start ();
```

public final void suspend ():
This method is used for suspending the thread from current execution of thread. When the thread is suspended, it sends to waiting state by keeping the temporary results in process control block (PCB) or job control block (JCB).

public final void resume ():
This method is used for bringing the suspended thread from waiting state to ready state. When the thread is resumed to start executing from where it left out previously by retrieving the previous result from PCB.

public final void stop ():
This method is used to stop the execution of the current thread and the thread goes to halted state from running state. When the thread is restarted it starts executing from the beginning only.

public final void wait (long msec):
This method is used for making the currently executing thread into waiting state for a period of time. Once this period of time is over, automatically the waiting thread will enter into ready state from waiting state.

Static methods:
i**) public static void sleep (long msec) throws InterruptedException** method is used (waiting state). If the sleep time is over automatically thread will come from waiting state to ready state.

For example:

```
Thread.sleep (1000);
```

ii) **public static Thread currentThread ()** is used for obtaining the threads which are running in the main memory of the computer.

For example:

```
Thread t=Thread.currentThread ();
System.out.println (t);// Thread [main (fat), 5, main (bat)]
```

Write a JAVA program to find the threads which are running internally and print priority values.
Answer:

```
class ThDemo
{
	public static void main (String [] args)
	{
		Thread t=Thread.currentThread ();
		System.out.println (t);
		t.setName ("ABC");
		System.out.println (t);
		System.out.println ("IS IT ALIVE..?"+t.isAlive ());// true
		Thread t1=new Thread ();// new state
		System.out.println ("IS IT ALIVE..?"+t.isAlive ());// false
		System.out.println ("DEFAULT NAME OF THREAD = "+t1.getName ());// Thread-0
		System.out.println ("MAXIMUM PRIORITY VALUE = "+Thread.MAX_PRIORITY);// 10
		System.out.println ("MINIMUM PRIORITY VALUE = "+Thread.MIN_PRIORITY);// 1
		System.out.println ("NORMAL PRIORITY VALUE = "+Thread.NORM_PRIORITY);// 5
	}
};
```

Output:

```
	Thread[main,5,main]
	Thread[ABC,5,main]
	IS IT ALIVE..?true
	IS IT ALIVE..?true
	DEFAULT NAME OF THREAD = Thread-0
	MAXIMUM PRIORITY VALUE = 10
	MINIMUM PRIORITY VALUE = 1
	NORMAL PRIORITY VALUE = 5
```

Instance methods:
public boolean isAlive () method is used for checking whether the thread is executing or not. It returns 'true' as long as the thread is in ready running and waiting states. It returns 'false' as long as the thread is in new and halted state.

Write a thread program which displays 1 to 10 numbers after each and every 1 second.
Answer:
```
class Th1 extends Thread
{
public void run ()
{
try
{
for (int i=1; i<=10; i++)
{
System.out.println ("VALUE OF I = "+i);
Thread.sleep (1000);
}
}
catch (InterruptedException ie)
{
System.out.println (ie);
}
}
};
class ThDemo1
{
public static void main (String [] args)
{
Th1 t1=new Th1 ();
System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());
t1.start ();
System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());
}
};
```
Output:
```
IS T1 ALIVE BEFORE START = false
IS T1 ALIVE AFTER START = true
VALUE OF I = 1
VALUE OF I = 2
VALUE OF I = 3
VALUE OF I = 4
VALUE OF I = 5
VALUE OF I = 6
VALUE OF I = 7
VALUE OF I = 8
VALUE OF I = 9
VALUE OF I = 10
```
Day - 58:

Re-write the above program using runnable interface.
Answer:
```
class Th1 implements Runnable
{
        public void run ()
        {
                try
```

```
				{
					for (int i=1; i<=10; i++)
					{
						System.out.println ("VALUE OF I = "+i);
						Thread.sleep (1000);
					}
				}
				catch (InterruptedException ie)
				{
					System.out.println (ie);
				}
		}
};
class ThDemo2
{
		public static void main (String [] args)
		{
				Runnable t=new Th1 ();
				Thread t1=new Thread (t, "ABC");
				System.out.println ("THREAD NAME = "+t1.getName ());
				System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());
				t1.start ();
				System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());
		}
};
```

Output:

```
THREAD NAME = ABC
IS T1 ALIVE BEFORE START = false
IS T1 ALIVE AFTER START = true
VALUE OF I = 1
VALUE OF I = 2
VALUE OF I = 3
VALUE OF I = 4
VALUE OF I = 5
VALUE OF I = 6
VALUE OF I = 7
VALUE OF I = 8
VALUE OF I = 9
VALUE OF I = 10
```

Write a JAVA program which produces 1 to 10 numbers in which even numbers are produced by one thread and odd numbers are produced by another thread.

Answer:

```
class Th1 extends Thread
{
		public void run ()
		{
				try
				{
						for (int i=1; i<=10; i+=2)
						{
								System.out.println ("VALUE OF ODD : "+i);
								Thread.sleep (1000);
						}
				}
				catch (InterruptedException ie)
				{
						System.out.println (ie);
```

```
            }
        }
};

class Th2 implements Runnable
{
        public void run ()
        {
                try
                {
                        for (int j=2; j<=10; j+=2)
                        {
                                System.out.println ("VALUE OF EVEN : "+j);
                                Thread.sleep (1000);
                        }
                }
                catch (InterruptedException ie)
                {
                        System.out.println (ie);
                }
        }
};

class ThDemo6
{
        public static void main (String [] args)
        {
                Th1 t1=new Th1 ();// object of Thread class
                Th2 t2=new Th2 ();// object of Runnable class
                Thread t=new Thread (t2);// Runnable is converted into Thread object
                System.out.println ("BEFORE START T1 IS : "+t1.isAlive ());
                System.out.println ("BEFORE START T2 IS : "+t.isAlive ());
                t1.start ();
                t.start ();
                System.out.println ("AFTER START T1 IS : "+t1.isAlive ());
                System.out.println ("AFTER START T2 IS : "+t.isAlive ());
                try
                {
                        t1.join ();// to make thread to join together for getting performance
                        t.join ();
                }
                catch (InterruptedException ie)
                {
                        System.out.println (ie);
                }
                System.out.println ("AFTER JOINING T1 IS : "+t1.isAlive ());
                System.out.println ("AFTER JOINING T2 IS : "+t.isAlive ());
        }
};
```

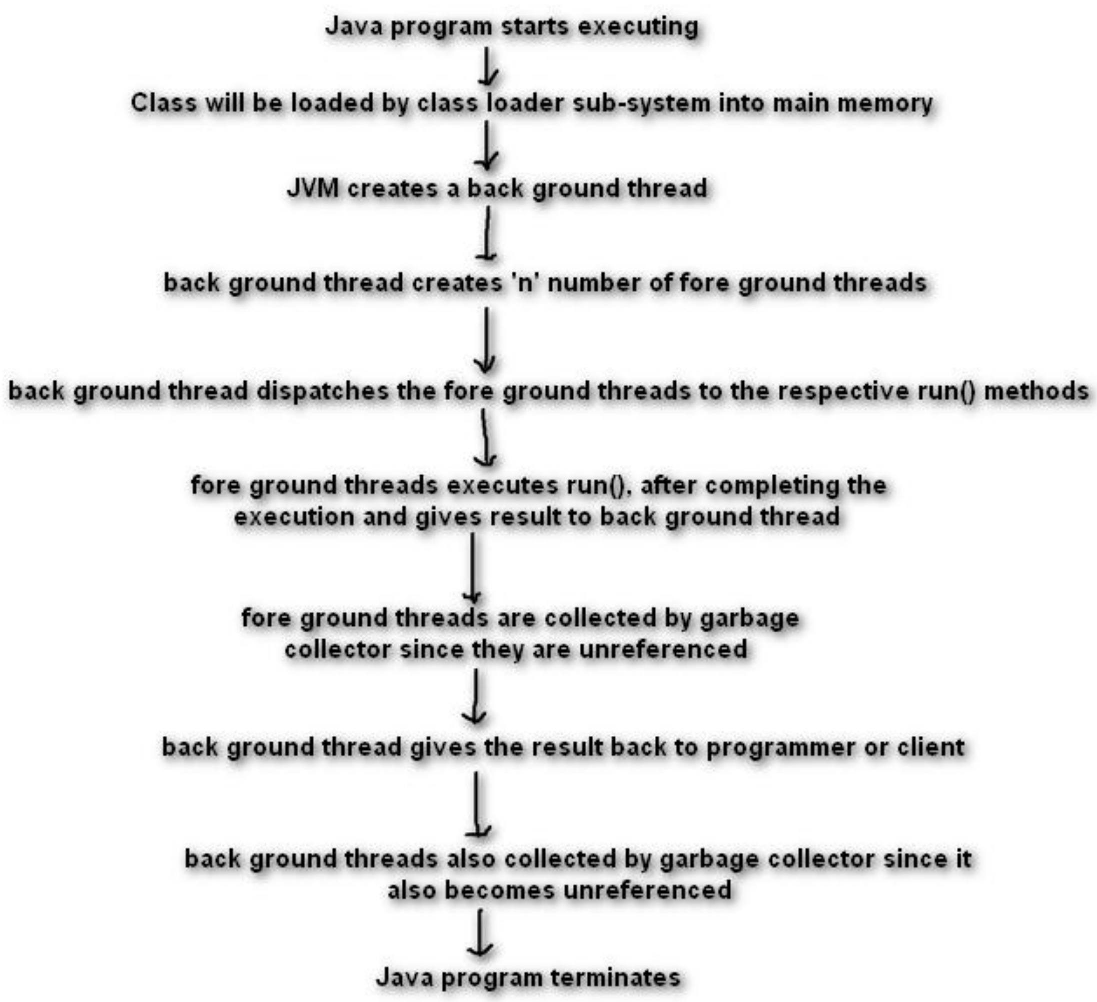
Output:

```
BEFORE START T1 IS : false
BEFORE START T2 IS : false
AFTER START T1 IS : true
AFTER START T2 IS : true
VALUE OF ODD : 1
VALUE OF EVEN : 2
VALUE OF ODD : 3
VALUE OF EVEN : 4
VALUE OF ODD : 5
VALUE OF EVEN : 6
VALUE OF ODD : 7
VALUE OF EVEN : 8
VALUE OF ODD : 9
VALUE OF EVEN : 10
```

```
AFTER JOINING T1 IS : false
AFTER JOINING T2 IS : false
```

public void join () Throws InterruptedException
This method is used for making the fore ground threads to join together so that JVM can call the garbage collector only one time for collecting all of them instead of collecting individually.

How a thread executes internally?
Answer:



Java program starts executing
Class will be loaded by class loader sub-system into main memory
JVM creates a back ground thread
back ground thread creates 'n' number of fore ground threads
back ground thread dispatches the fore ground threads to the respective run() methods
fore ground threads executes run(), after completing the execution and gives result to back ground thread
fore ground threads are collected by garbage collector since they are unreferenced
back ground thread gives the result back to programmer or client
back ground threads also collected by garbage collector since it also becomes unreferenced
Java program terminates

SYNCHRONIZATION:
It is the process of allowing only one thread at a time among ‘n’ number of threads into that area which is sharable to perform write operation.

If anything (either data members or methods or objects or classes) is sharable then we must apply the concept of synchronization.

Let us assume that there is a sharable variable called balance whose initial value is zero. There are two threads t1 and t2 respectively. t1 and t2 want to update the balance variable with their respective values i.e., 10 and 20 at the same time. After completion of these two threads the final value in the balance is either 10 or 20 but not 30 which is the inconsistent result. To achieve the consistent result we must apply the concept of synchronization.

When we apply synchronization concept on the above scenario, when two threads are started at same time, the first thread which is started is given a chance to update balance variable with its respective value. When second thread is trying to access the balance variable value JVM makes the second thread to wait until first thread completes its execution by locking balance variable value. After completion of first thread the value in the balance is 10 and second thread will be allowed to update balance variable value. After completion of second thread the value of the balance is 30, which is the consistent result. Hence, in synchronization locking and unlocking is taking place until all the threads are completed their execution.

Synchronization Techniques:
In JAVA we have two types of synchronization techniques. They are

1. Synchronized methods
2. Synchronized blocks.

1. Synchronized methods:
If any method is sharable for 'n' number of threads then make the method as synchronized by using a keyword synchronized.
In JAVA we have two types of synchronized methods. They are

i) Synchronized Instance methods and
ii) Synchronized static methods.

Synchronized Instance methods: If the ordinary instance method is made it as synchronized then the object of the corresponding class will be locked.
Syntax:
```
synchronized <return type> method name (method parameters if any)
{
       Block of statements;
}
```
For example:
```
class Account
{
       int bal=0;
       synchronized void deposit (int amt)
       {
              bal=bal+amt;
              System.out.println ("CURRENT BALANCE = "+bal);
       }
};
```

Synchronized static method: If an ordinary static method is made it as synchronized then the corresponding class will be locked.
Syntax:
```
synchronized static <return type> method name (method parameters if any)
{
       Block of statements;
}
```
For example:
```
class Account
{
       static int bal=0;
       synchronized static void deposit (static int amt)
       {
              bal=bal+amt;
              System.out.println ("CURRENT BALANCE = "+bal);
       }
};
```

2. Synchronized block:

This is an alternative technique for obtaining the concept of synchronization instead of synchronized methods.

When we inherit non-synchronized methods from either base class or interface into the derived class, we cannot make the inherited method as synchronized. Hence, we must use synchronized blocks.

Syntax:
```
synchronized (object of current class)
{
	Block of statement(s);
}
```
For example:
```
class BankOp
{
	void deposit (int amt);// p a instance
};

class Account implements BankOp
{
	int bal=0;
	public void deposit (int amt)
	{
		synchronized (this);
		{
			bal=bal+amt;
			System.out.println (“current value=”+bal);
		}
	}
};
```

Write a synchronization program in which there exist an account, there exist ‘n’ number of customers and all the customers want to deposit 10 rupees to the existing balance of account.

Answer:
```
class Account
{
	private int bal=0;
	synchronized void deposit (int amt)
	{
		bal=bal+amt;
		System.out.println ("CURRENT BALANCE="+bal);
	}
	int getBal ()
	{
		return (bal);
	}
};

class cust extends Thread
{
	Account ac;// has-a relationship
	cust (Account ac)
	{
		this.ac=ac;
	}
	public void run ()
	{
		ac.deposit (10);
	}

};

class SyncDemo
```

```
{
	public static final int noc=5;
	public static void main (String [] args)
	{
		Account ac=new Account ();
		cust cu []=new cust [noc];
		for (int i=0; i<noc; i++)
		{
			cu [i]=new cust (ac);// giving account object 'ac' to each & every customer
object
		}
		for (int i=0; i<noc; i++)
		{
			cu [i].start ();
		}
		for (int i=0; i<noc; i++)
		{
			System.out.println ("IS ALIVE..? "+cu [i].isAlive ());
		}
		try
		{
			for (int i=0; i<noc; i++)
			{
				cu [i].join ();
			}
		}
		catch (InterruptedException ie)
		{
			System.out.println (ie);
		}
		for (int i=0; i<noc; i++)
		{
			System.out.println ("IS ALIVE..? "+cu [i].isAlive ());
		}
		System.out.println ("TOTAL BALANCE="+ac.getBal ());
	}
};
```

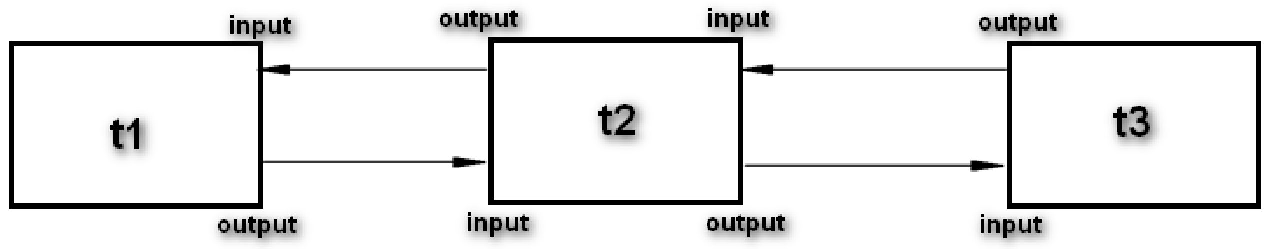
```
Output:
IS ALIVE..? true
IS ALIVE..? true
IS ALIVE..? true
IS ALIVE..? true
CURRENT BALANCE=10
CURRENT BALANCE=20
CURRENT BALANCE=30
CURRENT BALANCE=40
CURRENT BALANCE=50
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
IS ALIVE..? false
TOTAL BALANCE=50
```

NOTE: If we execute the above program on single user operating systems it is mandatory for the JAVA programmer to write synchronized keyword since user operating system cannot take care about synchronization by default. Whereas if we run the above program in multi user or threaded operating systems we need not to use a keyword synchronized (optional).

It is always recommended to write synchronized keyword irrespective which ever operating system we use.

Inter thread communication:

If two or more threads are exchanging the data then that communication is known as inter thread communication. In inter thread communication; an output of one thread is given as an input to another thread. In order to develop inter thread communication applications we must make use of a class called java.lang.Object.



Methods in java.lang.Object:

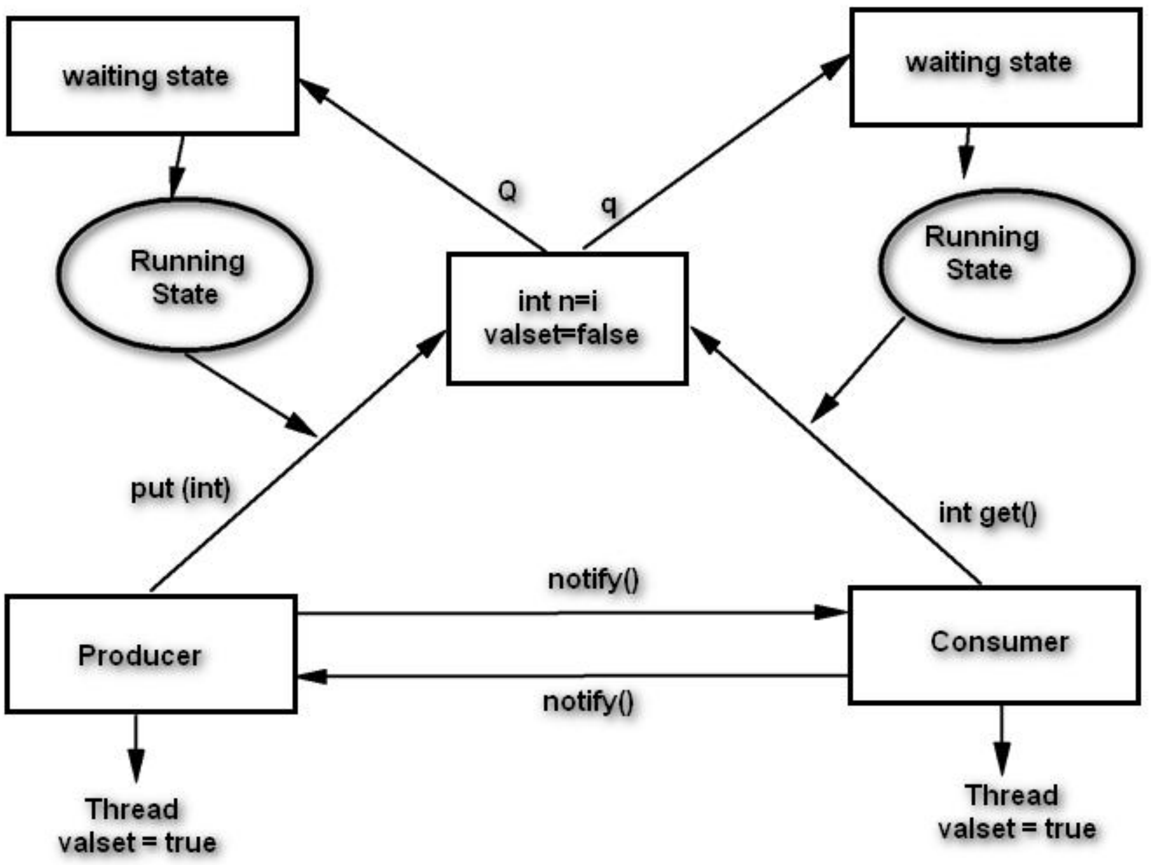
1. public void wait (long msec)
2. public void wait (long msec, int nanosec)
3. public void wait ()
4. public void notify ()
5. public void notifyAll ()

Methods 1 and 2 are used for making the thread to wait for some period of time. Once the waiting time is over automatically the thread will come from waiting state to ready state.

Method 3 is used for making the thread to wait without specifying waiting time.

Method 4 is used for bringing a single waiting thread into ready state.

Method 5 is used for bringing all the threads from waiting state to ready state.

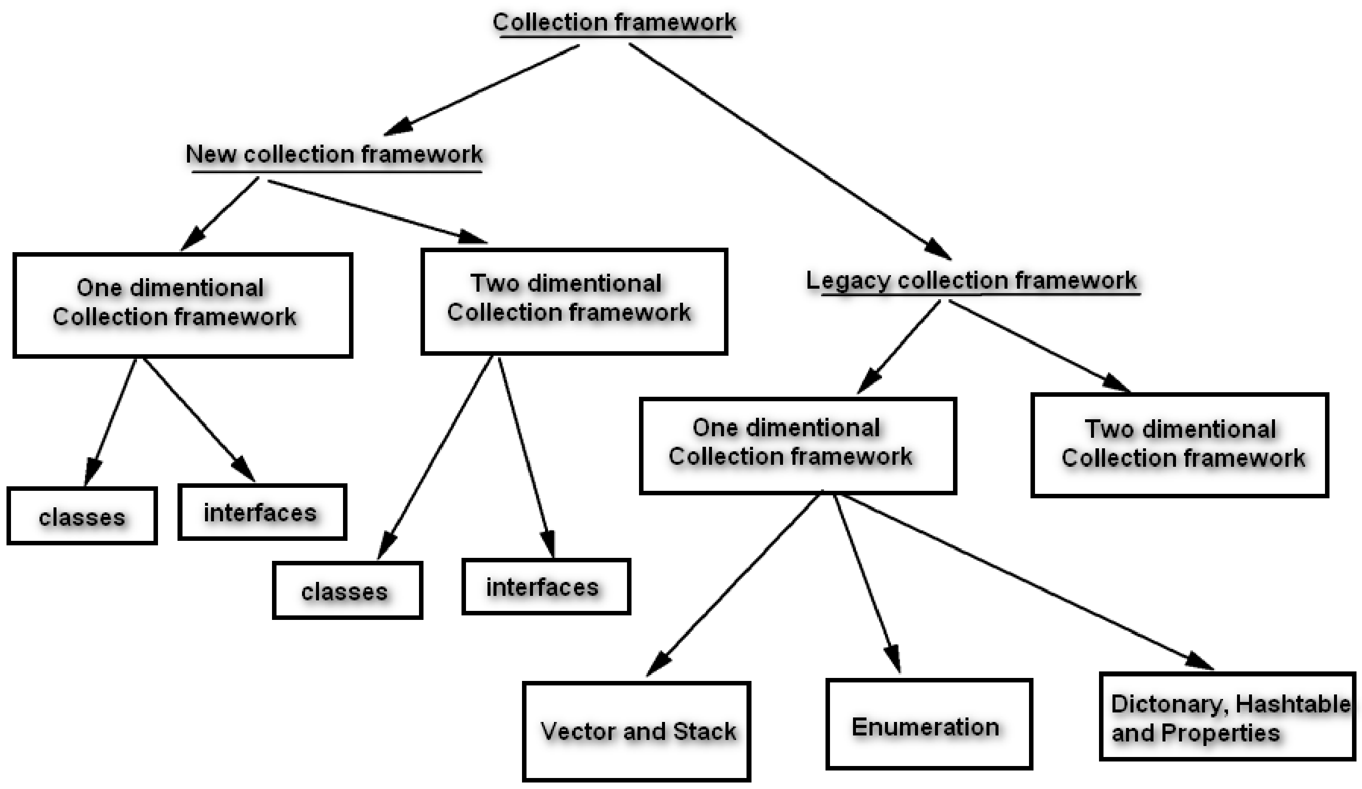


Develop producer consumer program by using inter thread communication?
Answer:
```
class Q
{
        int n;
        boolean valset;
        synchronized void put (int i)
        {
                try
                {
                        if (valset)
                        {
                                wait ();
                        }
                }
                catch (InterruptedException ie)
                {
                        System.out.println (ie);
                }
                n=i;
                System.out.println ("PUT="+i);
                valset=true;
                notify ();
        }// put

        synchronized int get ()
        {
                try
                {
                        if (!valset)
                        {
                                wait ();
                        }
```

```
            }
            catch (InterruptedException ie)
            {
                  System.out.println (ie);
            }
            System.out.println ("GET="+n);
            valset=false;
            notify ();
            return (n);
      }// get
};// Q

class Producer implements Runnable
{
      Q q;
      Thread t;
      Producer (Q q)
      {
            this.q=q;
            t=new Thread (this, "Producer");
            t.start ();
      }
      public void run ()
      {
            int i=0;
            System.out.println ("NAME OF THE THREAD = "+t.getName ());
            while (true)
            {
                  q.put (++i);
            }
      }
};// Producer

class Consumer implements Runnable
{
      Q q;
      Thread t;
      Consumer (Q q)
      {
            this.q=q;
            t=new Thread (this, "Consumer");
            t.start ();
      }
      public void run ()
      {
            System.out.println ("NAME OF THE THREAD = "+t.getName ());
            while (true)
            {
                  int i=q.get ();
            }
      }
};// Consumer

class PCDemo
{
      public static void main (String [] args)
      {
            Q q=new Q ();
            Producer p=new Producer (q);
            Consumer c=new Consumer (q);
      }
};// PCDemo
```

COLLECTION FRAMEWORK:

Collection framework is the standardized mechanism of grouping of similar or dissimilar type of objects into a single object. This single object is known as collection framework object.


Collection framework
New collection framework
One dimentional Collection framework
Two dimentional Collection framework
classes
interfaces
classes
interfaces
Legacy collection framework
One dimentional Collection framework
Two dimentional Collection framework
Vector and Stack
Enumeration
Dictonary, Hashtable and Properties

Goals of collection frameworks:
1. Collection framework improves the performance of JAVA, J2EE projects (when we want to transfer the bulk amount of data from client to server and server to client, using collection framework we can transfer that entire data at a time).
2. Collection framework allows us to prove similar or dissimilar type of objects.
3. Collection framework is dynamic in nature i.e., they are extends (arrays contains the size which is fixed in nature and they allows similar type of data).
4. Collection framework contains adaptability feature (adaptability is the process of adding one collection object at the end of another collection object).
5. Collection framework is algorithmic oriented (collection framework contains various sorting and searching techniques of data structures as a predefined concepts).
6. In order to deal with collection framework we must import a package called java.util.*

Collection framework is divided into two categories. They are
1. New collection framework and
2. Legacy (old) collection framework.

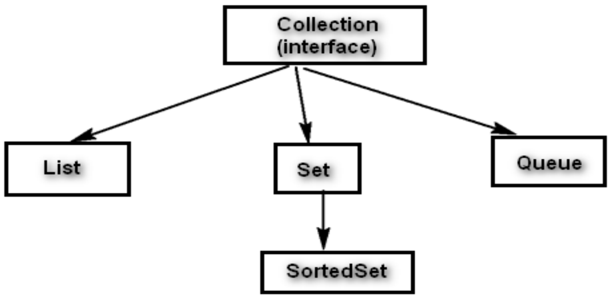
NEW COLLECTION FRAMEWORK:
New collection framework is again broadly divided into two categories. They are
1. One dimensional collection framework and
2. Two dimensional collection framework.

One dimensional collection framework

A one dimensional collection framework is one in which the data is organized either in the form of row or in the form of column by containing similar or dissimilar categories of objects into a single object. This single object is known as one dimensional collection framework object. One dimensional collection framework contains one dimensional collection framework interfaces and one dimensional collection framework classes.

a) One dimensional collection framework interfaces
As a part of one dimensional collection framework interfaces in JAVA, we have five interfaces.
They are

i. Collection
ii. List
iii. Set
iv. SortedSet
v. Queue



<u>java.util.Collection:</u>
Collection is an interface whose object allows us to organize similar or different type of objects into single object. The Collection interface is having the following features:
i)It is available at the top of the hierarchy of all the interfaces which are available in the collection framework.
ii) An object of Collection allows us to add duplicate elements.
iii) Collection object always displays the data in forward direction only.
iv) Collection object will print the data on the console in random order.
v) Collection object always allows us to insert an element at end only i.e., we cannot insert an element at the specific position.

<u>java.util.List:</u>
i) List is the sub-interface of java.util.Collection interface.
ii) List object also allows us to add duplicates.
iii) List object automatically displays the data in sorted order.
iv) List object allows us to add an element either at the ending position or at specific position.

v) List object allows us to retrieve the data in forward direction, backward direction and random retrieval.

java.util.Set:

i) Set is the sub-interface of java.util.Collection interface.
ii) An object of Set does not allows duplicates i.e., all the elements in the set must be distinct(unique).
iii) Set object always displays the data in random order.
iv) Set object allows us to add the elements only at ending position.
v) Set object allows us to retrieve the data only in forward direction.

java.util.SortedSet:

i) SortedSet is the sub-interface of java.util.Set interface.
ii) SortedSet object does not allows duplicates.
iii) SortedSet object will displays the data automatically in sorted order.
iv) SortedSet object allows us to add the elements only at ending position.
v) SortedSet object allows us to retrieve only in forward direction.

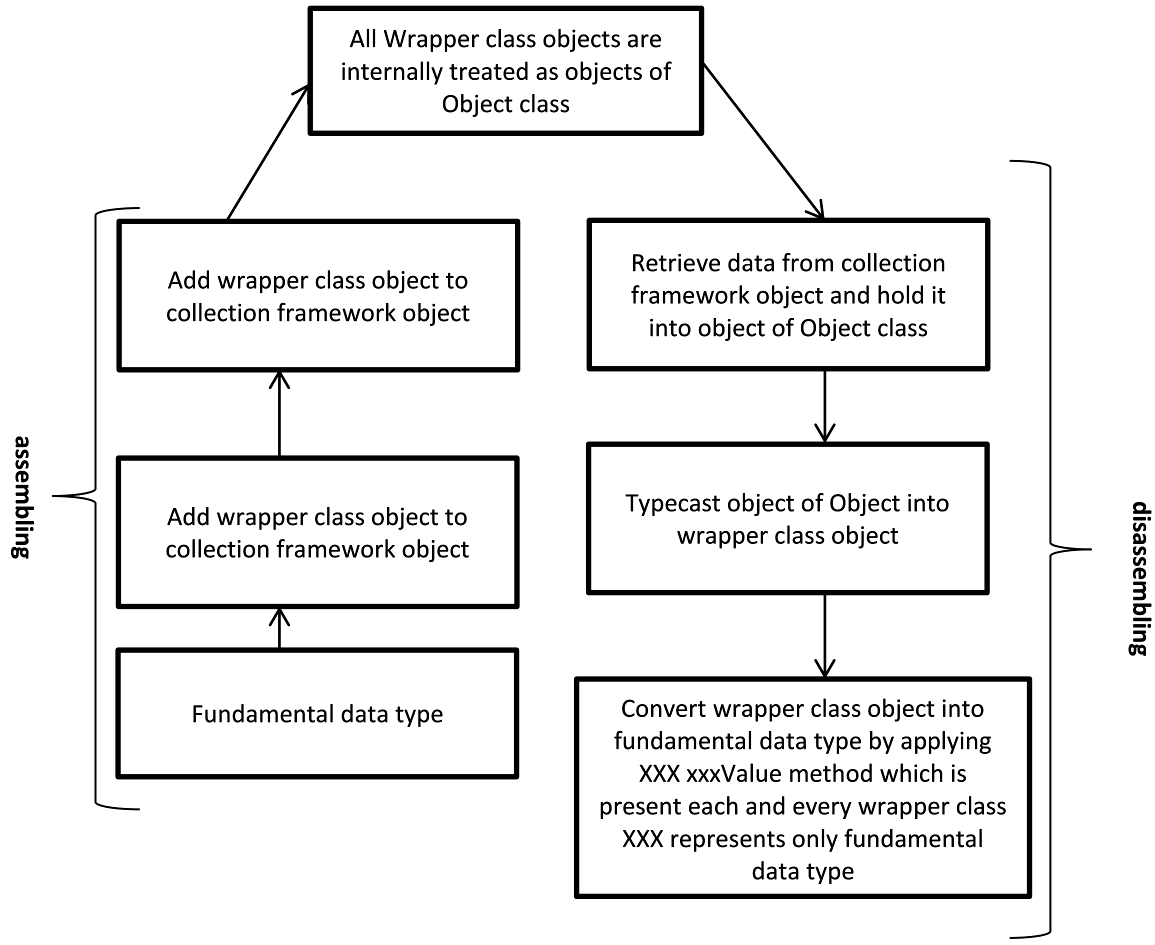
Methods in Collection interface:

1. **public int size ():** This method is used for determining the number of elements in Collection interface object.
2. **public boolean add (java.lang.Object):** This method is used for adding an object to Collection object. When we use this method to add the object to Collection objects and List object, this method always returns true. Since, Collection object and List object allows duplicates. When we apply the same method with Set and SortedSet methods and when we are trying to add duplicates, this method returns false.

NOTE:
Whatever the data we want to add through collection framework object that fundamental data must be converted into the corresponding wrapper class object and all the objects of wrapper classes are treated as objects of java.lang.Object class.

3. **public boolean addAll (Collection):**
This method is used for adding the entire collection object at the end of another Collection object. As long as we apply this method with List and Collection interfaces, it returns true. Since, they allow duplicates. When we apply this method with Set and SortedSet, this method may return false. Since, duplicates are not allowed.

NOTE:
Collection framework is nothing but assembling different or similar type of objects and dissembling similar or different type of objects and it is shown in the following diagrammatic representation.

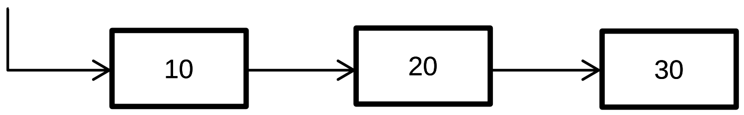

All Wrapper class objects are internally treated as objects of Object class
Add wrapper class object to collection framework object
Add wrapper class object to collection framework object
Fundamental data type
assembling
Retrieve data from collection framework object and hold it into object of Object class
Typecast object of Object into wrapper class object
Convert wrapper class object into fundamental data type by applying XXX xxxValue method which is present each and every wrapper class XXX represents only fundamental data type
disassembling

4. **public boolean isEmpty ():**
Returns true when there are no object found in Collection interface object otherwise return false.

5. **public Object [] toArray ():**
This method is used for extracting the data from Collection object in the form of array of objects of java.lang.Object class.

For example:
```
int s=0;
Object obj []=s.toArray ();
for (int i=0; i<obj.length; i++)
{
     Interger io= (Integer) obj [i];
     int x=io.intValue ();
     s=s+x;
}
System.out.println ("Sum = "+s);
```

6. public Iterator iterator ():
This method is used for extracting the data from Collection framework object.



For example:
    Iterator itr=co.iterator ();

```
Int s=0;
While (itr.hasNext ())
{
        Object obj=itr.next ();
        Integer io= (Integer) obj;
        int x=io.intValue ();
        s=s+;
}
```

Iterator interface: Iterator is an interface which always uses the extract the data from any Collection object.

Methods in Iterator interface:

```
1. public boolean hasNext ()
2. public Object next ()
3. public Object remove ()
```

Method 1 is used for checking whether we have next element or not. If next element available in Collection object it returns true otherwise it returns false.
Method 2 is used for obtaining the next element in the Collection object.
Method 3 is used for removing the element from Collection object.
Methods 2 and 3 can be used as long as method 1 returns true.

List: List is an interface which extends Collection.

Methods in List interface:
1. **public Object get (int):** This method is used for obtaining that element from the specified position. If the get method is not returning any value because of invalid position the value of object of object is NULL.
2. **public Object remove (int):** This method is used for removing the objects from List object based on position.
3. **public Object remove (Object):** This method is used for removing the objects from List object based on content.
4. **Public void add (int pos, Object):** This method is used for adding an object at the specified position.
5. **public void addAll (int pos, Collection):** This method is used for adding one Collection object to another Collection object at the specified position.
6. **public List headList (Object obj):** This method is used for obtaining those objects Xi's which are less than or equal to target object (Xi≤obj).
7. **public List tailList (Object obj):** This method is used for obtaining those objects Xi's which are greater than target object (Xi>obj or Xi≥(obj-1)).
8. **public List subList (Object obj1, Object obj2):** This method is used for obtaining those values which are a part of List i.e., range of values [or] In subList method is select those values Xi's which are less than or equal to object 1 and greater than object 2 (obj1≤Xi<obj2).
9. **public ListIterator listIterator ():** This method is used for extracting the data from List object either in forward or in backward or in both the directions. ListIterator is an interface which extends Iterator interface.
This interface contains the following methods:

```
public boolean hasPrevious (); → 1
public Object previous (); → 2
public void set (Object); → 3
```

All the methods of iterator are also coming.

Method-1 is used for checking weather we have previous element or not, this method returns true as long as we have previous elements otherwise false.
Method-2 is used for obtaining previous element.
Method-3 is used for modifying the existing Collection object returns true.
Methods 2 and 3 can be used as long as method-1 returns true.

<u>One dimentional collection framework classes:</u>
One dimensional collection framework classes are defining all the methods of collection framework interfaces. The following table gives the collection framework classes and its hierarchy.

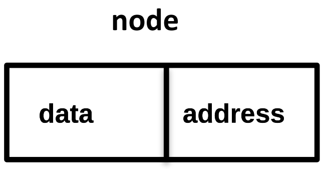
Interfaces	Classes
Collection	AbstractAbstractCollection implements Collection
List	AbstractList extends AbstractCollection implements List
Set	AbstractSet extends AbstractCollection implements Set
SortedSet	AbstractSequentialList extends AbstractList implements SortedSet
	LinkedList extends AbstractSequential
	ArrayList extends AbstractSequential
	HashSet extends AbstractSet
	TreeSet extends AbstractSet

Collection framework classes contains the definition for those methods which are coming from Collection interfaces i.e., all Collection interface methods defined in collection classes AbstractCollection implements Collection, AbstractList extends AbstractCollection implements List, AbstractSet extends AbstractCollection implements Set, AbstractSequentialList extends AbstractList implements SortedSet, LinkedList extends AbstractSequential, ArrayList extends AbstractSequential, HashSet extends AbstractSet and TreeSet extends AbstractSet.

The classes AbstractCollection implements Collection, AbstractList extends AbstractCollection implements List, AbstractSet extends AbstractCollection implements Set and AbstractSequentialList extends AbstractList implements SortedSet are abstract classes we never make use of them as a part of real time applications but we make use of the classes LinkedList extends AbstractSequential, ArrayList extends AbstractSequential, HashSet extends AbstractSet and TreeSet extends AbstractSet.

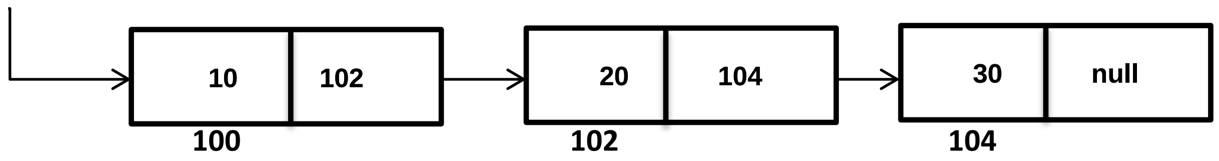
java.util.LinkedList:
LinkedList is the concrete sub-class of collection classes. LinkedList object allows us to group similar or dissimilar type of objects. Creating a LinkedList is nothing but creating an object of java.util.LinkedList class.

The data is organized in LinkedList in the form of nodes. The node contains two parts, they are data part, address part.



Data part always gives the actual data which we want to represent. Address part represents address of the next node.
For the last node in the LinkedList the address part must be NULL which indicates end of the LinkedList.

ll (Linked List)



LinkedList API:

Constructors:

```
      LinkedList ();
      LinkedList (int size);
```

Instance methods:

```
      Object getFirst (); →1
      Object getLast (); → 2
      public void addFirst (Object obj); → 3
      public void addLast (Object obj); → 4
      public void removeFirst (); → 5
      public void removeLast (); → 6
```

Methods 1 and 2 are used for obtaining first and last objects of LinkedList respectively.
Methods 3 and 4 are used for adding similar or different objects to the LinkedList object respectively.
Methods 5 and 6 are used for removing first and last objects from LinkedList respectively.

Set: Set does not contain any special method except Collection interface methods.

SortedSet:
SortedSet extends Set. The following are the methods which are specially available in SortedSet interface.

```
      public Object first ();
      public Object last ();
      public SortedSet headSet (Object obj); xi <= obj
      public SortedSet tailSet (Object obj1, Object obj2);
```

Write a java program which implements the concept of LinkedList?

Answer:

```
import java.util.*;
class Linkedlist
{
      public static void main (String [] args)
      {
            LinkedList ll=new LinkedList ();
            System.out.println ("CONTENTS OF l1 = "+ll);
            System.out.println ("SIZE = "+ll.size ());
            ll.add (new Integer (10));
            ll.add (new Integer (20));
            ll.add (new Integer (30));
            ll.add (new Integer (40));
            System.out.println ("CONTENTS OF ll = "+ll);
            System.out.println ("SIZE = "+ll.size ());
            // retrieving data of ll using toArray ()
            Object obj []=ll.toArray ();
            int s=0;
            for (int i=0; i<obj.length; i++)
            {
                  Integer io= (Integer) obj [i];
                  int x=io.intValue ();
                  s=s+x;
            }
            System.out.println ("SUM USING toArray () = "+s);
            ll.addFirst (new Integer (5));
            ll.addFirst (new Integer (6));
            System.out.println ("CONTENTS OF ll = "+ll);
            System.out.println ("SIZE = "+ll.size ());
            // retrieving data of ll using iterator ()
            Iterator itr=ll.iterator ();
            int s1=0;
            while (itr.hasNext ())
            {
                  Object obj1=itr.next ();
                  Integer io1= (Integer) obj1;
                  int x1=io1.intValue ();
                  s1=s1+x1;
            }
            System.out.println ("SUM USING iterator () = "+s1);
            // retrieving data of ll using ListIterator ()
            ListIterator litr=ll.listIterator ();
            while (litr.hasNext ())
            {
                  Object obj2=litr.next ();
                  System.out.print (obj2+",");
            }
            System.out.println ("\n");
            while (litr.hasPrevious ())
            {
                  Object obj3=litr.next ();
                  System.out.print (obj3+",");
            }
            System.out.println ("\n");
            Object obj4=ll.get (2);// random retrieval
            System.out.println (obj4);
      }
};
```

<u>**Disadvantages of LinkedList:**</u>
1. Additional memory space is created for address part of the node in heap memory.
2. Retrieval time is more.
3. Since, we are wasting most of the memory space for addresses, performance will be reduced.

java.util.ArrayList:
ArrayList is also concrete sub-class of collection framework classes whose object allows us to organize the data either in similar type or in different type.

Creating ArrayList is nothing but creating an object of ArrayList class.

ArrayList API:

ArrayList ();
ArrayList (int size);

Advantages of ArrayList over LinkedList:
1. No additional memory space is required for data of ArrayList.
2. Retrieval time is quite faster.
3. Performance is high. Since, there is no memory space is required for maintaining address of data of ArrayList.

HashSet and TreeSet:

HashSet	TreeSet
1. It extends AbstractSet.	1. It extends AbstractSet and implements SortedSet.
2. It follows hashing mechanism to organize its data.	2. It follows binary trees (AVL trees) to organize the data.
3. We cannot determine in which order it displays its data.	3. It always displays the data in sorted order.
4. Retrieval time is more.	4. Retrieval time is less.
5. The operations like insertion, deletion and modifications takes more amount of time.	5. The operations like insertion, deletion and modifications take very less time.
6. Creating HashSet is nothing but creating an object of HashSet () class.	6. Creating TreeSet is nothing but creating an object of TreeSet () class.

Write a JAVA program which illustrates the concept of TreeSet?
Answer:
```
import java.util.*;
class tshs
{
	public static void main (String [] args)
	{
		TreeSet ts=new TreeSet ();
		System.out.println ("CONTENTS OF ts = "+ts);
		System.out.println ("SIZE OF ts = "+ts.size ());
		ts.add (new Integer (17));
		ts.add (new Integer (188));
		ts.add (new Integer (-17));
		ts.add (new Integer (20));
		ts.add (new Integer (200));
		ts.add (new Integer (177));
		System.out.println ("CONTENTS OF ts = "+ts);
		System.out.println ("SIZE OF ts = "+ts.size ());
		Iterator itr=ts.iterator ();
		while (itr.hasNext ())
		{
			Object obj=itr.next ();
			System.out.println (obj);
		}
	}
};
```

Write a JAVA program which illustrates the concept of HashSet?
Answer:
```
import java.util.*;
class hsts
{
	public static void main (String [] args)
	{
		HashSet hs=new HashSet ();
		System.out.println ("CONTENTS OF hs = "+hs);
		System.out.println ("SIZE OF hs = "+hs.size ());
		hs.add (new Integer (17));
		hs.add (new Integer (188));
		hs.add (new Integer (-17));
		hs.add (new Integer (20));
		hs.add (new Integer (200));
		hs.add (new Integer (177));
		System.out.println ("CONTENTS OF hs = "+hs);
		System.out.println ("SIZE OF hs = "+hs.size ());
		Iterator itr=hs.iterator ();
		while (itr.hasNext ())
```

```
            {
                  Object obj=itr.next ();
                  System.out.println (obj);
            }
      }
};
```

Two dimensional framework or maps:

Two dimensional framework organize the data in the form of (key,value) pair. The value of key is an object and they must be unique. The value of value is also an object which may or may not be unique. Two dimensional framework contains collection of interfaces and collection of classes which are also known as map interfaces and map classes.

Map interfaces:

In maps we have three essential interfaces; they are java.util.Map, java.util.Map.Entry and java.util.SortedMap

java.util.Map:

java.util.Map extends Collection. An object of Map allows to organize the data in the form of (key, value) pair. Here key and value must be objects. An object of Map allows displaying the data in that order in which order we have added the data.

Methods:

public boolean put (Object kobj, Object vobj):

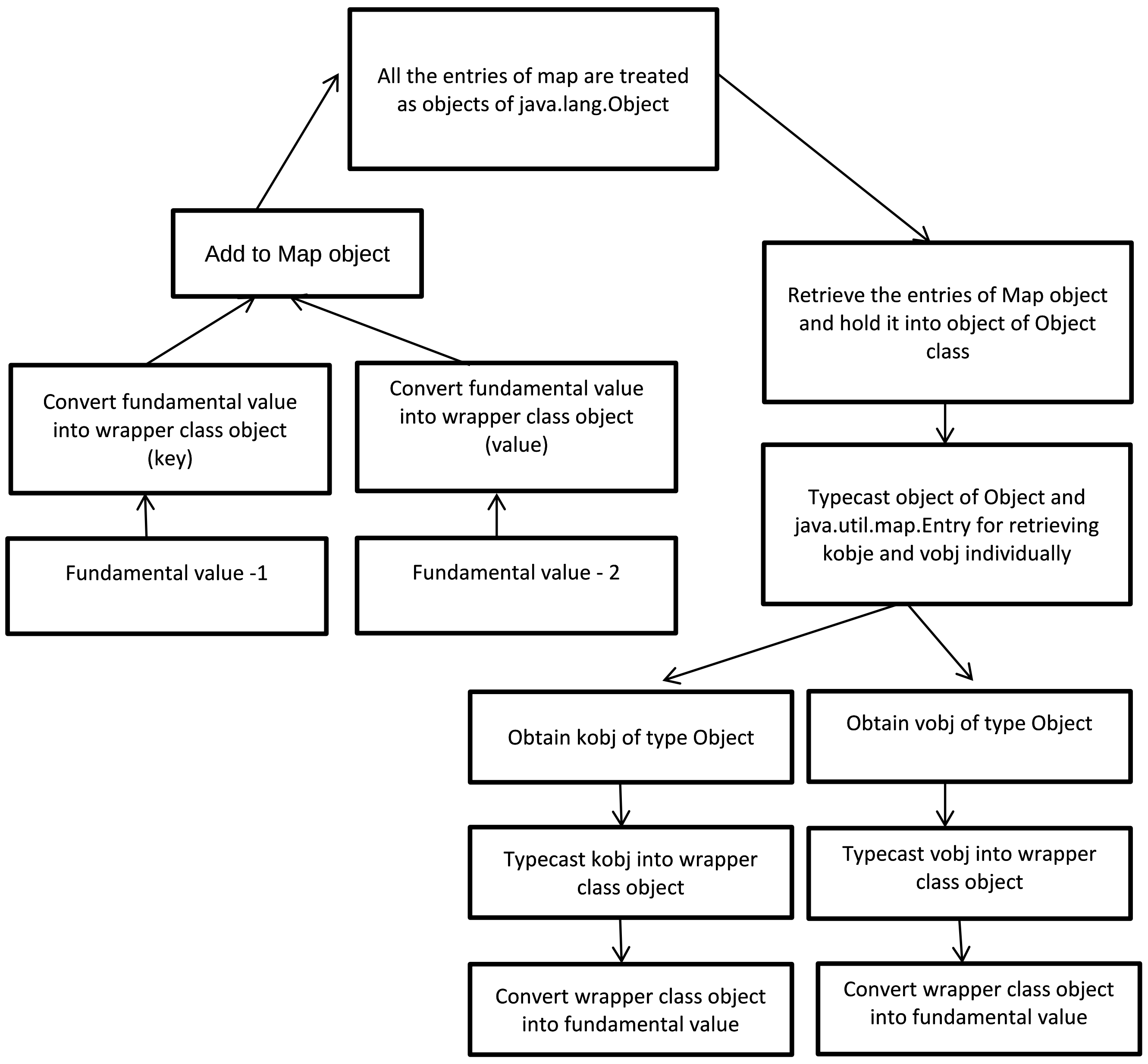
This method is used for adding the data in the form of (key, value). This method returns false when we are trying to add duplicate key and values. This method returns true as long as we enter unique key objects.

public boolean putAll (Map): This method is used for adding one Map object at the end of another Map object.

public Set entrySet (): This method is used for obtaining the data of the map in the form of Set object.

public Object get (Object vobj): This method is used for obtaining value of value by passing value of key object.

public void remove (Object kobj): This method is used for removing the entire map entry by passing the value of key object.


All the entries of map are treated as objects of java.lang.Object
Add to Map object
Convert fundamental value into wrapper class object (key)
Convert fundamental value into wrapper class object (value)
Fundamental value -1
Fundamental value - 2
Retrieve the entries of Map object and hold it into object of Object class
Typecast object of Object and java.util.map.Entry for retrieving kobje and vobj individually
Obtain kobj of type Object
Obtain vobj of type Object
Typecast kobj into wrapper class object
Typecast vobj into wrapper class object
Convert wrapper class object into fundamental value
Convert wrapper class object into fundamental value

NOTE:
The following diagram gives an idea about how to organize the data in the form of (key,value) and how to retrieve the data from Map object.

java.util.Map.Entry:
Here Map is an interface and Entry is the class in Map interface. java.util.Map.Entry is used for retrieving the data separately in the form of key object and value object from the Map object.
Methods:
```
public Object getKey (); → 1
public Object getValue (); → 2
```

Method 1 and 2 is used for obtaining key object and value object separately.

java.util.SortedSet:
SortedMap extends Map. An object of SortedMap displays the data by default in sorted order.

Methods:

```
        public Object first ();
        public Object last ();
        public SortedSet headMap (Object kobj); xi <= kobj
        public SortedSet tailMap (Object kobj); xi > kobj
        public SortedSet subMap (Object kobj, Object vobj); kobj1 <= xi < kobj2
```

Map classes:

Map classes contains all the definitions for the abstract methods of Map interface. In java.util.* package we have the following Map classes and whose hierarchy is given below:

1. AbstractMap implements Map
2. AbstractSortedMap extends AbstractMap implements SortedMap
3. HashMap extends AbstractMap
4. TreeMap extends AbstractSortedMap

HashMap	TreeMap
1. It extends AbstractMap.	1. It extends AbstractSortedMap.
2. It follows hashing mechanism.	2. It follows binary tree concept to obtain data in (k, v) form.
3. Retrieval time is more.	3. Retrieval time is less.
4. Insert, update and delete operations takes more time.	4. Insert, update and delete operations takes less time.
5. Creating an object of HashMap	5. Creating an object of TreeMap.
6. Random order.	6. Sorted order.

Creating Hashtable is nothing but creating an object of Hashtable class.

Write a java program which illustrates the concept of HashMap?

Answer:

```
import java.util.*;
class hmtm
{
        public static void main (String [] args)
        {
                HashMap hm=new HashMap ();
                System.out.println ("CONTENTS OF hm = "+hm);
                System.out.println ("SIZE OF hm = "+hm.size ());
                hm.put (new Integer (10), new Float(129.97f));
                hm.put (new Integer (1), new Float(143.93f));
                hm.put (new Integer (100), new Float(99.8f));
                System.out.println ("CONTENTS OF hm = "+hm);
                System.out.println ("SIZE OF hm = "+hm.size ());
                Set s=hm.entrySet ();
                Iterator itr=s.iterator ();
                while (itr.hasNext ())
                {
                        Map.Entry me= (Map.Entry) itr.next ();
                        Object kobj=me.getKey ();
                        Object vobj=me.getValue ();
                        System.out.println (vobj+"-->"+kobj);
                }
        }
};
```

Write a java program which illustrates the concept of TreeMap?

Answer:

```
import java.util.*;
class tmhm
```

```
{
	public static void main (String [] args)
	{
		TreeMap tm=new TreeMap ();
		System.out.println ("CONTENTS OF tm = "+tm);
		System.out.println ("SIZE OF tm = "+tm.size ());
		tm.put (new Integer (10), new Float(129.97f));
		tm.put (new Integer (1), new Float(143.93f));
		tm.put (new Integer (100), new Float(99.8f));
		System.out.println ("CONTENTS OF tm = "+tm);
		System.out.println ("SIZE OF tm = "+tm.size ());
		Set s=tm.entrySet ();
		Iterator itr=s.iterator ();
		while (itr.hasNext ())
		{
			Map.Entry me= (Map.Entry) itr.next ();
			Object kobj=me.getKey ();
			Object vobj=me.getValue ();
			System.out.println (vobj+"-->"+kobj);
		}
	}
};
```

Legacy collection framework:

When SUN Micro Systems has developed java, collection framework was known as data structures. Data structures in java were unable to meet industry requirements at that time. Hence data structures of java was reengineered and they have added 'n' number of classes and interfaces and in later stages the data structures of java is known as new collection framework we have one interface and classes.

Interface:
We have only one interface, namely java.util.Enumeration. This interface is used for extracting the data from legacy collection framework classes.

Classes:
As a part of legacy collection framework we have the following essential classes: Vector, Stack, Dictionary, Hashtable and properties. Here, Vector and Stack belongs to one dimensional classes whereas Dictionary, Hashtable and Properties belongs to two dimensional classes.

What is the difference between normal collection framework and legacy collection framework?
Answer: All the classes in the normal collection framework are by default belongs to non-synchronized classes whereas all classes in legacy collection framework are by default belongs to synchronized classes.

Vector:
Its functionality is exactly similar to ArrayList but Vector class belongs to synchronized whereas ArrayList belongs to non-synchronized class.

Creating a Vector is nothing but creating an object of java.util.Vector class.

Vector API:
Constructors:

```
	Vector (); → 1
	Vector (int size); →2
```

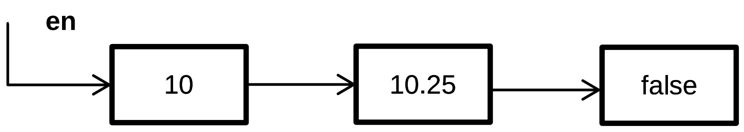
Instance methods:

```
	public int size (); → 3
	public void addItem (Object obj); [old] → 4
	public void addElement (Object obj); [new]   → 5
	public void addItem (int pos, Object obj); → 6
```

```
public void addElement (int pos, Object obj); → 7
public Object getItem (int pos); → 8
public Object remove (int pos); → 9
public void remove (Object obj ); → 10
public void removeAll (); → 11
public Enumeration elements (); → 12
```

The methods 4,5,6 and 7 are used for adding an object to the vector either at end or at specified position.
Method-12 is used for extracting the data from vector object.

```
Enumeration en=v.elements ();
While (en.hasMoreElements ())
{
	Object obj=en.nextElement ();
	System.out.println (obj);
}
```



Write a java program which listed the concept of Vector?
Answer:

```
import java.util.*;
class vector
{
	public static void main (String [] args)
	{
		Vector v=new Vector ();
		v.addElement (new Integer (10));
		v.addElement (new Float (100.37f));
		v.addElement (new Boolean (true));
		v.addElement ("SRINIVAS REDDY");
		System.out.println ("SIZE = "+v.size ());
		System.out.println ("CONTENTS = "+v);
		Enumeration en=v.elements ();
		while (en.hasMoreElements ())
		{
			Object val=en.nextElement ();
			System.out.println (val);
		}
	}
};
```

Methods in Enumeration method:

```
public boolean hasMoreElements (); → 1
public Object nextElement (); → 2
```

Method-1 is used for checking weather we have next elements or not. This method returns false when we have next element otherwise it returns false.
Method-2 is used for obtaining next element. Method-2 can be used as long as method-1 returns true .

Stack: Stack is the sub-class of Vector class. The basic working principal of Stack is Last In First Out.

Stack API:
Constructors:

```
Stack ();
Stack (int size);
```

Instance methods:

```
	public boolean empty (); → 1
	public void push (Object); → 2
	public Object pop (); → 3
	public Object peek ();     → 4
	public int search (Object); → 5
```

Method-1 returns true when the Stack does not contain any elements otherwise false.
Method-2 is used for inserting an object into the Stack.
Method-3 is used to remove top most elements permanently.
Method-4 is used to retrieve top most elements without removing.
Method-5 returns relative position of the element, if it found otherwise returns -1.

Write a java program which illustrates the concept of Stack?
Answer:

```
import java.util.*;
class stack
{
	public static void main (String [] args)
	{
		Stack st=new Stack ();
		System.out.println ("IS STACK EMPTY ? "+st.empty ());
		System.out.println (st);
		st.push (new Integer (10));
		st.push (new Integer (20));
		st.push (new Integer (30));
		st.push (new Integer (40));
		System.out.println (st);
		System.out.println ("TOP MOST ELEMENT = "+st.peek ());
		System.out.println (st);
		System.out.println ("DELETED ELEMENT = "+st.pop ());
		System.out.println ("MODIFIED STACK = "+st.peek ());
		System.out.println ("IS 10 FOUND ? "+st.search (new Integer (10)));
		Enumeration en=st.elements ();
		while (en.hasMoreElements ())
		{
			Object obj=en.nextElement ();
			System.out.println (obj);
		}
	}
};
```

Dictionary:
Dictionary is an abstract class, whose object allows to retrieve to store the data in the form of (key, value) pair. An object of Dictionary never allows duplicate values as key objects and null values.

Hashtable:
Hashtable is the concrete sub-class of duplicates and where object allows us to store in the form of (key, value) pair. Hashtable object organizes its data by following hashing mechanism. We cannot determine in which order the Hashtable displays its data.

Hashtable API:
Constructor:

```
	Hashtable ();
```

Instance methods:

```
	public boolean put (Object kobj, Object vobj); → 1
	public void remove (Object kobj); → 2
	public Object get (Object kobj); → 3
	public Enumeration keys (); → 4
```

Method-1 is used for obtaining value object by passing key object. If the key object is not found then the value object is null.
Method-2 is used for extracting key objects from Hashtable.

Write a java program which illustrates the concept of Hashtable?
Answer:
```
import java.util.*;
class hashtable
{
	public static void main (String [] args)
	{
		Hashtable ht=new Hashtable ();
		ht.put ("AP","Hyd");
		ht.put ("Orissa","Bhuvaneshwar");
		ht.put ("Karnatake","Bng");
		ht.put ("TN","Chennai");
		ht.put ("Bihar","Patna");
		System.out.println (ht);
		Enumeration en=ht.keys ();
		while (en.hasMoreElements ())
		{
			Object k=en.nextElement ();
			Object v=ht.get (k);
			System.out.println (k+"   "+v);
		}
		if (args.length==0)
		{
			System.out.println ("PASS THE STATE");
		}
		else
		{
			String st=args [0];
			Object cap=ht.get (st);
			if (cap==null)
			{
				System.out.println (cap+" IS THE CAPITAL OF "+st);
			}
		}
	}
};
```

Properties class:
Properties is the sub-class of Hashtable class. Properties class object is used for reading of maintaining system environmental variables and reading the data from resource data file or properties file.

Properties API:
Constructor:
```
	Properties ();
```
Instance Methods:
```
	public void setProperty (Object kobj, Object vobj);
	public Object getProperty (Object kobj);
	public void load (InputStream);
```

Write a java program which illustrates the concept of Properties class?
Answer:
```
import java.util.*;
import java.io.*;
class properties
{
	public static void main (String [] args)
```

```
        {
                try
                {
                        Properties p=new Properties ();
                        FileInputStream fis=new FileInputStream ("x.prop");
                        p.load (fis);
                        Object vobj1=p.get ("dno");
                        Object vobj2=p.get ("dname");
                        Object vobj3=p.get ("pwd");
                        System.out.println ("USER NAME : "+vobj2);
                        System.out.println ("DEPT NUMBER : "+vobj1);
                        System.out.println ("PASSWORD : "+vobj3);
                }
                catch (Exception e)
                {
                        System.out.println (e);
                }
        }
};
```

NETWORK PROGRAMMING:

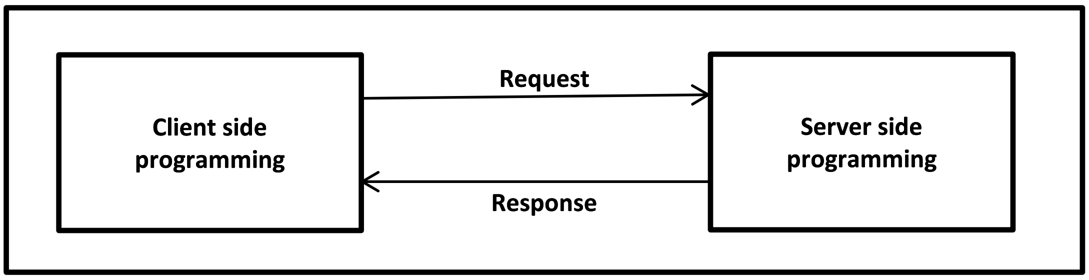
Collection of interconnected autonomous or non-autonomous computers is known as network. Autonomous represents independent processing power whereas non-autonomous represents dependent processing.

Aims of networking:

1. Fastest and reliable communication can be achieved.
2. Communication cost is so less.
3. Data can be shared either locally (centralized application) or globally (distributed application).

As a part of networking we write two types of programs. They are client side programming and server side programming.

- A client side programming is one which always makes a request to get the service from the server side program.
- A server side programming is one which receives client request, process the request and gives response back to the client.



In order to exchange the data between the client and server we must have a protocol. A protocol is the set of rules which exchange the data between client and server programs.

When we are developing any client and server application we must follow certain steps at client side and server side.

Steps to be performed at client side:

1. Connect to the server side program. To connect to the server the client side program must pass server name or domain naming services or IP address.
2. Make a request by passing data.
3. After performing some operations at server side, the client side program must receive the response from the server side.
4. Read the response data which is sent by the server.

Steps to be performed at server side:

1. Every server side program must run at certain port number (a port number is a logical numerical ID at which logical execution of the program is taking place).
2. Every server must have a physical name to access the services or programs which are running. Server can be access in two ways. On name (localhost by default) called DNS and on IP address (127.0.0.1 default).
3. Every server side program must receive client request.
4. Every server side program must read request data (request parameter).
5. Process the data or request.
6. Send the response back to the client.

Socket: In order to develop the program at client side we must use the class Socket.

Socket API:
Constructor:

```
	Socket (String hname, int Portno) throws UnknownHostException, IOException
```

Instance methods:

```
	public OutputStream getOutputStream ();  → 1
	public InputStream getInputStream (); → 2
	public void close (); → 3
```

Method-1 is used for establishing the connection with server socket by passing host name and port number of the server side program.
Method-2 is used for writing the data to the server side program. Method-3 is used for receiving or reading the response given by the server side program.
Method-4 is used for closing socket or client communication with the server.

```
	Socket s=new Socket (“localhost”, 7001);
```

ServerSocket: In order to develop the program at server side we must use the class ServerSocket.

ServerSocket API:
Constructor:

```
	ServerSocket (int portno) throws IOException → 1
```

Instance methods:

```
	public Socket accept (); → 2
	public void close (); → 3
```

Method-1 is used for making the server side program to run at certain port number.
Method-2 is used for accepting socket data (client data).
Method-3 is used for closing or terminating server side program i.e., ServerSocket program.

Write a java program which illustrates the concept of Socket and ServerSocket classes?
Answer:

```
server.java:
import java.net.*;
import java.io.*;
class server
{
	public static void main (String [] args)
	{
		try
		{
			int pno=Integer.parseInt (args [0]);
			ServerSocket ss=new ServerSocket (pno);
```

```
				System.out.println ("SERVER IS READY");
				while (true)
				{
					Socket s=ss.accept ();
					InputStream is=s.getInputStream ();
					DataInputStream dis=new DataInputStream (is);
					int n=dis.readInt ();
					System.out.println ("VALUE OF CLIENT = "+n);
					int res=n*n;
					OutputStream os=s.getOutputStream ();
					DataOutputStream dos=new DataOutputStream (os);
					dos.writeInt (res);
				}
			}
			catch (Exception e)
			{
				System.out.println (e);
			}
		}
};


client.java:
import java.io.*;
import java.net.*;
class client
{
	public static void main (String [] args)
	{
		try
		{
			String sname=args [0];
			int pno=Integer.parseInt (args [1]);
			Socket s=new Socket (sname, pno);
			System.out.println ("CLIENT CONNECTED TO SERVER");
			OutputStream os=s.getOutputStream ();
			DataOutputStream dos=new DataOutputStream (os);
			int n=Integer.parseInt (args [2]);
			dos.writeInt (n);
			InputStream is=s.getInputStream ();
			DataInputStream dis=new DataInputStream (is);
			int res=dis.readInt ();
			System.out.println ("RESULT FROM SERVER = "+res);
		}
		catch (Exception e)
		{
			System.out.println (e);
		}
	}
};
```

Disadvantages of networking:

1. We are able to develop only one-one communication or half-duplex or walky talky applications only.
2. We are able to get only language dependency (client side and server side we must write only java programs).
3. There is no support of predefined protocol called http.
4. There is no internal support of third party servers such as tomcat, weblogic, etc.
5. We are able to develop only intranet applications but not internet applications.

<u>DESIGN PATTERNS:</u>

Design patterns are set of predefined proved rules by industry experts to avoid recurring problems which are occurring in software development. As a part of information technology we have some hundreds of design patterns but as a part of J2SE so far we have two types of design patterns.
They are

1. Factory method and
2. Singleton class.

A method is said to be a factory method if and only if whose return type must be similar to name of the class where it presents.

Rules for factory method:

1. Every factory method must be public method.
2. Every factory method must be similar to name of the class where it presents.

A Singleton class is one which allows us to create only one object for JVM. Every Singleton class must contain one object on its own and it should be declared as private. Every Singleton class contains at least one factory method. Default constructor of Singleton class must be made it as private.

Write a java program which illustrates the concept of factory method and Singleton process?
Answer:

```
class Stest
{
	private static Stest st;
	private Stest ()
	{
		System.out.println ("OBJECT CREATED FIRST TIME");
	}
	public static Stest create ()
	{
		if (st==null)
		{
			st=new Stest ();
		}
		else
		{
			System.out.println ("OBJECT ALREADY CREATED");
		}
		return (st);
	}
};

class DpDemo
{
	public static void main (String [] args)
	{
		Stest st1=Stest.create ();
		Stest st2=Stest.create ();
		Stest st3=Stest.create ();
		if ((st1==st2)&&(st2==st3)&&(st3==st1))
		{
			System.out.println ("ALL OBJECTS ARE SAME");
		}
		else
		{
			System.out.println ("ALL OBJECTS ARE NOT SAME");
		}
	}
};
```

<u>STRING HANDLING:</u> A string is the sequence of characters enclosed within double quotes.

For example:

```
    “Java is a programming language”
```

In order to deal with strings we have two classes. They are java.lang.**String** and java.lang.**StringBuffer**

What is the difference between String and StringBuffer classes?
Answer: String class is by default not mutable (non-modifiable) and StringBuffer class is mutable (modifiable).
String API:
Constructors:

```
String (); → 1
String (String);     → 2
String (char []); → 3
```

Constructor-1 is used for creating empty string. Constructor-2 is used for creating a String object by taking another string parameter. Constructor-3 is used for converting sequence of characters into string.

Instance methods:

```
public char charAt (int); → 1
public int length (); → 2
public boolean equals (String); → 3
public boolean equalsIgnoreCase (String); → 4
public String concat (String);  → 5
public boolean startsWith (String); → 6
public boolean endsWith (String); → 7
```

Method-1 is used for obtaining a character from a string by specifying valid character position.
Method-2 is used for obtaining number of characters in the string.
Method-3 is used for comparing two strings. If two strings are equal in its case and meaning then this method returns true otherwise false.
Method-4 is used for comparing two strings by considering meaning by ignoring case.
Method-5 is used for concatenating two strings and the result is stored in another string.
Method-6 returns true provided the target String object present in first position of source String object otherwise false.
Method-7 returns true provided the String object present in last position of source String object otherwise false.

Static methods:

```
public static String valueOf (byte);
public static String valueOf (short);
public static String valueOf (int);
public static String valueOf (long);
public static String valueOf (float);
public static String valueOf (double);
public static String valueOf (char);
public static String valueOf (boolean);
```

These methods are used for converting any fundamental value into String object and this method is overloaded method.

```
public String substring (int start); → 1
public String substring (int start, int end); → 2
```

Method-1 is used for obtaining the characters from specified position to end character position.
Method-2 is used for obtaining those characters by specifying starting position to ending position.

What is the similarity between String and StringBuffer classes?

Answer: Both String and StringBuffer classes are public final. Hence, they are not extended by any of the derived classes and we cannot override the methods of String and StringBuffer class.

StringBuffer class:
Whenever we create an object of StringBuffer we get 16 additional characters memory space. Hence an object of StringBuffer is mutable object.

StringBuffer API:
```
1. StringBuffer ()
2. StringBuffer (String)
3. StringBuffer (char [])
4. StringBuffer (int size)
```

Constructor-1 is used for creating an object of StringBuffer whose default capacity is 16 additional characters memory space.
For example:
```
StringBuffer sb=new StringBuffer ();
System.out.println (sb.capacity ());
```

Constructor-2 is used for converting String object into StringBuffer object.
For example:
```
String s=”HELLO”;
StringBuffer sb=new StringBuffer (s);
System.out.println (sb.capacity ());
System.out.println (sb.length ());
```

Constructor-3 is used for converting array of characters into StringBuffer object.
For example:
```
char ch [] = {‘J’, ‘A’, ‘V’, ‘A’};
StringBuffer sb=new StringBuffer (ch);
System.out.println (sb);
```

Constructor-4 is used for creating an StringBuffer object with from specific size.
For example:
```
StringBuffer sb=new StringBuffer (256);
Sb=“JAVA IS AN APPLICATION”;
```

Instance methods:
1. **public int length ();**
This method is used for determining the length of the string.

2. **public int capacity ();**
This method is used for determining the capacity of StringBuffer object. Capacity of StringBuffer object is equal to the number of characters in the StringBuffer plus 16 additional characters memory space.
For example:
```
StringBuffer sb=new StringBuffer (“HELLO”);
System.out.println (sb.length ());
int cap=sb.capacity ();
System.out.println (cap);
```

3. **public StringBuffer reverse ();**
This method is used for obtaining reverse of source StringBuffer object.
For example:
```
StringBuffer sb=new StringBuffer (“HELLO”);
StringBuffer sb1=sb.reverse ();
System.out.println (sb1);
```

4. public void append (byte);
5. public void append (short);
6. public void append (int);
7. public void append (long);
8. public void append (float);
9. public void append (double);
10.public void append (char);
11.public void append (boolean);
12.public void append (String);
All the above methods mean (4 to 12) are used for appending the numerical data or the string data at the end of source StringBuffer object.

13.**public StringBuffer deleteCharAt (int);**
This method is used for removing the character at the specified position and obtaining the result as StringBuffer object.
For example:
```
	StringBuffer sb=new StringBuffer (“JAVA PROGRAM”);
	StringBuffer sb1=sb.deleteCharAt (8);
	System.out.println (sb1);
```

14.**public StringBuffer delete (int start, int end);**
This method is used for removing the specified number of characters from one position to another position.
For example:
```
	StringBuffer sb=new StringBuffer (“JAVA PROGRAM”);
	StringBuffer sb1=sb.delete (5,8);
	System.out.println (sb1);
```

15.**public StringBuffer replace (String, int, int);**
This method is used for replacing the string into StringBuffer object form one specified position to another specified position.
For example:
```
	StringBuffer sb=new StringBuffer (“JAVA PROGRAM”);
	System.out.println (sb1);
```

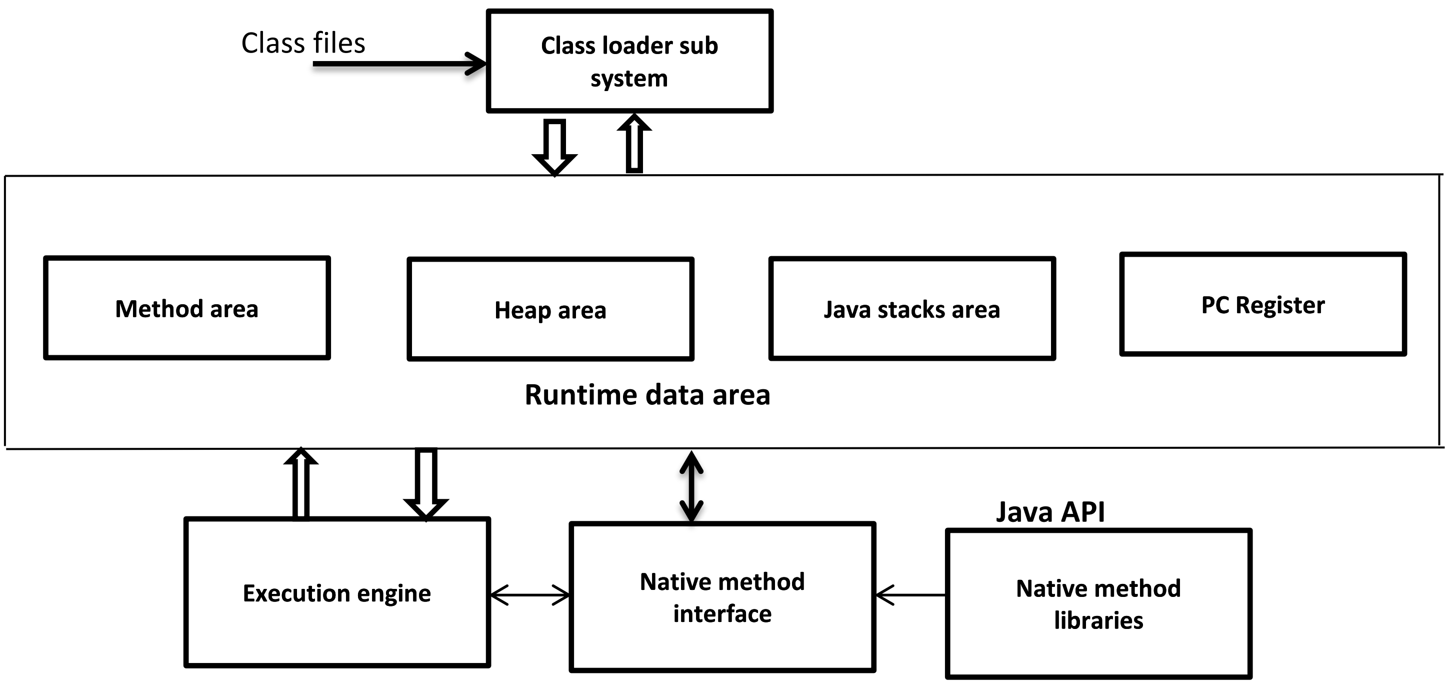
16.**public StringBuffer insert (String, int, int);**
This method is used for inserting the string data from one specified position to another specified position.
For example:
```
	StringBuffer sb=new StringBuffer (“JAVA PROGRAM”);
	StringBuffer sb1=sb.insert (“J2SE/, 0 ,3”);
	System.out.println (sb1);
```

In JVM we have three layers, they are:
1. Class loader sub system:
It is a part of JVM which loads the class files into main memory of the computer. While generating class files by the JVM it checks for existence of java API which we are using as a part of java program. It the java API is not found it generates compilation error.

2. Runtime data area:
When the class files are loaded into main memory JVM requires the following runtime data area,
they are
Heap memory: It is used for allocating the memory space for the data members of the class at runtime.

Java stacks/method area: In order to execute any method in java, JVM allocated some amount of memory space in the main memory on stack memory known as method area.
PC register: PC stands for Program Counter which is known as general purpose register. The PC register always gives address of next instruction of java program to the JVM.

JVM Architecture



3. Execute engine:

When a JVM is compiling in a java program in that context JVM is known as compiler. When a JVM is running a java program in that context JVM is known as interpreter.

JVM reads the byte code of .class line by line. Byte codes are set of optimized instructions which are used by JVM for generating the result of the java programs.

Method interface in the JVM architecture represents the middle man role between JVM (execution engine) and java API. Java API contains collection of packages.

TOOLS used in JDK:
Tools are nothing but the exe file which are developed by SUN micro system to run various applications of java, J2EE.
1. appletviewer:
This is a tool used for running applet application when the browser is not supporting properly.
2. jar:

jar stands for java archive. We use jar file for developing business component or application class (In J2EE applications we generate jar file for EJB applications).

war stands for web archive used for developing web components (Servlets and JSP are called web components).
Syntax for create war/jar file:
```
    jar cfv filename.war/jar *.class
```

For example:
```
    jar cfv sathya.jar *.class
```
3. java: It is used to run a java program.
4. javac: It is used to compile a java program.
5. javap: It is used to see the API of the specific class or interface which belongs to a specified package.

FEATURES OF JAVA

- Simple
- Platform
- Architecture Neutral
- Portable
- Network
- High Performance
- Interpreter
- Secured
- Dynamic
- Robust
- Multi threading
- Object Oriented Programming Language
  - class
  - object
  - Data encapsulation
  - Polymorphism
  - Dynamic binding
  - Message passing
  - Data abstraction

Single inheritance
Multi level inheritance
Hierarchial inheritance
Multiple inheritance
Hybrid inheritance

Physical level abstraction
Logical level abstraction
View level abstraction

Inheritance or Reusability

Data types

- Integer category data types
  - byte (1 – byte)
  - short (2 – bytes)
  - int (4 – bytes)
  - long (8 – bytes)
- Float category data types
  - float (4 – bytes)
  - double (8 – bytes)
- Character category data types
- Boolean category data types

Constants

- Final
  - At variable level
    - Final variable initialize
    - Final variable declaration
  - At method level
  - At class level

Keywords

- Instance
  - At variable level
  - At method level
- Static
  - At variable level
  - At method level
- this
  - this()
  - this(…)
- super
  - At variable level
  - At method level
  - At constructor level
    - Super()
    - Super(…)

String data into fundamental data

Wrapper classes

Byte	public static Byte parseByte(String);
Short	public static Short parseShort(String);
Integer	public static Integer parseInt(String);
Long	public static Long parseLong(String);
Float	public static Float parseFloat(String);
Double	public static Double parseDouble(String);
Character	public static Character parseChar(String);
Boolean	public static Boolean parseBoolean(String);

Object to fundamental data

- public byte byteValue();
- public short shortValue();
- public int intValue();
- public long longValue();
- public float floatValue();
- public double doubleValue();
- public char charValue();
- public boolean booleanValue();
- public String stringValue();

PrintStream classes

println methods

```
public void println(byte);
public void println(short);
public void println(int);
public void println(long);
public void println(float);
public void println(double);
public void println(char);
public void println(boolean);
public void println(String);
```

print methods

```
public void print(byte);
public void print(short);
public void print(int);
public void print(long);
public void print(float);
public void print(double);
public void print(char);
public void print(boolean);
public void print(String);
```

Object to String type

```
public String toString();
```

String object into wrapper class object

Wrapper class

Byte	public static Byte valueOfByte (String);
Short	public static Short valueOfShort (String);
Integer	public static Integer valueOfInteger (String);
Long	public static Long valueOfLong (String);
Float	public static Float valueOfFloat (String);
Double	public static Double valueOfDouble (String);
Character	public static Character valueOfCharacter(String);
Boolean	public static Boolean valueOfBoolean (String);

Constructors

- Default constructors
- Overloaded Constructors
  - Number of parameters different
  - Type of parameters different
  - Order of parameters different

Access specifiers

- private
- default
- protected
- public

All The Best *************************************