Core Java Programming

Introduction to Software

Software is collections of programs such as C , C++ , Java, SQL and other programs.
Software basically two categories:

Software	
System software	**Application Software**
it interacts with hardware components Eg: Device drivers and Operating systems	for the purpose of application development Eg: C, C++, Java, .Net , ERP package and others Except Operating Systems, all are Application software

Device Drivers:

These are programs to communicate with hardware components Every device has its own drivers.
Device drivers are supplied by hardware vendors
Eg: printer driver, network drivers

Operating System:

Q. Can we run any program without OS ?
Ans: No
Except BIOS, to run all programs OS required .

BIOS Basic Input and Output System.

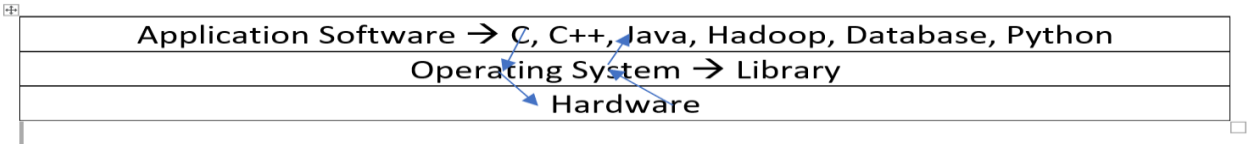
BIOS is a hardware program, it is a chip level program

BIOS is responsible to boot Operating System

Booting is a process of loading OS from Hard Disk to RAM memory.

What is an Operating System ?

1) It is a platform, where we can run all the application software.
2) It is an interface between hardware and application software.
3) Every Operating System has its own libraries to communicate between Application software and Hardware components.
4) Every program has to link with OS library before going to run.



Application Software	
Front End Software	**Back End Software**
These are to interact with end users, it collects data from end users and stores into backend Eg: Languages, Packages ERPs	These are used to store and maintain data Eg: Databases and files , HDFS

Front End Software	
Languages	**Packages**
Programmer has to write coding. Complexity in development of application Eg: C, C++, JAVA, HTML, .Net	It is a readymade one, ready use. Easy to develop the applications MS-Office, ERP packages, Tally Accounting package

Languages	
Low Level Languages	**High Level Languages**
These are machine (processor) understandable languages Eg:	These are user understandable languages These are looks likes English language Easy to understand.

Machine Language (IGL) and Assembly Language (II GL) Assembly language in the form of pneumonic codes Assembler converts assembly language into machine language	III GL and IV GL are high level languages

Software Languages

==========

1) Computer Understandable Language
2) User Understandable Language
3)Computer Understandable Language

==========

that depends on Generation of Computers.

There are five generations
I Gen --> Vaccume Tubes based
II Gen --> Transister based
III Gen --> IC based
IV Gen --> Processor based (present)
V Gen --> AI based

Processor based Computer Understandable Language

it is binary code (10011001011)
it is a System Understandable language
System --> processor

AI based understandable language, is high level language like, R, java and Python
Software languages are two Categories
1) Low Level Language
2) High Level Language

1) Low Level Language:

==========

it is a System understandable language
it is in the form of binary code (11000010010101010)
System --> Processor

It is a **Processor** understandable language
Processor --> 8085 and 8086
it is called as Machine Language

2) High Level Language:

==========

it is like a English Language it is very easy to Understand

Q. Which is a 1st Developed high level language ?

C - Language, is first high level language developed by Dennis Ritchie in the year 1972.
Eg: C, C++, Java, Python, HTML, .net C#, Scala..
every High Level Language program has to convert into binary code (10100101110)
High Level Language ---> binary code
Interpreter or Compiler converts from High Level Language to binary code .

Generation of Languages

Low Level Languages	I GL	Machine Language , It is in form of binary code (0 1 0 1 1 0 1 1 0)
	II GL	Assembly Language in form of pneumonic codes Assembler coverts assembly language into binary code
High Level Languages	III GL	(procedure oriented languages) Cobol, Pascal, Basic, FORTRAN , Python
	IV GL	(functional, OOP, Object Based languages) C, C++, Java, . Net

Procedure and Function, both are set of statements to perform some task,
The difference is that, **function** returns a value whereas **procedure** does not return a value.
Compliers and interpreters are used to convert form high level languages into low level language
(processor understandable language)

Difference between compiler and interpreter

S.No	Interpreter	Compiler
1	It interprets line by line and executes	Whole program at a time
2	It gives the result line by line	It gives whole output at a time
3	If any error occurs interpreter stops Hence it shows only one error	It checks all statements in the program and shows all the errors in the program.
4	It will not generate executable file	If no errors in the program, then it generates executable file
5	It always executes only source code	It executes exe file
6	all scripting languages	all programming language
7	Eg: HTML, PERL, Java Script, Python,Shell	Eg; C, C++ ,java and .Net C#
8	Interpreter based languages don't have exe file concept	Every program converts into exe file
9	Open Source, it can be modified any time	It is not an open source , it is exe based , it's code can not be modified

Java depends on both Compiler as well interpreter In Java Errors Checking by the **Compiler**
If no errors , then Java compiler generates .class file .class file is a byte code file
Java runtime by the **Interpreter**
Source code (.java) **javac (java compiler)** Byte Code file (.class) **java (interpreter)** byte code to binary and links with OS Library Output

Source Code Byte Code Binary Code Processor Output
Source Code Byte Code java compiler
Byte Code Binary Code java interpreter
Java depending on Both Compiler as well as Interpreter
Java runtime is an Interpreter based.
Any language, runtime is an interpreter based, no exe file.
Since Java runtime is an Interpreter based, No exe file concept in Java.

Q.Can we create exe file in Java
Ans : No
There is no Concept of exe files in Java,

In Java every thing is a .class file. .class file is not an exe file.
Exe file is binary code file and it is a processor Understandable language
.class file is a byte code file and it is a JVM Understandable Language
Exe file is an OS dependent ,
.class file is an independent of platform

JAVA

Features of Java :
1) It is an independent of platform.
2) It is a portable (write once run any where).
3) It is a scalable language.
4) It provides security for local resources.
5) Object Oriented Programming language.

Platform:
Any hardware or software environment in which a program runs is known as a Platform.
Since Java has its own runtime environment (JRE) and API, it is called Platform.
JVM is the responsible to make java application as independent of platform and portable
1) It is an independent of platform Java application runs any operating system and any Hardware
2) It is a portable (write once run any where)Java application can be moved any platform with any changes to current application

3) It is a scalable language Easy to use without disturbing the existing java functionality
4) It provides security for local resources. JVM has Security Manger which is responsible to prevent java application to access your private information in your PC.

5) Object Oriented Programming language In java every transaction handled by object, without object we cannot perform any transactions, hence it is called as Object Oriented Programming (OOP) language.

Difference between Exe file and .class file

Exe file	.class file
1) It is a binary code file	It is a byte code file
2) It is a system understandable language	It is a JVM understandable language
3) It contains program code + Software Library + OS library	It is a Pure Java program in byte format. Don't have any OS library It contains byte code for java program + Byte Code for Java Library
4) Exe file is a prepared file for specific OS	.class file is not a prepared file, it don't have any OS library
5) OS library, statically linking with exe file	OS library dynamically linking with .class file at run time .
6) Exe file is already prepared one, before moving OS	.class file is not a prepared file , after moving on to OS, JVM prepares as per current OS
7) Exe file executes faster than .class file	Since .class file preparing at runtime , it executes slowly than exe file
8) To run exe file , Application Software not required to install	To run .class file JDK should be installed on OS.

What is java ?
Java is compile as well as Interpreter based. Java compiler coverts source code into byte code file
Java Interpreter converts byte code into binary as per the Operating system and Hardware.

What is an exe file ?
Exe file contains binary code, It is a file with actual program code, application software library, and OS library and hardware information Exe file is OS dependent as it contains OS library Exe file is prepared file for specific operating system.
Eg:
Any exe file prepared on windows will not execute on Linux and vice versa Java byte code files not an exe file.
Java Editions:
There are Four Editions in java :

1. **Java SE = Standard Edition**
   This is the core Java programming platform. It contains all of the libraries and APIs that any Java programmer should learn (java.lang, java.io, java.math, java.net, java.util, etc...).
2. **Java EE = Enterprise Edition**
   It is platform, collection of services. We can develop Web applications and Enterprise applications.
3. **Java ME = Micro Edition / Mobile Edition**
   This is the platform for developing applications for mobile devices and embedded systems such as set-top boxes. Java ME provides a subset of the functionality of Java SE, but also introduces libraries specific to mobile devices.
4. **Java FX:**
   JavaFX is a Java library used to build Rich Internet Applications (IoT, Internet of Things) The applications written using this library can run consistently across multiple platforms. The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc..
   JavaFX, Java programmers can now develop GUI applications effectively with rich content.

JSE Features:
1) Object-oriented programming language.
2) Java supports exception Handling
3) File I/O transactions
4) Collections
5) Strings and String Buffer Handling.

6) JDBC (Java Data Base Connectivity) to data base applications.
7) Networking.

8) Multi-threading.
9) Applets for Web applications -- >It was outdated concept.
10) AWT (Abstract Windowing Tool Kit) for GUI Applications --> It was outdated concept.

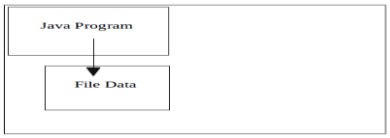
JEE Features :-
It is a collections Services such as
1) Servlets
2) JSP
3) RMI (Remote Method Invocation)
4) EJB (Enterprise Java Beans)
5) Java Mail API

6) Java Message Service
7) Java Naming and Directory Interface (JNDI)
8) JTA (Java Transaction API)
9) XML Parser
10) Web Services
11) Restful Services

JEE Provides Below Frameworks
1) Struts Framework
2) Spring Framework
3) Hibernate Framework.
4) Angular JS

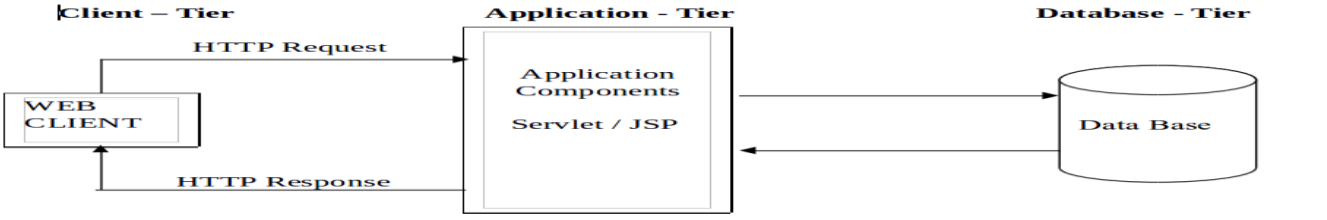
5) Node JS
6) Node 2
7) Hadoop Frame work
8) Spark Frame work
9) Ignite Frame work

Types of Java Applications
There are mainly 6 types of applications that can be created using java programming:
1) Standalone Application (single – tier)
In this applications, data and application both resides in same memory space.
Eg:
java program processing a file data. C- program processing a file data. Any Application processing files data.
It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications
Java Programme and Data in the same memory space.



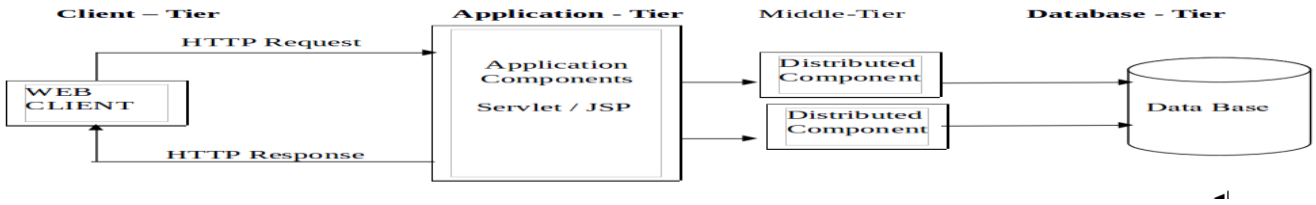
2) Client and Server Applications: (Two – Tier)
These are TCP/IP based application, one to one communication always. Here no browser and web server. One server but can be multiple clients,
but communication is always one to one, synchronous (one after another)

3) Web Application (three – tier)
An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.



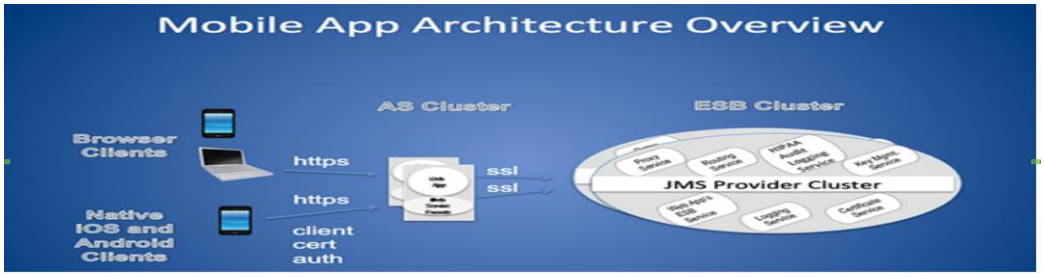
4)Enterprise Application (N- Tier)

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications. It will use middleware distributed components, to improve the performance of application.



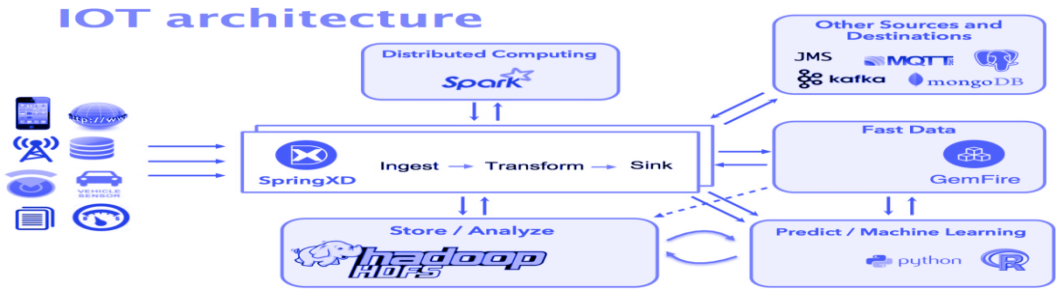
5) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.



6) IOT Applications:

IoT (Internet of Things) is an advanced automation and analytics system which exploits networking, sensing, big data, and artificial intelligence technology to deliver complete systems for a product or service. These systems allow greater transparency, control, and performance when applied to any industry or system.



Difference between C++ and Java :

1) C++ is complier based, which has exe file,
Java is a compiler as well interpreter based, Java compiler generates .class file, contains byte code, which is independent of platform. As result of C++ compiler is exe file, it is a platform dependent.
2) Java does not Support C++ pointers
3) No C-Language structures in Java
4) No sizeof operator in java
5) In java every thing is inside of the class.
6) Java is an Object Oriented language whereas C++ is a functional oriented as well as Object oriented language and C – Language is a functional oriented language.
7) Like C++ no friend functions in Java
8) No destructors concept in java
10) Java compiler doesn't interact with OS kernel
11) Java doesn't provide multiple inheritance

Different Versions of Java.

S.No.	Version	Code Name	Year
1.	JDK Alpha and Beta		1995
2.	JDK 1.0	Oak	1996
3.	JDK 1.1	Oak	1997
4.	J2SE 1.2	Playground	1998
5.	J2SE 1.3	Kestrel	2000
6.	J2SE 1.4	Merlin	2002

7.	J2SE 5.0/1.5	Tiger	2004
8.	Java SE 6/1.6	Mustang	2006
9.	Java SE 7/1.7	Dolphin	2011
10.	Java SE 8/1.8	code name culture is dropped	2014
11	Java SE 9/1.9		2016

Java Installation :

Installing Java

Place the JDK1.8 into C-Drive and double click yes next next next close

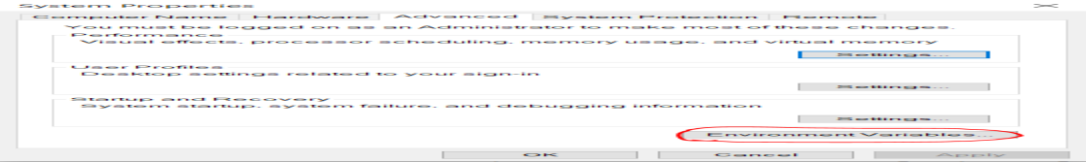
Setting Environment Variable for Java

Win-10

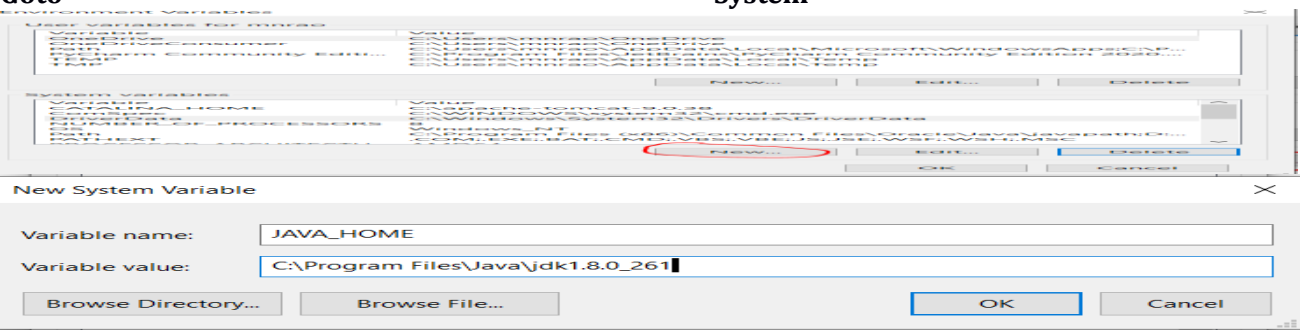
Right click on **This PC** Icon select properties

Select Advanced System settings

Click on Environment variables as below

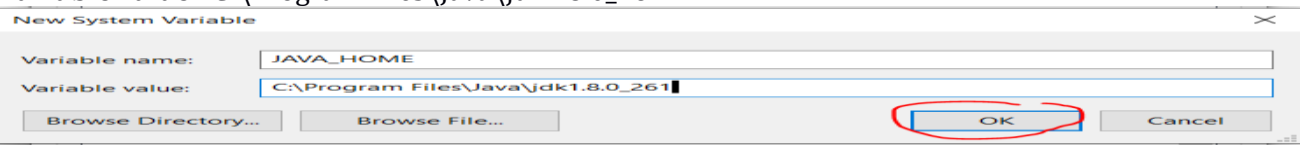


Goto System variables



Variable Name : JAVA_HOME

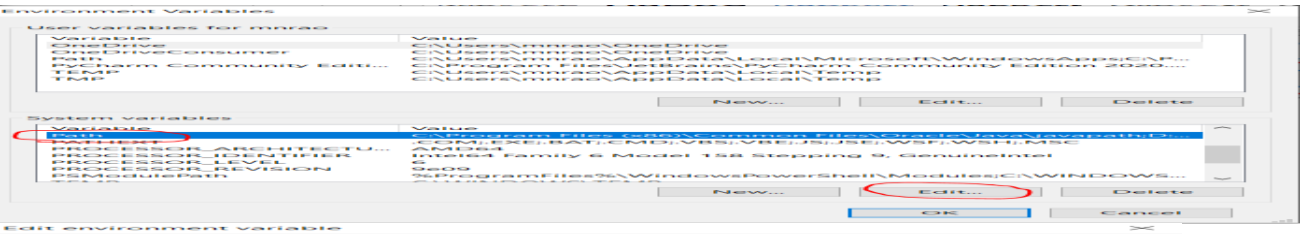
Variable value : C:\Program Files\Java\jdk1.8.0_261

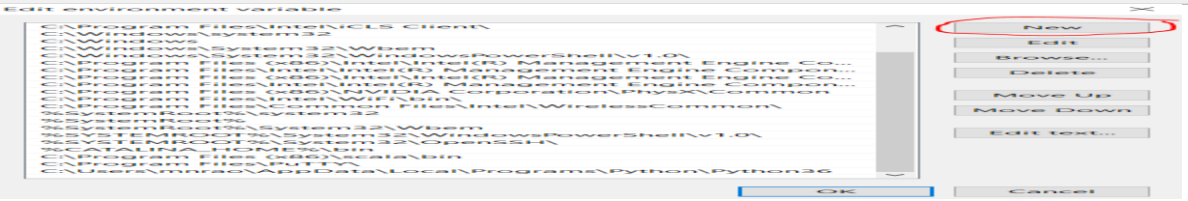


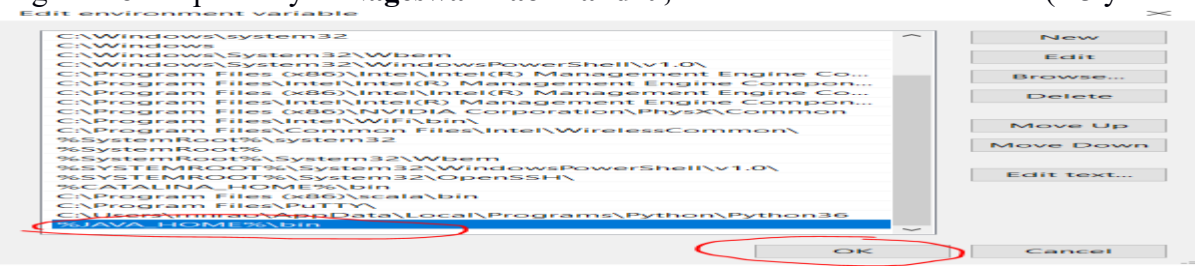
Setting path :

Goto system variables

Select **PATH** variable and **edit**







Click on OK ⇨ OK ⇨ OKSetting Java Path in Linux OS:

$gedit ~/.bashrcgo to end of .bashrc file and provide following PATH

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386

export PATH=$PATH: $JAVA_HOME/bin

checking for java installation and version:

```
Command Prompt
Microsoft Windows [Version 10.0.18363.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\mnrao>java -version
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)

C:\Users\mnrao>
```

How to check value of OS environment Variables at command line

Ans:

set is command

- **set JAVA_HOME**

```
C:\Users\mnrao>set JAVA_HOME
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_121

C:\Users\mnrao>
```

```
C:\Users\mnrao>set PATH
Path=C:\ProgramData\Oracle\Java\javapath;D:\app\oratest\product\12.2.0\dbhome_1\bin;C:\Program Files (x86)\Intel\iCLS Client\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Apache-tomcat-9.0.38\bin;C:\Program Files (x86)\scala\bin;C:\Users\mnrao\AppData\Local\Programs\Python\Python36\Scripts\;C:\Program Files\Java\jdk1.8.0_121\bin;C:\Users\mnrao\AppData\Local\Microsoft\WindowsApps;C:\Program Files\Microsoft VS Code\bin;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
```

What is purpose of java PATH setting ?

Ans:

Operating System, to identify the location of Java Software.

Eclipse setup:

Take the eclipse zip (archive) and place into C-dirve ⇨ rt.click extract here⇨ Open Eclipse folder ⇨ right click on eclipse icon ⇨ send to desk top ⇨ created short cut on desk for eclipse.

On launching eclipse, it check for the JAVA_HOME, then integrates jdk with eclise.

Working with eclipse:

Eclipse is an IDE

IDE: Integrated Development Environment.

Java Compiler, JRE, JVM are integrated to eclipse automatically, when PATH variable is defined in the OS.

Steps to Develop Java Project in Eclipse

1) Create Workspace
2) Create a Project
3) Create a Package
4) Create a Class
5) Create Methods
6) **Step1:**

Create Workspace

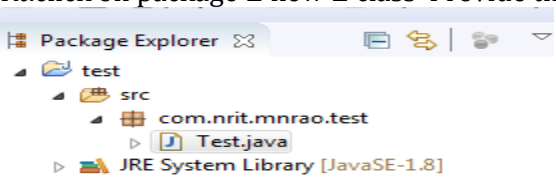
Goto Desk top and double clikc on Eclipse Icon, then Launch eclipse ⇨ Select Work Space location (browse) and provide the name , D:\test\SampleWS SampleWS ⇨ is a workspace name⇨ Ok

Close the welcome screen.

Step2:

Create Project

Rt.Click on Project Explorer area new project java project next
provide project name (test) execution environment : JavaSE1.8
Finish
Step3:
Create package
Expand your project rt.click on **src** new package Package name (lower case) : <com/org>.<client_name>.<project_name>.<module_name> com Company org Organization
Eg:
com.nrit.track.admin
Step4:
Create a class
Rt.click on package new class Provide the class name : Test




```
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

How to run java program Rt.click on main() program file runAS java Applicaton
At Console O/p:
Hello World
How to write sample program in java: Hello World program:
D:\>test>notepad Test.java windows

```
import java.lang.String ;
import java.lang.System ;
public class Test {
        public static void main(String [] args ){
                System.out.println("Hello World");
                System.out.println("Hello NRIT");
        }
}
```

Save this file as **Test.java** Name of program file should be same as the class name in which main method is defined.
To compile: **D:\test>** javac Test.java
Result file : Test.class
To execute: **D:\test>** java Test
Commenting :
to write description about the program.

```
// Single line commenting in java
/* multiple lines
  commenting
  in java
*/
```

import is a keyword to import java package,
java.lang, java.io, java.net, java.util, java.math, java.sql
Q. what is a package ?
package is a collection of related classes and interfaces.
import java.lang.* ,
Here * all the classes from that package
Below is an individual import

```
import java.lang.String ;
import java.lang.System ;
```

Q. What is the purpose of importing package ?
Ans:

purpose of import is to use pre-defined classes and interfaces
Note :
individual importing is recommended .* will import un necessary classes. Individual import, it will import only required classes. importing **java.lang** is an optional, as it is a default available package in java applications.
Any class from java.lang package , not necessary to import .

class is a keyword is used to define a classes in java.
Test : is name of the class.
How to choose class Name

1) Naming Conditions
   These are as per the java compiler.
2) Naming Conventions
   These are as per the coding standards .

Naming conditions of a class :
These are as per the compiler.

1) It contains only alphabets (a-z, A-Z), digits (0 – 9) and under score (_)
2) Should not be used spaces and special chars, keywords.
3) Max length can be up to 255 chars

Naming conditions are same for all identifiers
identifiers:
=======
class name, interface name, variable name, object name, method name
Naming conventions of a class :
These are as per coding standards for easy understanding the code.
1) class name should start with upper case alphabet

2) If class name contains multiple words then every word should start with upper case alphabets **(Camel Case in java naming conventions)**

Eg :

ContractEmployee
MyFirstExample
ActionEvent, ActionListener

Java follows camel case syntax for naming the class, interface, method and variable.

public keyword is an access specifier which represents visibility, it means it is visible to all. No two classes are public in the same program file. A class must be public which name is same as the file name
static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method.
main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
main represents start up of the program. To compile java program, main() is not compulsory but to execute main() should presented.
Naming conditions of method are same as the class as explained above.
Naming conventions of a method :
1)Name of the method starts with lowercase alphabets
2) if name contains multiple words, then first word starts with lowercase and next words starts with uppercase alphabet
e.g.
getData(), setData(), getEmpInformation()
actionPerformed(), firstName(),

setEmpDeptName()

getDatabaseConnection()

actionPerformed()

getRateOfInterest()

getMinRateOfInterest()

getMaxRateOfInterest()

void is the return type of the method, it means it doesn't return any value.
String[] args is used for command line argument. We will learn it later.
System.out.println () is used print statement.

print() and println ()
println () inserts a new line char at the end of the data, where print() does not.
System: it is a class from java.lang package

It provides three objects
1) in System.in : to read data from input buffer
2) out System.out : to write data to output buffer
3) err System.err : to Write JVM implicit error message to output buffer

A program file can have multiple classes but only one class must be public A class must be public, which name is same as file name. Public class name and file name should be same.
In the following program which main() executes ?
Program file is Test.java

```
class Sample {
        public static void main(String [] args ) {
        System.out.println("Hello NRIT");
        }
}
public class Test{
        public static void main(String [] args ) {
                System.out.println("Hello World");
        }
}
```

Ans : Test class main()
o/p: Hello World

JVM looks for the main(), in a class which name is same as the file name. Why public class name , should be same as file name ?
Ans:
JVM to identify the location of main()
Q. Can I compile empty .java file ?

Yes , an empty .java program file can be compiled but not generates .class file Naming conditions are same for all identifiers .
Naming convention are as below.

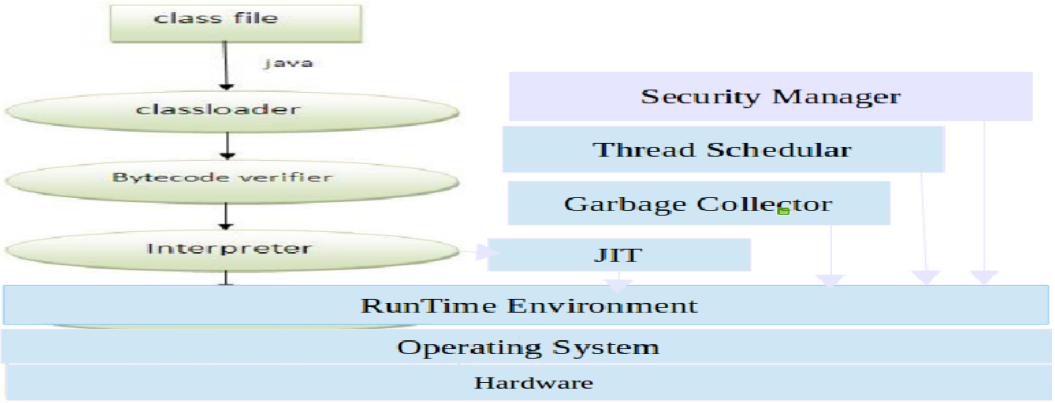
Name	Conventions
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println () etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
Object name	should start with lowercase letter e.g. contractEmployee , permanentEmployee etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc. com.durga.mnrao.test
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

What happens at runtime?
At runtime, following steps are performed:

JVM performs following:
Test.java --> javac --> Test.class --> JVM

Overview of JVM

Internal Architecture, covered at the end of this Document.



Class Loader : Loading .class file into memory
Byte code verifier :
Byte code Verification

When a class loader presents the byte codes of a newly loaded Java platform class to the virtual machine, these byte codes are first inspected by a *verifier.* The verifier checks that the instructions cannot perform actions that are obviously damaging. All classes except for system classes are verified.

Here are some of the checks that the verifier carries out:

- Variables are initialized before they are used.
- Method calls match the types of object references.
- Rules for accessing private data and methods are not violated.
- Local variable accesses fall within the runtime stack.
- The runtime stack does not overflow.

Interpreter :

It coverts java byte code to binary code and prepares as per the current platform (Hardware) runtime environment.

JIT :
Just-In-Time (JIT) compiler is a component of the Java Runtime Environment that improves the performance of Java applications at run time
The JIT compiler helps improve the performance of Java programs by converting byte code into native machine code at run time.

Garbage Collector :

Garbage collector is a daemon thread. CPU is responsible to call Garbage collector. If there are no threads running in the system, then CPU makes a call to Garbage collector. Java provides facility to Java developer for invoking Garbage collector through Runtime.gc() / System.gc() Task of Garbage Collector is to clear all un-used objects(Garbage objects)Unreferenced objects are called as Garbage objects.

Thread Schedular:

Thread schedular is to manage different states of thread in a multi threaded application.

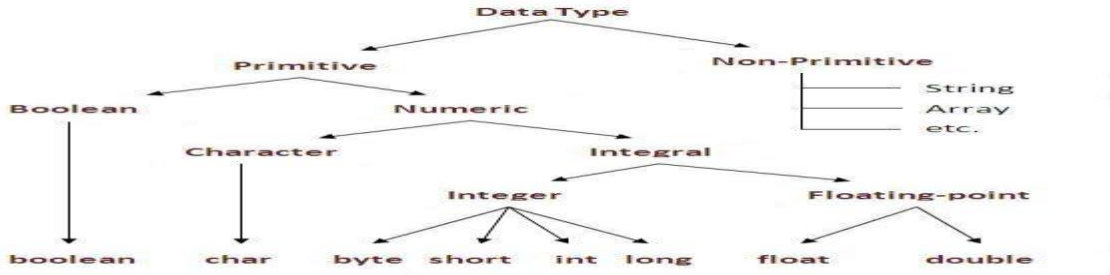
Security Manager:

Security Manager is to provide the security for local system resources from Java Application.

Data Types in Java

In java, there are two types of data types

1) Primitive data types (basic data types / System defined)
   System defined data types
   System --> compiler
   these are from the compiler
   these are also called as pre-defined, fundamental types, basic data types
2) Non-primitive data types
   all java classes (OOP)



Data type	Default value	Size	Range of data
Boolean	False	1 bit	1 (true) or 0 (false)
Byte	0	1 byte	-128 to 127, -2^7 to $(2^7 - 1)$
Short	0	2 byte	-32,768 to 32,767 , -2^{15} to $(2^{15} - 1)$
Int	0	4 byte	-2^{31} to $(2^{31} - 1)$
Long	0L	8 byte	-2^{63} to $(2^{63} - 1)$
Float	0.0f	4 byte	7 significant decimal digits
Double	0.0d	8 byte	15 significant decimal digits
Char	'\u00000'	2 byte	0 to 65535 (unsigned char)

Why char uses 2 bytes in java and what is \u0000 ?
Java char is a unicode character. It is because java uses Unicode system than ASCII code system.
ASCII code system accepts only ASCII chars key board set(0 to 255) .

Unicode system accepts any key board set across universe.
Size of ASCII char is one byte.
Size of unicode char is two bytes.

Java is a net based product, to accept all keyboard sets across the universe it is designed with 2 bytes.
Char with 2 bytes can accept the value ranging from 0 to 65535 (many keyboard sets)
1 byte = 8 bits

1 KB ≈ (10^3) = 1024 bytes
1 MB ≈ (10^6) = 1024 KB
1 GB ≈ (10^9) = 1024 MB
1 TB ≈ (10^{12}) = 1024 GB ≈ 10^3
1 PB ≈ (10^{15}) = 1024 TB
1 EB ≈ (10^{18}) = 1024 PB
1 ZB ≈ (10^{21}) = 1024 EB

TB to ZB ≈ is a Big Data.
Hadoop is frame work to store and Process Bigdata.
There are two modules in Hadoop
1) HDFS --> it is file system to store bigdata using as backend

2) MapReduce --> java programs to process data using like a front end

Storage of Data in memory

Numeric to Binary Conversion

byte b = 10;

```
        2 10
        2  5  --  0
    2   2  --  1
    2   1  --  0
        0  --  1
1 0 1 0
```

b (size of b = 1 byte = 8 bits as below) first bit

0	0	0	0	1	0	1	0

Last bit is for sign

0 → +ve

1 → -ve

Binary to numeric format conversion

b (size of b = 1 byte = 8 bits as below) first bit

2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	1	0	1	0

Last bit is for sign

0 → +ve

1 → -ve

$0*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$

$0 + 0 + 0 + 8 + 0 + 2 + 0 = 10$

If all data bits are 1's , then it is a max value

b (size of b = 1 byte = 8 bits as below) first bit

2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	1	1	1	1	1

Last bit is for sign

0 → +ve

1 → -ve

2 1 + 2 + 4 + 8 + 16 + 32 + 64 = 127

1 byte max value = 127

2 bytes of memory first bit

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1

2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	

Last bit is for sign

0 → +ve

1 → -ve

1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512 + 1024 + 2048 + 4096 + 8192 + 16384 = 32767

2 bytes max value = 32767

4 bytes of memory

first bit

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1

2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^{9} 2^{8}

Last bit is for sign

1 byte max value = 127

2 bytes max value = 32,767

Character type

==========

Why char uses 2 bytes in java and what is \u0000 ?

size of the char = 2 bytes , why ?

why size of the char is 2 bytes in Java ?

there are two types of characters

1) ASCII char --> American Standard Code for Information and Interchange

size of ASCII char = 1 byte

2) Unicode --> Universally Accepted Code

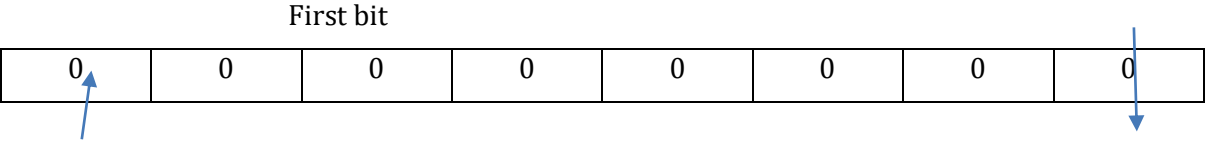
size of Unicode char = 2 bytes

character is an unsigned type (there is no sign for character)

ASCII char = 1 byte

unsigned 1 byte , what is the min value.

First bit

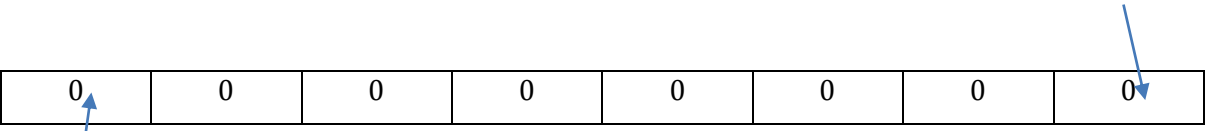


Last bit, is also for data (there is no sign)

In Unsigned type , there is no sign, all bits used for data only.

If all bits are 0's

First bit



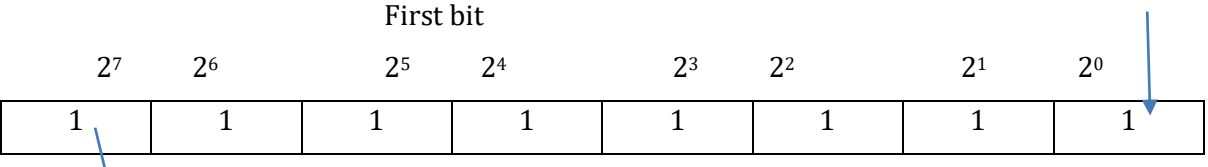
Last bit is also for data

If all bits are 0's then it is a min value = 0

unsigned 1 byte , what is the max value.

First bit

2^{7} 2^{6} 2^{5} 2^{4} 2^{3} 2^{2} 2^{1} 2^{0}



Last bit is also for data

If all bits are 1's then it is a max value = 255

$1*2^{7}+ 1*2^{6} + 1*2^{5} + 1*2^{4} + 1*2^{3} + 1*2^{2} + 1*2^{1} + 1*2^{0}$

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1= 255

Unsigned 1 byte Range = 0 to 255

ASCII char Range = 0 to 255

What is ASCII value ?

On key board for every key some integer number given to convert from char to binary format .

That number is an ASCII value.

ASCII value is ranging from 0 to 255

All keys ASCII value between 0 to 255 only

Eg:

A-Z 65 to 90

a-z 97 to 122

0-9 48 to 57

Why Uncode Char introduced in Java ?

Java is net based product, Java app works on the Internet.

If character type is ASCII char it can support char ranging from 0 to 255 only

ASCII chars only (USA, English) ASCII key board.

If char size is only 1 byte it supports only USA, English key board , not other keyboard sets.

ASCII keyboard set Registered from 0 to 499 chars.

Any key board set released in the future that should be started from 500.

Sunsoft has developed standard for every keyboard with max only 500 keys and same thing was Registered.

If it is ASCII char , it can take only USA, English , it does not accept other keyboard sets.

0 to 499 Registered for USA, English.

To accommodate other keyboard sets also char designed with 2 bytes .

Char with 2 bytes of memory called as Unicode char (Universally accepted code)

Unsigned 2 bytes what is max value ?

2 bytes 2^0

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

2^{15}

1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512 + 1024 + 2048 + 4096 + 8192 + 16384 + 32768 = 65535

Unsigned 2 bytes max value = 65535.

It's Range = 0 to 65,535.

If size of the char is 1 byte it can take only USA, English.

If size of the char is 2 bytes , it can take many keyboard sets across the world wide.

To support different languages in the Universe char designed with 2 bytes.

Size of Unicode char = 2 bytes , it's max value = 65,535

Size of each key board = 500 keys

No of keyboard sets supporting by Unicode char = 65535/500 = 131

No of langauges that can be used with Unicode char = 131

To support different languages in the Universe char designed as 2 bytes.

If java application implemented i18n concept , it supports all regional languages in the world

i18n = Internationalization

Program to display size and Range of Values.

```
public class Test{
	public static void main(String [] args){
		System.out.println ("character size :"+Character.SIZE+"\tMin Value : "+Character.MIN_VALUE+"\tMax Value : "+Character.MAX_VALUE);
		System.out.println ("byte size :"+Byte.SIZE+"\tMin Value : "+Byte.MIN_VALUE+"\tMax Value : "+Byte.MAX_VALUE);
		System.out.println ("short size :"+Short.SIZE+"\tMin Value : "+Short.MIN_VALUE+"\tMax Value : "+Short.MAX_VALUE);
		System.out.println ("int size :"+Integer.SIZE+"\tMin Value : "+Integer.MIN_VALUE+"\tMax Value : "+Integer.MAX_VALUE);
		System.out.println ("long size :"+Long.SIZE+"\tMin Value : "+Long.MIN_VALUE+"\tMax Value : "+Long.MAX_VALUE);
		System.out.println ("float size :"+Float.SIZE+"\tMin Value : "+Float.MIN_VALUE+"\tMax Value : "+Float.MAX_VALUE);
		System.out.println ("double size :"+Double.SIZE+"\tMin Value : "+Double.MIN_VALUE+"\tMax Value : "+Double.MAX_VALUE);
	}
}
```

O/P:

Variable :

What is variable ?

it is a name given some location in the program memory it's value changes at runtime. it is a temporary memory , it clears automatically once program stopped

variable naming :

==========;

how to choose name of the variable

1) Naming Conditions
2) Naming Conventions

1) Naming Conditions

These are as per the compiler.

1) It contains only alphabets (a-z, A-Z), digits (0 – 9) and under score (_)
2) Should not be used spaces and special chars, keywords.
3) Max length can be up to 255 chars

Naming conditions are same for all identifiers

identifiers:

=======

class name, interface name, variable name, object name, method name

2) Naming Conventions

These are different for different identifiers Variable name conventions

1) variable name should start with lower case alphabet
2) if variable contains multiple words , then 1st word start with lowercase , and next word start with upper case alphabet

Eg: empDeptName;
empSalary;

Declaration of a variable.

data_type variable_name;

eg :

```
int empNum;
String empName;
double empSalary;
String empDeptName;
String empGender;
int empAge
```

Assigning values to variables:

```
empNum = 1001;
empName = "mnrao";
empSalary = 50505.50;
```

```
        empDeptName ="it";
        empGender = "male";
        empAge = 35;
```

Memory Management by JVM

JVM will split the memory into different parts as below

Higher order memory space :

To store all command line parameters and OS environment variable.

Stack Memory space :

To store all local variables. Memory allocation is LIFO (last In First Out) based. This memory space grows automatically as long as sharable memory available for it If stack filled it raises stack over flow exception.

Shared Memory space:

This memory can be shared by both stack and heap.

Heap Memory space:

This memory space is for dynamic memory management. In this area, memory allocation and de-allocating is always at runtime. In heap memory space, memory allocation is randomly This memory space grows automatically as long as sharable memory available for it.

Global Memory space:

This memory space is to store global variables. as per the no of global variables declaration in Java-Application.

Static Memory space:

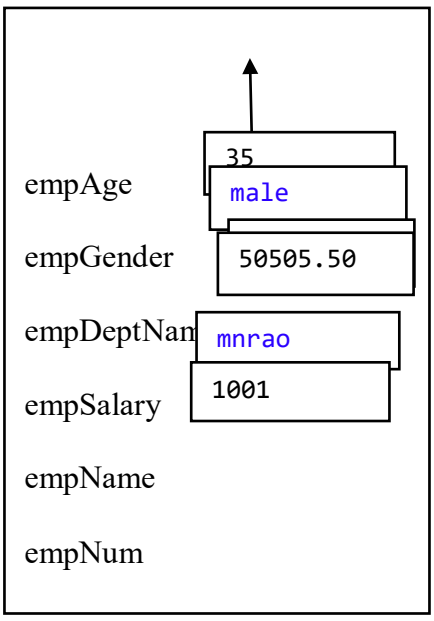
This memory is to store the static variables. It's size is a fixed one as per the no of static variables declaration in Java - Application.

Text Memory Space:

This memory contains all Java-Program instructions, Java-Libraries Stack Memory for the below Program

```
public class Test {
        public static void main(String[] args) {
                // declaring variables
                int empNum;
                String empName;
                double empSalary;
                String empDeptName;
                String empGender;
                int empAge;

                // assigning values to variables
                empNum = 1001;
                empName = "mnrao";
                empSalary = 50505.50;
                empDeptName ="it";
                empGender = "male";
                empAge = 35;
                System.out.println(empNum);
                System.out.println(empName);
                System.out.println(empSalary);
                System.out.println(empDeptName);
                System.out.println(empGender);
                System.out.println(empAge);
                }  }
```

Stack



Once control come out of main method, stack becomes empty.

Initialization of variables:

Assigning values at the time of declaration

```
int empNum = 1001;
String empName = "mnrao";
double empSalary = 50505.50;
String empDeptName = "admin";
String empGender = "male";
int empAge = 35 ;
```

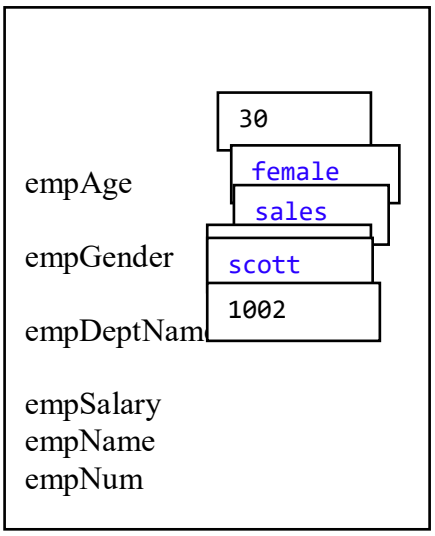
program for initialization

```
public class Test {
	public static void main(String[] args) {
		int empNum = 1001;
		String empName = "mnrao";
		double empSalary = 50505.50;
		String empDeptName = "admin";
		String empGender = "male";
		int empAge = 35 ;
		System.out.println(empNum);
		System.out.println(empName);
		System.out.println(empSalary);
		System.out.println(empDeptName);
		System.out.println(empGender);
		System.out.println(empAge);
	} }
```

Changing value of varaibles :

```
package com.durga.mnrao.abc;

public class Test {
	public static void main(String[] args) {
		int empNum = 1001;
		String empName = "mnrao";
		double empSalary = 50505.50;
		String empDeptName = "admin";
		String empGender = "male";
		int empAge = 35 ;
		System.out.println("initial values are ");
		System.out.println(empNum);
		System.out.println(empName);
		System.out.println(empSalary);
		System.out.println(empDeptName);
		System.out.println(empGender);
		System.out.println(empAge);
		// changing values
		empNum = 1002;
		empName = "scott";
		empSalary = 60607.60;
		empDeptName = "sales";
		empGender = "female";
		empAge = 30;
		System.out.println("after changing value are ");
		System.out.println(empNum);
		System.out.println(empName);
		System.out.println(empSalary);
		System.out.println(empDeptName);
		System.out.println(empGender);
		System.out.println(empAge);
	} }
```

Stack



Same location value overwrites .

initial values are

```
1001   mnrao  50505.5 admin  male 35
after changing value are
1002  scott  60607.6 sales female  30
```

Mutiple Variables displaying in a single statement :

+ is used for concatenation of text and variables.

+ ? concatenation operator

Concatenation means joining .

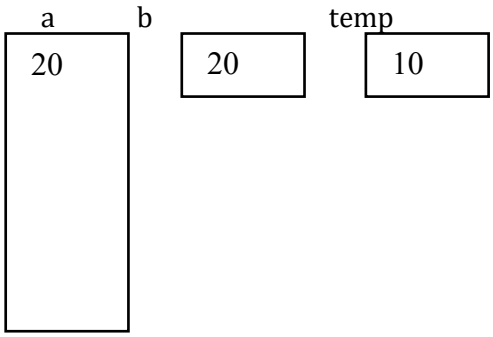
```
package com.durga.mnrao.test;
public class Test {
        public static void main(String[] args) {
                                // initialization of variables
                int empNum = 1001;
                String empName = "mnrao";
                double empSalary = 50505.50;
                String empDeptName = "admin";
                String empGender = "male";
                int empAge = 35 ;
                System.out.println("initial values are ");
        System.out.println("emp num = " + empNum + "\t name = " + empName + "\t salary = " + empSalary + "\t dept = " +
empDeptName + "\t gender = " + empGender + "\t age = " + empAge);
                // changing values , assignment .
                empNum = 1002;
                empName = "scott";
                empSalary = 60607.60;
                empDeptName = "sales";
                empGender = "female";
                empAge = 30;
                System.out.println("after changing value are ");
                System.out.println("emp num = " + empNum + "\t name = " + empName + "\t salary = " + empSalary + "\t dept
= " + empDeptName + "\t gender = " + empGender + "\t age = " + empAge);
        }  }
```

Dynamic initialization of a variable:

```
int a =10;   // compile time initialization
int b =20; // compile time initialization
// below is the dynamic initialization
int c = a*b;
```

Programm to swap the two variables:

```
public class Test {
        public static void main(String[] args) {
                int a = 10;
                int b = 20;
                System.out.println ("Before swapping");
                System.out.println ("a=" + a + "\tb=" + b);
                int temp = a;
                a = b;
                b = temp;
                System.out.println ("After swapping");
                System.out.println ("a=" + a + "\tb=" + b);
        }  }
```




```
Before swapping
a=10    b=20
After swapping
a=20    b=10
==============
```

Reading data from keyboard

There are many ways to read data from the keyboard. For example:

- Input-Stream-Reader
- Console

- Scanner
- Data-Input-Stream etc.
- Buffered-Reader

At this stage, we can focus only on **Scanner** and others will be discussed in **I/O streams.**

Java Scanner class:

it is a class from java.util package:

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default.
It provides many methods to read and parse various primitive values.

Methods of Scanner class

Method	Description
public String next()	It returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte (max 127)
public short nextShort()	it scans the next token as a short value (max 32767)
public int nextInt()	it scans the next token as an int value (upto 8 digits)
public long nextLong()	it scans the next token as a long value. (upto 12 digits)
public float nextFloat()	it scans the next token as a float value. (small real number)
public double nextDouble()	it scans the next token as a double value. (big real number)
public boolean nextBoolean()	Reading true / false

All methods will read data upto next token (space or tab or \n, which is coming first), except, nextLine()
nextLine() : it will read input data upto eneter key including space and tab. It will read any type of data.
next() : It will read input data up to space or tab or new line char, which is coming first. It will read any type of data.

All methods, except nextLine() and next(), can read only numeric data.

Program to read different type of data from keyboard

```
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
        Scanner sc = new Scanner( System.in );
                System.out.println("Enter byte value ");
                byte b = sc.nextByte();
                System.out.println("byte variable b = "+b);
                System.out.println("Enter short value ");
                short s = sc.nextShort();
                System.out.println("Short variable s = "+s);
                System.out.println("Enter int value ");
                int i = sc.nextInt();
                System.out.println("int varible i = "+i);
                System.out.println("enter long value ");
                long l = sc.nextLong();
                System.out.println("long variable l = "+l);
                System.out.println("enter real number ");
                float f = sc.nextFloat();
                System.out.println(" float variable f = "+f);
                System.out.println("enter big real number ");
                double d = sc.nextDouble();
                System.out.println("double variable d = "+d);
                System.out.println("Enter a String ");
                String str = sc.next();
                System.out.println("String value = "+str);
                System.out.println("Enter boolean value ");
                boolean flag = sc.nextBoolean();
                System.out.println("boolean variable flag = "+flag);
                sc.close();
        } }
```

Program to read emp information

```
import java.util.Scanner;
```

```
public class Test {
	public static void main(String[] args) {
		Scanner  sc = new Scanner(System.in);
		System.out.println("Enter EMP NO : ");
		int eno = sc.nextInt();
		System.out.println("Enter EMP Name : ");
		String name = sc.next();
		System.out.println("Enter Salary ");
		double salary = sc.nextDouble();
		System.out.println("Enter Dept : ");
		String dept = sc.next();
	System.out.println("EMP NO "+eno+"\tEMP NAME "+name+"\tEMP SALARY "+salary+"DEPT "+dept);	sc.close();

	}  }
```

```
Enter EMP NO :1001   Enter EMP Name :NRIT
Enter Salary :-5000    Enter Dept : dev
EMP NO 1001   EMP NAME NRIT        EMP SALARY 5000.0DEPT dev
```

Type Casting:

It is a process of converting one data type to another data type
These are of two types
1) premitive data types casting (basic to basic)
2) Non-Premitive casting (user defined classes)

primitive data types casting

these are

1) Implicit Casting
2) Explicit casting

1) Implicit Casting

Implicit means internally (automatically) Implicit casting is available for the below

1) Small size to big size
2) Integer to real number

Small size to bigsize :

eg1:

```
byte b=10;
short s = b ; --> valid
```

eg2:

```
int a=10;
long b=a; --> valid
```

eg3:

```
float a=10.5f;
double b=a; --> valid
```

Integer to real number :

Eg:

```
int i = 100;
float f = i ;
```

eg:

```
int a = 1000;
double d = a;
```

data and type promotion as below.

byte ===> short ===> int ===> long ==>float ===> double
char ==> int ==>long==>float ===> double
converting from real number to integer, implicit casting is not applicable.
To convert real to integer type explicit casting required.

2) Explicit casting:

Java developer has to convert explicitly .

it is required for below one

1) big type to Small type
2) real number type to Integer type

1) big type to Small type

```
short s = 10;
```

```
        byte b = (byte) s;
eg2:
        int i = 100;
        short s = (short )i;
eg3:
        double d = 10.5;
        float f = (float) d;
```

Eg:

short a=10;
byte b = a; --> invalid, since size of variable " a" is 2 bytes , which is more than size of variable "b" (1 byte)
such of kind of scenarios, we use explicit type casting.
short a=10;
byte b = (byte) a ; --> it is valid
same scenario with following data
short a=130;
byte b = (byte) a ; --> it compiles but at run time loss of data. It results in unexpected data (i.e leads to bugs)
here value of b is -126 .
hence explicit type casting is recommended is only for **constants**, not for variables.

2) real number type to Integer type

double d = 10.5;
int i = (int) d;
long l = (long) d ;
float f= 10.5 ; --> invalid, since in java real numbers by default treats as double.
float f = (float) 10.5; (or)
float f = 10.5f;
default data type.
Real numbers are treated as **double** type.

Result of mathematical expressions, with different data types.

1) byte + byte = int
2) short + short = int
3) int + int = int (compiles) but at runtime there is chance of data over flow. Recommended one is (int+ int = long)
4) long + long = long
5) char + char = int.
6) float + float = double
7) double + double = double
8) int + long = long
9) float + double = double
10) int + float = float

Division (/) operation:

```
int a =10;

float  x = a/3 ;
```

result value is 3.0
reason in division, both operands are int type then result int.
to get real number, at leat one operand must be float / double

```
int a =10;
float  x = a / 3f ;
```

result is 3.333333
eg:

```
int a =10;
int b = 3;
float  x = a / b ;
```

result value is 3.0
reason in division, both operands are int type then result int.
solution for the above is, type casting .

```
float  x = (float) a / b ;(or)
float  x =  a / (float) b ;
```

Java Operators

Based on no of operands, Operators are three categories

1) Unary Operators Single Operand
2) Binary Operators Two Operands
3) Ternary Operators Three Operands

Binary Operators:

1) Arithmetic / Mathematical Operators 2) Relational Operators
3) Logical Operators 4) Assignment Operator
5) short hand assignment Operator 6) Bitwise operators

1)Arithmetic / Mathematical Operators

Operator	Task
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modula's (remainder)

Eg:

10 + 3 =13 10-3 = 7
10*3 =30 10/5=2
10%3=7

2)Relational Operators:

Operator	Task
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equals to
==	Is equal to
!=	Not equal to

== --> comparison
= --> assignment.

Relational operators returns either **true or false (Boolean values)**

Eg:

```
int a = 10;
int b = 5;
boolean res = a>b;
```

here res value is true

3)Logical Operators:

Operator	Task
& &	And
| |	Or
!	Not

Using (&&) operator:

Condition1	Operator	Condition2	Result
TRUE	**&&**	TRUE	TRUE
TRUE	**&&**	FALSE	FALSE
FALSE	**&&**	TRUE	FALSE
FALSE	**&&**	FALSE	FALSE

Using (\| \|) operator:

Condition1	Operator	Condition2	Result
TRUE	| |	TRUE	TRUE
TRUE	| |	FALSE	TRUE
FALSE	| |	TRUE	TRUE
FALSE	| |	FALSE	FALSE

Using not (!):

! TRUE --> FALSE
! FALSE --> TRUE

program to add, subtract and multiply two numbers.

```
import java.util.Scanner;
```

```
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter 1st number ");
                int a = sc.nextInt();
                System.out.println("enter 2nd number ");
                int b = sc.nextInt();
                int c = a + b;
                System.out.println("addition = "+c);
                c = a - b;
                System.out.println("subtraction = "+c);
                c = a * b;
                System.out.println("Multiplication = "+c);
        }  }
```

Unary Operator:

It works with only one operand.

1) Unary Minus
2) Increment Operator (++)
3) Decrement Operator (--)

Unary Minus:

Eg:

```
int a=10;
int b =  - a;  ( unary minus )
```

here value of b is -10, value of **a** remains same (10) no change

```
int  c =  a – b ; ( binary minus )
```

Increment and decrement operators :

Increment Operator (++) :

It increases value of it's operand by 1.

Decrement operator (- -):

It decreases value of it's operand by 1.

Increment Operator (++) :

Pre – increment	Post – increment
1) syntax : ++ variable;	1) syntax : variable++;
2)It increase value of it's Operand by 1 before executing statement	2)It increase value of it's Operand by 1 after executing statement
3) eg : int a=10; int b = ++a: --> a=11, b=11	3) eg : int a=10; int b = a++: -->b=10, a=11

```
public class Test{
        public static void main(String[] args) {
                int a=10;
                int b;
                int c;
                int d;
                b=++a;
                c=b++;
                d=++c;
                System.out.println (a+"\t"+b+"\t"+c+"\t"+d);
        }  }
```

O/p : 11 12 12 12

```
public class Test{
        public static void main(String[] args){
                int a=10;
                int b;
                int c;
                int d;
                b=a++;
```

```
                c=++b;
                d=c++;
                b=d++;
                a=++c;
                System.out.println (a+"\t"+b+"\t"+c+"\t"+d);
        } }
```

O/P : 13 11 13 12

```
public class Test {
        public static void main(String[] args){
                int a=10;
                int b;
                int c;
                int d;
                d=++a;
                b=d++;
                a=++b;
                c=d++;
                d=++c;
                System.out.println (a+"\t"+b+"\t"+c+"\t"+d);
        }   }
```

O/P :

12 12 13 13

Decrement Operator (- -) :

Pre – Decrement	Post – Decrement
1) syntax : -- variable;	1) syntax : variable--;
2)It decreases value of it's Operand by 1 before executing statement	2)It decreases value of it's Operand by 1 after executing statement
3) eg : int a=10; int b = --a: --> a=9, b=9	3) eg : int a=10; int b = a--: -->b=10, a=9

```
public class Test{
        public static void main(String[] args){
                int a=10;
                int b;
                int c;
                int d;
                b=--a;
                c=b++;
                d=--c;
                a=++b;
                d=a--;
                System.out.println (a+"\t"+b+"\t"+c+"\t"+d);
        }}
```

O/P :

10 11 8 11

```
public class Test{
        public static void main(String[] args){
                int a = 10;
                int b;
                int c;
                int d;
                d = a--;
                b = ++d;
                c = a++;
                a = --b;
                d = c++;
```

```
        System.out.println (a + "\t" + b + "\t" + c + "\t" + d);
    } }
```

O/p :

10 10 10 9

Java Control Statements

1) if statement
2) switch-case statement
3) while loop
4) do-while loop
5) for loop
6)for-each loop

if – statement:

these are of four types

1) simple-if
2) if-else
3) Nested – if
4) ladder -if

1) simple-if:

```
if < condition >
{
    ==============
    ==============
    true part;
   ================
    ================
}
```

Higher Order Memory
Stack Memory
Shared Memory
Heap Memory
Static Memory
Global Memory
Text Area
Method Area

2) if-else

```
if < condition >
{
    ======================
    ======================
    true part;
   ========================
    ========================
}
else
{
    ======================
    ======================
    false part;
   ========================
    ========================
}
```

3) Nested – if :

```
if < condition1 >
{
    ================
    ================
        if < condition2 >
        {
        ================
        ================
                if < condition3 >
                {
                ================
                ================
                }
                else
                {
                ================
                ================
                }
        ================
        ================
        }
        else
        {
        ================
        ================
        }
================
================
}
else
{
    ================
    ================
}
```

5) Ladder if:

```
if <condition1>
{
_____________
_____________
}
else if <condition2>
{
_____________
_____________
}
else if <condition3>
{
_____________
_____________
}
else
{
_____________
_____________
}
```

Ladder if is a reverse of nested if.
Nested is used for positive approach.
Ladder if is used for negative approach.

if statement works based on condition return values (true / false)

eg:

```
if( true ){
        System.out.println ("true part");
}
else{
        System.out.println ("false part");
}
```

program to use condition return value:

```
public class Test {
    public static void main(String []  args){
        int a=20;
        int b=30;
        boolean c;
        c=a>b;
        if( c ){
            System.out.println ("true part");
        }
        else{
            System.out.println ("else part");
        }
        c=a<b;
        if( c ){
            System.out.println ("true part");
        }
        else{
            System.out.println ("else part");
        }
        c=(a<=b);
        if( c ){
            System.out.println ("true part");
        }
        else{
            System.out.println ("else part");
        } } }
```

o/p:

else part
true part
true part

Greater of two numbers:

```
package com.durga.mnrao.test;
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
    Scanner  sc = new Scanner(System.in);
        System.out.println("Enter a 1st number ");
        int a = sc.nextInt();
        System.out.println("Enter a 2nd number ");
        int b = sc.nextInt();
```

```
        if( a>b ){
                System.out.println("greater number a = "+a);
        }
        else{
                System.out.println("greater number b = "+b);
        } } }
```

program to check for equal and greater of two numbers

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner  sc = new Scanner(System.in);
                System.out.println("Enter a 1st number ");
                int a = sc.nextInt();
                System.out.println("Enter a 2nd number ");
                int b = sc.nextInt();
                if( a==b ){
                        System.out.println("both are equal");
                }
                else if( a>b ){
                        System.out.println("greater number a = "+a);
                }
                else
                {
                        System.out.println("greater number b = "+b);
                }  } }
```

program to find greater of three numbers:

```
public class Test {
        public static void main(String []  args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter 1st number ");
                int a = sc.nextInt();
                System.out.println("Enter 2nd number ");
                int b = sc.nextInt();
                System.out.println("Enter 3rd number ");
                int c = sc.nextInt();
                if (a > b){
                        if(a>c){
                                System.out.println (" a is greater");
                        }
                        else{
                                System.out.println (" c is greater");
                        }
                }
                else if (b>c){
                        System.out.println (" b is greater");
                }
                else
                {
                        System.out.println (" c is greater");
                } } }
```

Program to check the result of student based on subjects marks:
if all subjects marks are greater than or equal to 40 then pass otherwise fail.

Using Nested - if

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter sub1 Marks ");
```

```
		int m1 = sc.nextInt();
		System.out.println("enter sub2 Marks ");
		int m2 = sc.nextInt();
		System.out.println("enter sub3 Marks ");
		int m3 = sc.nextInt();
		if(  m1>=40 ){
			if(m2>=40){
				if(m3>=40){
					System.out.println("Pass");
				}
				else{
					System.out.println("Fail");
				}
			}
			else{
				System.out.println("Fail");
			}
		}
		else{
			System.out.println("Fail");
		} } }
```

Using and (&&) operator :

```
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("enter sub1 Marks ");
		int m1 = sc.nextInt();
		System.out.println("enter sub2 Marks ");
		int m2 = sc.nextInt();
		System.out.println("enter sub3 Marks ");
		int m3 = sc.nextInt();
		if(  m1>=40 && m2>=40 && m3>=40 ){
			System.out.println("Pass");
		}
		else{
			System.out.println("Fail");
		} } }
```

Working of (&&) Operator :

```
Cond1   &&      Cond2   &&      Cond3   &&      Cond4   &&      Cond5  Result

T               T               T               T               T  -->T

T               T               T               T               F -->F

T               T               T               F --> F

T               T               F -->

T               T               F -->F

F --> F
```

Another way: Using ladder-if

```
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("enter sub1 Marks ");
		int m1 = sc.nextInt();
		System.out.println("enter sub2 Marks ");
		int m2 = sc.nextInt();
```

```
                System.out.println("enter sub3 Marks ");
                int m3 = sc.nextInt();
                if( m1<40 ){
                        System.out.println("fail");
                }
                else if( m2<40 ){
                        System.out.println("fail");
                }
                else if( m3<40 ){
                        System.out.println("fail");
                }
                else{
                        System.out.println("Pass");
                } } }
```

Another way:

Using or (||) operator:

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter sub1 Marks ");
                int m1 = sc.nextInt();
                System.out.println("enter sub2 Marks ");
                int m2 = sc.nextInt();
                System.out.println("enter sub3 Marks ");
                int m3 = sc.nextInt();
                if( m1<40 || m2<40 || m3<40 ){
                        System.out.println("fail");
                }
                else{
                        System.out.println("Pass");
                } } }
```

Working of (| |) Operator :

Cond1	||	Cond2	||	Cond3	||	Cond4	||	Cond5	Result
F			F			F			F			F -->F
F			F			F			F			T -->T
F			F			F			T --> T			
F			F			T -->T						
F			T -->T									
T -->T												

Grade of the Students:

Grade of the students based average Marks

Student should be passed, then grades

avg>=90 --> Grade A

80 to 89 --> Grade B

70 to 79 --> Grade C

60 to 69 --> Grade D

<60 --> Grade E (40 to 59)

if failed no grade

```
package com.durga.mnrao.test;
import java.util.Scanner;
public class Test {
        public static void main(String [] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter sub1 Marks");
```

```
        int m1 = sc.nextInt();
        System.out.println("Enter sub2 Marks");
        int m2 = sc.nextInt();
        System.out.println("Enter sub3 Marks");
        int m3 = sc.nextInt();
        if(  m1>=40 && m2>=40 && m3>=40 ){
                int avg = ( m1 + m2 + m3 ) / 3;
                if( avg>=90 ){
                        System.out.println("Passed and Grade A");
                }
                else if( avg >= 80 ){
                        System.out.println("Passed and Grade B");
                }
                else if( avg >= 70  ){
                        System.out.println("Passed and Grade C");
                }
                else if( avg >= 60  ){
                        System.out.println("Passed and Grade D");
                }
                else {
                        System.out.println("Passed and Grade E");
                }
        }
        else{
                System.out.println("Failed and No Grade");
        } } }
```

Program to check subject wise Result and Final Result of a student

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args)  {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter sub1 marks ");
                int m1 = sc.nextInt();
                System.out.println("Enter sub2 marks ");
                int m2 = sc.nextInt();
                System.out.println("Enter sub3 marks ");
                int m3 = sc.nextInt();
                int passCount=0;
                if(m1>=40){
                        passCount++;
                        System.out.println("Sub1 Passed");
                }
                else{
                        System.out.println("Sub1 Failed");
                }
                if(m2>=40){
                        passCount++;
                        System.out.println("Sub2 Passed");
                }
                else{
                        System.out.println("Sub2 Failed");
                }

                if(m3>=40){
                        passCount++;
                        System.out.println("Sub3 Passed");
                }
                else{
                        System.out.println("Sub3 Failed");
```

```
        }
        if(passCount>=3){
                System.out.println("Student Passed");
        }
        else{
                System.out.println("Student Failed");
        } } }
```

There are four subjects , at least three subjects passed then student is passed

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args)  {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter sub1 marks ");
                int m1 = sc.nextInt();
                System.out.println("Enter sub2 marks ");
                int m2 = sc.nextInt();
                System.out.println("Enter sub3 marks ");
                int m3 = sc.nextInt();
                System.out.println("Enter sub4 marks ");
                int m4 = sc.nextInt();
                int passCount = 0;
                if( m1 >= 40 ){
                        passCount++;
                }
                if( m2 >= 40 ){
                        passCount++;
                }
                if( m3 >= 40 ){
                        passCount++;
                }
                if( m4 >= 40 ){
                        passCount++;
                }
                if( passCount>=3 ){
                        System.out.println("Student Passed");
                }
                else{
                        System.out.println("Student Failed");
                } } }
```

Both Subjectwise result as well final result of the student, for the above requirement

```
import java.util.Scanner;
public  class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Eneter Sub1 Marks ");
                int m1 = sc.nextInt();
                System.out.println("Eneter Sub2 Marks ");
                int m2 = sc.nextInt();
                System.out.println("Eneter Sub3 Marks ");
                int m3 = sc.nextInt();
                System.out.println("Eneter Sub4 Marks ");
                int m4 = sc.nextInt();
                int passCount = 0;
                if( m1>=40 ){
                        passCount++;
                        System.out.println("Sub1 Passed ");
                }
                else{
                        System.out.println("Sub1 Failed ");
```

```
        }
        if( m2>=40 ){
                passCount++;
                System.out.println("Sub2 Passed ");
        }
        else{
                System.out.println("Sub2 Failed ");
        }
        if( m3>=40 ){
                passCount++;
                System.out.println("Sub3 Passed ");
        }
        else{
                System.out.println("Sub3 Failed ");
        }
        if( m4>=40 ){
                passCount++;
                System.out.println("Sub4 Passed ");
        }
        else{
                System.out.println("Sub4 Failed ");
        }
        if( passCount >= 3 ){
        System.out.println("Student Passed");
        }
        else{
                System.out.println("Student Failed");
        } } }
```

Program to display Message on Cricket bat's man score

```
        score<0  --> Invalid input
        score = 0  --> Duckout
        score ==> 1 to 49  ==> Normal Score
        score ==> 50 to 99  ==> Half Century
        score ==> 100 to 199 ==> Century
        score  >=200  ==> Double Century
import java.util.Scanner;
 public class Test {
        public static void main(String[] args)  {
        Scanner  sc = new Scanner(System.in);
        System.out.println("Enter the score ");
        int score =sc.nextInt();
        if(score<0)                {
        System.out.println("Invalid Input");
        }
        else if(score==0)                {
                System.out.println("Duck out");
        }
        else if(score<50)                {
                System.out.println("Normal score");
        }
        else if(score<100)                {
                System.out.println("Half Century");
        }
        else if(score<200)                {
                System.out.println("Century");
        }
        else                {
                System.out.println("Double Century");
        } } }
```

Ternary Operator / Conditional Operator(?) :

It works with three operands

Variable = <condition> **?** **expr1** : **expr2** ;
true false

if condition is true , Expr1 executes and result return to Variable.
if condition is false , Expr2 executes and result return to Variable.

Eg1:

```
int a=10;
int b=15;
int  c = ( a > b ) ?  a-b : a+b;
```

value of c is 25

Eg2:

```
int a=20;
int b=15;
int  c = ( a > b ) ?  a-b : a+b;
```

value of c is 5

Gretare of two numbers :

```
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("enter 1st number ");
		int a = sc.nextInt();
		System.out.println("Enter 2nd number ");
		int b = sc.nextInt();
		int big = a>b ? a : b;
		System.out.println("Greater number = "+big);
	} }
```

Greater of three numbers:

```
package com.durga.mnrao.abc;
import java.util.Scanner;
public class Test {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("enter 1st number ");
		int a = sc.nextInt();
		System.out.println("Enter 2nd number ");
		int b = sc.nextInt();
		System.out.println("Enter 3rd number ");
		int c = sc.nextInt();
		int big = a>b ? ( a>c ? a : c ) : (b>c ? b : c );
		System.out.println("Greater number = " + big);		}}
```

Switch-case:

```
switch (value)
{
	case constant1:
		=========================;
		=========================;
		break;
	case constant2:
		=========================;
		=========================;
		break;
	case constant3:
		=========================;
		=========================;
		break;
	case constant4:
		=========================;
		=========================;
		break;
	default:
		=========================;
		=========================;
		break;
}
```

if **value** matched with any case **constant**, that case statements block executes. If **value** is **not matching** with any of the case constants, then **default** case executes. **break** statement will take the control out of the switch case.

Use of **break** statement is an optional in default case (or) last case block If **break** is not presented, in any of the case blocks,

then next case block also executes till occurance of next **break**. Default case block, can be placed any where in the switch-case , at the end or beginning, or at middle also. **Recommended one is at the end.** Case Constant can be a **integer** or **char** or **String** but should **not be a** real number

Eg:

10, 20 , 30 , 40 , 50 valid
'a', 'b' , 'c', 'd' valid
"ONE" , "TWO", "THREE" valid
10.5, 20.1, 1.1 invalid

```
public class Test {
	public static void main(String[] args) {
	int a=10;
	System.out.println ("before switch case");
		switch (a)
		{
		case 10:
			System.out.println ("TEN");
			break;

		case 20:
			System.out.println ("Twenty");
			break;
		case 30:
			System.out.println ("Thirty");
			break;
		case 40:
			System.out.println ("Fourty");
			break;
		default:
			System.out.println ("DEFAULT");
			break;
		}
		System.out.println ("after switch case");
	} }
```

```
package com.nrit.mnrao.test;
public class Test {
	public static void main(String[] args) {
		char ch='a';
		switch ( ch ){
		case 'a':
				System.out.println("ONE");
				break;
		case 'b':
				System.out.println("Two");
				break;
		case 'c':
				System.out.println("Three");
				break;
		case 'd':
				System.out.println("Four");
				break;
		default:
				System.out.println("default");
				break;
		}
		System.out.println("After switch case");
	}  }
```

```
package com.nrit.mnrao.test;
public class Test {
	public static void main(String[] args) {
```

```
		String str ="ONE" ;
		switch ( str ){
		case "ONE":
				System.out.println("ONE");
				break;
		case "TWO":
				System.out.println("Two");
				break;
		case "THREE":
				System.out.println("Three");
				break;
		case "FOURE":
				System.out.println("Four");
				break;
		default:
				System.out.println("default");
				break;
		}
		System.out.println("After switch case");
	} }
```

Common statements for multiple cases :

```
import java.util.Scanner;
public  class Test {
	public static void main(String[] args) {
	Scanner sc = new Scanner(System.in);
		System.out.println("Enter your choice 1 - 7 ");
		int choice = sc.nextInt();
		switch( choice ){
			case 1:
			case 2:
			case 3:
			case 4:
			case 5:
					System.out.println("Working Day");
					break;
			case 6:
			case 7:
					System.out.println("Holiday");
					break;

			default:
					System.out.println("not in weekdays");
					break;
		} } }
```

Menu based application

```
import java.util.Scanner;
public class Test {
	public static void main(String[] args)  {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter first number ");
		int a=sc.nextInt();
		System.out.println("Enter Second number ");
		int b=sc.nextInt();
		System.out.println("Menu");
		System.out.println("1.Addition ");
		System.out.println("2.Subtraction ");
		System.out.println("3.Multiplication ");
		System.out.println("Enter your choice 1/2/3 : ");
		int choice = sc.nextInt();
```

```
        switch( choice ){
                case 1:
                                System.out.println("Addition = "+(a+b));
                                break;
                case 2:
                                System.out.println("Subtraction = "+(a-b));
                                break;
                case 3:
                                System.out.println("Multilication = "+(a*b));
                                break;
                default:
                        System.out.println("Wrong choice");
                        break;
        } } }
```

Loops:

1) while loop
2) do – while loop
3) for – loop

loops are used to execute same statements again and again (iterative statements)
iterative --> repeatedly

while loop:

Syntax:

```
        while( Cond ){
                =========;
                ========;
        }
```

To display 1 to 10

```
public class Test{
        public static void main(String[] args) {
                int i=1;
                while(i<=10){
                System.out.println (i);
                        i++;
                } } }
```

To display 10 to 1

```
public static void main(String[] args) {
                int i=10;
                while(i>=1){
                        System.out.println (i);
                        i--;
                } } }
```

To display 1 to n

```
public class Test{
        public static void main(String[] args){
          int i=1;
          int n=20;
          while(i<=n){
                    System.out.println (i);
                     i++;
          } } }
```

do-while:

syntax:

```
        do{
                =======;
                =======;
        }
```

while(cond) ; --> here semicolon required.

do-while loop executes at least once.

```
public class Test{
```

```
        public static void main(String[] args){
                int i=1;
                do{
                        System.out.println (i);
                        i++;
                }
                while(i<=10);
        }  }
```

for – loop:

syntax:

```
for (initialization  ;  condition   ;   increment / decrement)
{
                =======
                =======
}
public class Test{
        public static void main(String[] args) {
                for(int i=1;i<=10;       i++)
                {
                        System.out.println (i);
                } } }
```

Even Numbers:

```
public class Test{
        public static void main(String[] args){
        int i=1;
                while(i<=20){
                        if(i%2==0){
                                System.out.println (i);
                        }
                        i++;
                }       }  }
```

Odd Numbers :

```
public class Test{
        public static void main(String[] args){
                int i=1;
                while(i<=20){
                        if(i%2!=0){
                                System.out.println (i);
                        }
                        i++;
                } } }
```

Even Numbers using for loop:

```
public class Test{
        public static void main(String[] args){
                for( int i=1; i<=20; i++){
                        if(i%2==0){
                                System.out.println (i);
                        } } } }
```

program to find sum of the numbers from 1 to 10.

```
public class Test{
        public static void main(String[] args){
                int i=1;
                int sum=0;
                while(i<=10){
                        sum=sum+i;
                        i++;
                }
                System.out.println (sum);
        } }
```

using for loop.

```
public class Test{
	public static void main(String[] args){
		int sum=0;
		for( int i=1;i<=10;i++){
			sum=sum+i;
		}
		System.out.println ("sum value:"+sum);
	} }
```

program to find sum of even and odd numbers from 1 to 10.

```
public class Test{
	public static void main(String[] args){
		int i=1;
		int esum=0;
		int osum=0;
		while( i <=10){
			if(i%2==0){
				esum=esum+i;
			}
			else{
				osum=osum+i;
			}
			i++;
		}
		System.out.println ("Even sum = "+esum);
		System.out.println ("Odd sum = "+osum);
	} }
```

using for loop:

```
public class Test{
	public static void main(String[] args){
		int esum=0;
		int osum=0;
		for( int i=1; i<=10; i++){
			if(i%2==0){
				esum=esum+i;
			}
			else{
				osum=osum+i;
			}  }
		System.out.println ("even sum value:"+esum);
		System.out.println ("odd sum value:"+osum);
	} }
```

Program to print following series

1 2 3 5 5 8 7 11 9 14 11 17 13 20

```
public class Test{
	public static void main(String[] args){
		int i;
		int j;
		int n = 20;
		for (  i = 1, j=2;  i <= n && j <= n;    ){
			System.out.print(i + "\t" + j + "\t");
			i = i + 2;
			j = j + 3;
		} }
```

Program to find factorial of given number .

```
import java.util.Scanner;
public class Test {
	public static void main( String [] args) {
		Scanner scanner = new Scanner(System.in);
```

```
                System.out.println("Enter the number ");
                 int n = scanner.nextInt();
                int fact = 1;
               while( n>=1 )
               {
                        fact = fact * n;
                        n--;
               }
               System.out.println("Factorial  "+fact);
  } } }
```

Program to find sum of digits of a given number

```
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the number ");
                int n = scanner.nextInt();
               int sum = 0;
               while( n>0 )
               {
                        int r = n%10;
                        sum  = sum + r;
                        n = n /10;
               }
               System.out.println("sum =   "+sum);
} } }
```

Program to make the reverse of a given number

```
package com.durga.mnrao.bank;
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
                        Scanner sc = new Scanner(System.in);
                        System.out.println("Enter a number ");
                        int n = sc.nextInt();
                        int rev = 0;
                        while( n>0 )
                        {
                                int r = n%10;
                                rev = rev * 10 + r;
                                n = n / 10;
                        }
                        System.out.println("reverse = "+rev);
} }
```

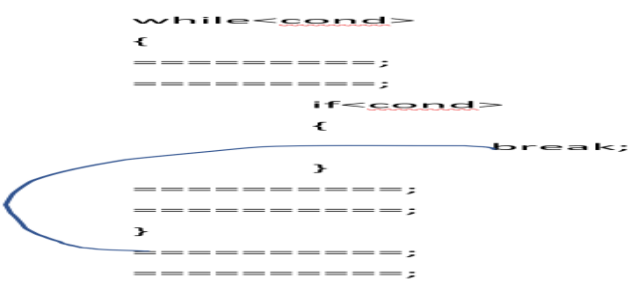
Program to check given number is palindrome or not

```
package com.durga.mnrao.bank;
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number ");
                        int n = sc.nextInt();
                        int temp = n;
                        int rev = 0;
                        while(n>0 ){
                                int r = n%10;
                                rev = rev * 10 + r;
                                n = n / 10;
                        }
                        System.out.println("reverse = "+rev);
```

```
                System.out.println(" n = "+n);
                System.out.println("temp = "+temp);
                if( rev == temp )                       {
                        System.out.println("it is a palindrome");
                }
                else{
                        System.out.println("not a palindrome");
                }}}
```

Loops with break statement:

break statement is to terminate the loop.



Program to check given number is octal number or not, using System.exit(0)

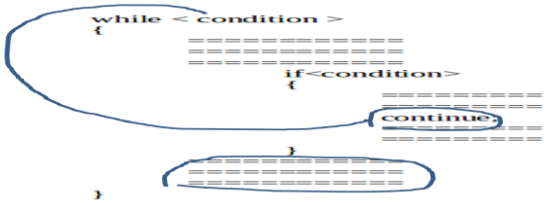
```
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
                Scanner sc = new Scanner(System.in);
                        System.out.println("Enter a number ");
                        int n = sc.nextInt();
                        while( n>0 ){
                                int r = n%10;
                                if( r > 7){
                                        System.out.println("not an Octal number");
                                        System.exit(0);
                                }
                                n = n / 10;
                        }
                        System.out.println("it is an octal number");
        } }
```

Program to check given number is octal number or not, using **break statement**

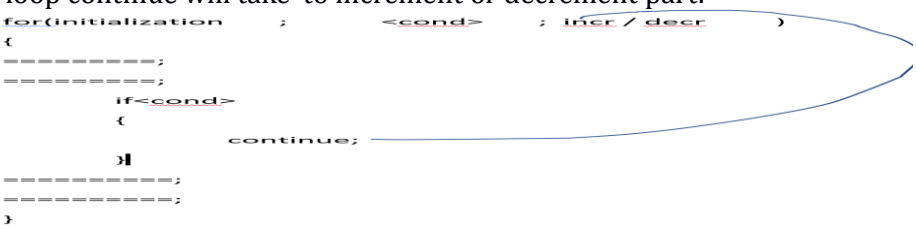
```
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
                        Scanner sc = new Scanner(System.in);
                        System.out.println("Enter a number ");
                        int n = sc.nextInt();
                        boolean flag = true;
                        while( n>0 ){
                                int r = n%10;
                                if( r > 7){
                                        flag=false;
                                        System.out.println("not an Octal number");
                                        break;
                                }
                                n = n / 10;
 }
                        if(flag){
                                System.out.println("it is an octal number");
                        }       }}
```

Loop with continue statement

it takes the control to beginning of the loop (condition) and skips the rest of the loop



With for loop continue will take to increment or decrement part.



program to skip even numbers and print only odd numbers 1 to n

```
import java.util.Scanner;
public class Test {
	public static void main( String [] args) {
			int i=1;
			while( i<=10 ){
				if( i%2 == 0 ){
					i++;
					continue;
				}
				System.out.println(i);
			i++;
			} } }
```

program to skip odd numbers and print only even numbers 1 to n

```
public class Test{
	public static void main(String[] args){
		int i=1;
		int n=20;
		while(i<=n){
			if(i%2 != 0){
				i++;
				continue;
			}
			System.out.println (i);
			i++;
		} } }
```

fibonacci series

1 1 2 3 5 8 13 21 34 55.....n

```
import java.util.Scanner;
public class Sample {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the limit ");
		int n = sc.nextInt();
		int f1 = 0;
		int f2 = 0;
		int sum = 1;
		while( sum<=n ){
			System.out.print(sum+"  ");
			f1 = f2;
			f2 = sum ;
			sum = f1 + f2;
		}  } }
```

Nested loops :

while<cond>

```
        {
        =========;
        =========;
                while<cond>
                {
                ==========;
                ==========;
                }
        ==========;
        ==========;
        }
for(  initialization  ;   condition       ;    incr/decr          )
{
=================;
=================;
        for( initialization  ; condition ; incr  /decr   )
        {
        ===============;
        ===============;
        }
================;
================;
}
```

Program to check given number is prime or not

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter the limit ");
                        int n = sc.nextInt();
                        boolean flag = true;
                        for(int i=2; i<n ; i++){
                                if( n%i == 0){
                                        flag = false;
                                        System.out.println("not a prime number");
                                break;
                                }
                        }
                        if( flag ){
                                System.out.print("it is a prime number  ");
                        }   } }
```

Program to print prime numbers 1 to n .

```
mport java.util.Scanner;
public class Test {
        public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
                System.out.println("enter the limit ");
                int n = sc.nextInt();
                for(int i = 1; i<=n; i++){
                        boolean flag = true;
                        for(int j=2; j<i ; j++){
                                if( i%j == 0){
                                        flag = false;
                                        break;
                                }
                        }
                        if( flag ){
                                System.out.print(i+"  ");
                        } } } }
```

Program to print following 1 to 10 tables

```
1 * 1 = 1  1 * 2 = 2  1 * 3 = 3  1 * 4 = 4  1 * 5 = 5  1 * 6 = 6  1 * 7 = 7  1 * 8 = 8
1 * 9 = 9  1 * 10 = 10
```

To print tables 1 to 10.

```
public class Test {
        public static void main(String[] args)  {
                for(int i=1;i<=10;i++){
                        System.out.println ();
                        for(int j=1;j<=10;j++){
                         System.out.println (i+" * "+j+" = "+(i*j));
                        } } } }
```

Program to print following.

```
1
1  2
1  2  3
1  2  3  4
1  2  3  4  5
```

```
public class Test{
public static void main(String[] args) {
                for (int i = 1; i <= 5; i++){
                        System.out.println ();
                        for (int j = 1; j <= i; j++){
                                System.out.print(j + "  ");
                        } } } }
```

```
1
2  2
3  3  3
4  4  4  4
5  5  5  5  5
```

```
public class Test {
        public static void main( String [] args) {
                for(int i=1; i<=5; i++){
                        System.out.println();
                  for(int j=1  ; j<=i  ; j++ )  {
                    System.out.print(i+" ");
                  } } } }
```

Program to print following

```
1  2  3  4  5
1  2  3  4
1  2  3
1  2
1
```

```
public class Test {
        public static void main(String[] args) {
                for (int i = 5; i >= 1; i--){
                        System.out.println ();
                        for (int j = 1; j <= i; j++){
                                System.out.print(j + "  ");
                        } } } }
```

```
5  5  5  5  5
4  4  4  4
3  3  3
2  2
1
```

```
public class Test{
        public static void main(String[] args) {
                for (int i = 5; i >= 1; i--){
                        System.out.println ();
                        for (int j = 1; j <= i; j++){
```

```
                    System.out.print(i + "   ");
                } } } }
```

Program to print following triangle

```
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
16  17  18  19  20  21
public class Test{
        public static void main(String[] args){
                int n=1;
                for (int i = 1; i <= 6; i++){
                        System.out.println ();
                        for (int j = 1; j <= i; j++){
                                System.out.print(n + "   ");
                                n++;
                        } } } }
```

Program to print following trianlge

```
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
public class Test {
        public static void main(String[] args){
                for (int i = 1; i <= 5; i++){
                        System.out.println ();
                        for (int j = 1; j <= i; j++){
                                System.out.print("*" + "   ");
                        } } } }
```

```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
public class Test {
        public static void main( String [] args) {
                for(int i= 1 ;  i<=5  ; i++ ){
                        System.out.println();
                  for( int j=5 ; j>=i ; j-- ){
                        System.out.print(j+" ");
                  } } } }
```

1 1 1 1 1
2 2 2 2
3 3 3
4 4
5

```
public class Test {
        public static void main( String [] args) {
                for(int i= 1 ;  i<=5  ; i++ ){
                        System.out.println();
                  for( int j=5 ; j>=i ; j-- ){
                        System.out.print(i+" ");
                  } } } }
```

Try below example:

```
1
1  2
1  2  3
```

```
1  2  3  4
1  2  3  4  5
1  2  3  4
1  2  3
1  2
1
```

```
import java.util.Scanner;
public class Test {
        public static void main( String [] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter limit ");
                int n = sc.nextInt();
                int i;
                for( i= 1 ;  i<=n  ; i++ ){
                        System.out.println();
                  for( int j=1 ; j<=i ; j++ ){
                        System.out.print(j+" ");
                  } }
                for(i = i-2; i>=1; i--){
                        System.out.println();
                        for(int j=1; j<=i; j++){
                                System.out.print(j+" ");
                        } } } }
```

```
        1
      1 2
    1 2 3
  1 2 3 4
1 2 3 4 5
```

```
public class Test {
        public static void main( String [] args) {
                for(int i=1; i<=5; i++){
                        System.out.println();
                         for(int j=5 ; j>i  ;  j-- ){
                          System.out.print("  ");
                         }
                         for(int j=1; j<=i; j++){
                                 System.out.print(j+" ");
                         } } } }
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
public class Test {
        public static void main(String[] args) {
                for(int i=5; i>=1; i--){
                        System.out.println();
                        for(int j=5; j>i ; j-- ) {
                                System.out.print("  ");
                        }
                        for(int j=1; j<=i; j++){
                                System.out.print(j+" ");
                        } } } }
```

Print following Diamond pattern

```
                1
            1   2   1
        1   2   3   2   1
    1   2   3   4   3   2   1
1   2   3   4   5   4   3   2   1
```

```
1 2 3 4 3 2 1
  1 2 3 2 1
    1 2 1
      1
```

Program for the above one

```
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("enter the limit");
		int n = sc.nextInt();
		int i;
		for( i=1; i<=n; i++){
			System.out.println();
			for(int j=n; j>i; j--){
				System.out.print("  ");
			}
			for(int j=1; j<=i; j++){
				System.out.print(j+" ");
			}
			for(int j = i-1 ; j>=1 ; j-- ){
				System.out.print(j+" ");
			}
		}
		for(i= i-2; i>=1; i--){
			System.out.println();
			for(int j=n; j>i; j--){
				System.out.print("  ");
			}
			for(int j=1; j<=i; j++){
				System.out.print(j+" ");
			}
			for(int j = i-1 ; j>=1 ; j-- ){
				System.out.print(j+" ");
			} } } }
```

Print following Diamond pattern

```
        5
      5 4 5
    5 4 3 4 5
  5 4 3 2 3 4 5
5 4 3 3 1 2 3 4 5
  5 4 3 2 3 4 5
    5 4 3 4 5
      5 4 5
        5
```

For the above one :

```
 package com.durga.mnrao.x;
import java.util.Scanner;
public class Test  {
	public static void main( String [] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the limit ");
		int n = sc.nextInt();
		int i;
		for( i = n; i>=1; i--){
			System.out.println();
			for(int j = 1; j<=i  ;j++  ){
				System.out.print("  ");
		  }
```

```
                for(int j=n;j>=i; j-- ){
                        System.out.print(j+" ");
                }
                for( int j = i+1; j<=n; j++){
                        System.out.print(j+" ");
                }
        }
        for( i = i+2; i<=n; i++){
                System.out.println();
                for(int j = 1; j<=i  ;j++ ){
                        System.out.print("  ");
          }
                for(int j=n;j>=i; j-- ){
                        System.out.print(j+" ");
                }
                for( int j = i+1; j<=n; j++){
                        System.out.print(j+" ");
                } } } }
```

Try Following Butterfly Pattern

```
1                                               1
1       2                                   2   1
1       2       3                   3       2   1
1       2       3       4       4   3       2   1
1       2       3       4   5   4   3       2   1
1       2       3       4       4   3       2   1
1       2       3                   3       2   1
1       2                                   2   1
1                                               1
```

Program for the above

```
import java.util.Scanner;
public class Test {
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the limit");
        int n = sc.nextInt();
        int i;
        for(i=1; i<=n; i++){
                System.out.println();
                        for(int j=1; j<=i; j++){
                                System.out.print(j+" ");
                        }
                        for(int j=n; j>i; j--)
                                System.out.print("  ");
                        }
                        for(int j=n; j>i; j--){
                                if(j==n){
                                        continue;
                                }
                                System.out.print("  ");
                        }
                        for(int j=i; j>=1; j--){
                                if(j==n){
                                        continue;
                                }
                                System.out.print(j+" ");
                        }
                }
                for(i = i-2 ;i>=1; i-- ){
                        System.out.println();
```

```
                for(int j=1; j<=i; j++){
                        System.out.print(j+" ");
                }
                for(int j=n; j>i; j--){
                        System.out.print("  ");
                }
                for(int j=n; j>i; j--){
                        if(j==n){
                                continue;
                        }
                        System.out.print("  ");
                }
                for(int j=i; j>=1; j--){
                        if(j==n){
                                continue;
                        }
                        System.out.print(j+" ");
} } }   }
```

Below pattern :

```
5           5
5 4         4 5
5 4 3       3 4 5
5 4 3 2     2 3 4 5
5 4 3 2 1 2 3 4 5
5 4 3 2     2 3 4 5
5 4 3       3 4 5
5 4         4 5
5           5
```

For the above one , program as follows

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter limit ");
                int n = sc.nextInt();
                int i ;
                for( i = n; i>=1 ; i--){
                        System.out.println();
                        for( int j= n; j>=i; j--){
                                System.out.print(j+" ");
                        }
                        for(int j = 1 ; j<i ; j++ ){
                                System.out.print("  ");
                        }
                        for(int j = 1 ; j<i ; j++){
                                if(j==1){
                                        continue;
                                }
                                System.out.print("  ");
                        }
                        for(int j =i; j<=n; j++){
                                if(j==1){
                                        continue;
                                }
                                System.out.print(j+" ");
                        }
                }
                for( i = i+2; i<=n ; i++) {
                        System.out.println();
```

```
                for( int j= n; j>=i; j--){
                        System.out.print(j+" ");
                }
                for(int j = 1 ; j<i ; j++ ){
                        System.out.print("  ");
                }
                for(int j = 1 ; j<i ; j++){
                        if(j==1){
                                continue;
                        }
                        System.out.print("  ");
                }
                for(int j =i; j<=n; j++){
                        if(j==1){
                                continue;
                        }
                        System.out.print(j+" ");
                } } } }
```

Below Pattern

```
1       1       1       1       1       1       1       1       1
        2       2       2       2       2       2       2
                3       3       3       3       3
                        4       4       4
                                5
                        4       4       4
                3       3       3       3       3
        2       2       2       2       2       2       2
1       1       1       1       1       1       1       1       1
```

Program for the below inner Diamond

```
1       2       3       4       5       4       3       2       1
1       2       3       4               4       3       2       1
1       2       3                               3       2       1
1       2                                               2       1
1                                                               1
1       2                                               2       1
1       2       3                               3       2       1
1       2       3       4               4       3       2       1
1       2       3       4       5       4       3       2       1
```

Program for the above

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("enter the limit");
                int n = sc.nextInt();
                int i;
                for(i=n; i>=1;  i--){
                        System.out.println();
                        for(int j=1; j<=i; j++){
                                System.out.print(j+" ");
                        }

                        for(int j=n; j>i; j--){
                                System.out.print("  ");
                        }
                        for(int j=n; j>i; j--){
                                if(j==n){
                                        continue;
                                }
```

```
                System.out.print("  ");
        }
        for(int j=i; j>=1; j--){
                if(j==n){
                        continue;
                }
                System.out.print(j+" ");
        }
}
for(i= i+2 ; i<=n  ;  i++  ){
        System.out.println();
        for(int j=1; j<=i; j++){
                System.out.print(j+" ");
        }
        for(int j=n; j>i; j--){
                System.out.print("  ");
        }
        for(int j=n; j>i; j--){
                if(j==n){
                continue;
                }
                System.out.print("  ");
        }
        for(int j=i; j>=1; j--){
                if(j==n){
                        continue;
                }
                System.out.print(j+" ");
        } } } }
```

Java-Arrays

Array is a collections of similar type of elements

Simple Variables, which contains data

Eg: int x=100, y=200, z=300;

array declaration

int a[] ; reference to array.

(or)

int []a; reference to array.

Memory allocation :

a = new int[5]; // dynamic (runtime) memory allocation for the array

declaration as well memory allocation at a time:

int []a = new int[5];

new is a keyword to allocate memory at runtime (dynamically)

new always works in the heap memory

dynamic memory management is always in heap memory.

Access again array elements:

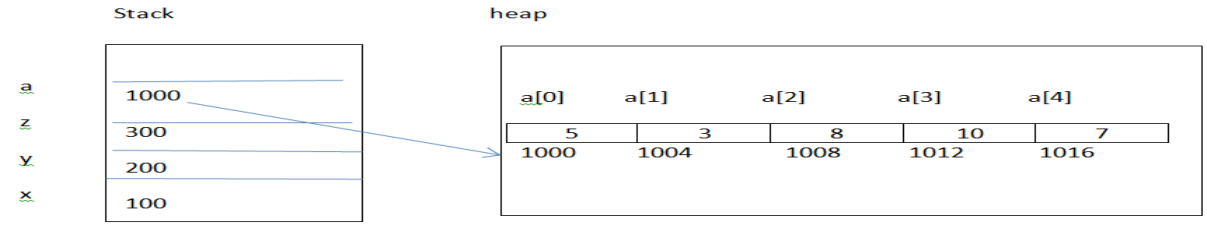
a[0] 1st element a[1] 2nd element a[2] 3rd element a[3] 4th element a[4] 5th element

assignment of values to array

a[0]=5; a[1]=3; a[2]=8; a[3]=10; a[4]=7;

a[5]=20; // it throws Exception ArrayIndexOutOfBoundsException

memory map of arrays:



Why index starts from 0 .

a[0] ⍰ a + 0 ⍰ 1000 + 0 * 4 = 1000 a[1] ⍰ a + 1 ⍰ 1000 + 1 * 4 = 1004 a[2] ⍰ a + 2 ⍰ 1000 + 2 * 4 = 1008 a[3] ⍰ a + 3 ⍰ 1000 + 3 * 4 = 1012 a[4] ⍰ a + 4 ⍰ 1000 + 4 * 4 = 1016

Simple variables contains data and reference variables contains address.

Array is a reference variable in java .

In the above **x, y and z** are called as simple variables, and **a (array)** called as reference variable

initializing an array :

int []a={1,2,3,4,5,6 }; compile time initialization.

int []a=new int[]{1,2,3,4,5,6}; run time memory allocation.

int []a=new int[6]{1,2,3,4,5,6}; invalid

int a[10]={1,2,34,5.....};// is not valid in java

int n = a.length --> returns no of elements in array

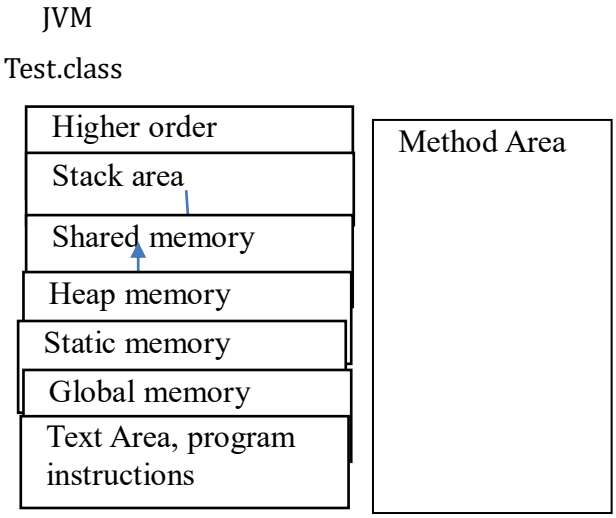
a.length ⍰ gives no of elements in array

length :

it is a variable provided by JVM . it is not from any pre-defined package (such as java. Lang) Memory Management:

Java Memory Management :

Child Process and it's memory concept



program to print array elements

```
 public class Test{
public static void main(String[] args){
                int a[]={1,2,3,4,5};   // compile time memory allocation
                for(int i=0; i<a.length;i++){
                System.out.println (a[i]);
        } } }
public class Test{
        public static void main(String[] args){
                int a[]=new int[]{1,2,3,4,5}; // run time memory allocation
                for(int i=0; i<a.length;i++){
                        System.out.println (a[i]);
                } } }
```

Different ways of arrays

```
1) int a[];
a=new int[5];
a[0]=10; a[1]=15; a[2]=21; a[3]=25;    a[4]=30;
2) int a[]=new int[]{1,2,3,4,5};  // run time initialization
3) int a[]={1,2,3,4,5}; // compile time initialization
```

Declaration of multiple arrays

```
int a[], b[], c[];  // all these are arrays

a=new int[6];
b=new int[9];
c=new int[8];
```

```
int [] a,b,c; // all these are arrays
a=new int[6];
b=new int[9];
c=new int[8];

int  a[], b, c // here b and c  are simple variables not arrays
 a=new int[6];
b=new int[9];  // Invalid, it is not an array
c=new int[8];  // Invalid, it is not an array
```

Program to find min and max of given elements in array.

```
public class Test{
	public static void main(String[] args){
		int a[]={11,21,32,-14,5,66};
		int max=a[0];
		int min=a[0];
		for(int i=1;i<a.length;i++) {
			if(a[i]>max){
				max=a[i];
			}
			if(a[i]<min){
				min=a[i];
			}  }
		System.out.println ("max ="+max);
		System.out.println ("min ="+min);
	} }
```

Program to find min , max, sum and avg of given elements in array.

```
public class Test {
	public static void main(String[] args) {
		int [] a = {10,18,15,25,14,3,12};
		int max=a[0];
		int min=a[0];
		for( int i=1; i<a.length; i++){
			if(a[i]>max){
				max=a[i];
			}
			if(a[i]<min){
				min=a[i];
			}  }
		int sum=0;
		for (int i = 0; i < a.length; i++) {
			sum = sum + a[i];
		}
		float avg = (float)sum/a.length;
		System.out.println("count of numbers "+a.length);
		System.out.println("max = "+max);
		System.out.println("min = "+min);
		System.out.println("Sum = "+sum);
		System.out.println("avg = "+avg);
	}  }
```

Above program with bettere logic including sum in the same loop:

```
public class Test {
	public static void main(String[] args) {
		int [] a = {10, 18,15,25,14,3,12};
		int max = a[0];
		int min = a[0];
		int sum = a[0];
		for( int i=1; i<a.length; i++){
			sum = sum + a[i];
```

```
                    if(a[i]>max){
                            max=a[i];
                    }
                    if(a[i]<min){
                            min=a[i];
                    }       }
            float avg = (float)sum/a.length;
            System.out.println("count of numbers "+a.length);
            System.out.println("max = "+max);
            System.out.println("min = "+min);
            System.out.println("Sum = "+sum);
            System.out.println("avg = "+avg);
    } }
```

Above program to read elements from keyboard

```
import java.util.Scanner;
public  class Test {
    public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            System.out.println("How many numbers you want : ");
            int n = sc.nextInt();
            int [] a = new int[n];
            for(int i=0; i<a.length; i++){
                    System.out.println("Enter the element");
                    a[i] = sc.nextInt();
            }
            int max = a[0];
            int min = a[0];
            int sum = a[0];
            for( int i=1; i<a.length; i++){
                    sum = sum + a[i];
                    if(a[i]>max){
                            max=a[i];
                    }
                    if(a[i]<min){
                            min=a[i];
                    }   }
            float avg = (float)sum/a.length;
            System.out.println("count of numbers "+a.length);
            System.out.println("max = "+max);
            System.out.println("min = "+min);
            System.out.println("Sum = "+sum);
            System.out.println("avg = "+avg);
    } }
```

Accessing array elements using for-each loop:

```
public class Test{
    public static void main(String[] args){
            int a[]={10,8,13,18,15,6};
            foreach (int  i : a){
            System.out.println (i);
            } } }
```

Iterative / Regular for loop is for any iterative statements
but for-each loop only for arrays.
with Iterative for loop we can skip beginning elements or ending elements of array .
but for-each loop we should process all elements of array.
When processing all the elements of array, for-each loop gives better performance.

Array of Strings:

It is a collection of strings. String is a class from java. lang package to handle the strings.

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

```
        String  [] names={"hadoop","java","html","oracle","hive","spark","pig"};
        for(int i=0; i<names.length; i++){
                System.out.println (names[i]);
        }   }  }
```

Using for-each loop:

```
public class Test {
    public static void main(String []  args){
        String  [] names={"hadoop","java","html","oracle","hive","spark","pig"};
        for(String name : names){
                System.out.println (name);
        }  } }
```

Searching Techniques:

1) **Linear search**
2) **Binary search**

Linear search:

It is a searching for the element from the beginning to end . Program for linear search:

```
import java.util.Scanner;
public  class Test {
    public static void main(String[] args) {
        int [] a ={10,25,15,8,50,12,80,23,65,78,32,49,37,28,83,90,38,65,54};
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the number you want to search");
        int num = sc.nextInt();
        boolean searchFlag = false;
        for(int i=0; i<a.length;i++){
                if(num == a[i]){
                        searchFlag = true;
                        System.out.println("your number found at a["+i+"]");
                        break;
                }
        }
        if( ! searchFlag){
                System.out.println("your number not found");
        }   }  }
```

Duplicate Search:

```
package com.nrit.mnrao.test;
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        int [] nums = {10,18,15,10, 14,3,12,30, 80,10,36,45, 90,87,10,20, 40,75,10,100};
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number You Want to search");
        int num = sc.nextInt();
        int counter=0;
        boolean searchFlag=false;
       for(int i=0; i<nums.length; i++){
         if(num==nums[i]){
                counter++;
                searchFlag=true;
                System.out.println("Your number found at nums["+i+"]");
         }
      }
      if( ! searchFlag){
            System.out.println("Your number not found");
      }
          System.out.println("no of occurances "+counter);
          }}
```

Binary Search:

It is a searching for the elements in half of the actual number of elements each iteration.

```
import java.util.Scanner;
public  class Test {
        public static void main( String [] args) {
                int [] a = {10,12,18,21,25,30,35,38,40,42,50,53,55,65,69,71,78,80,85,88 };
                int l = 0;
                int h = a.length-1;
                Scanner sc = new Scanner(System.in);
                System.out.println("enter the number you want to search ");
                int num = sc.nextInt();
                boolean searchFlag = false;
                while( l<=h )               {
                        int m = ( l + h )/2;
                        if( num == a[m] ) {
                                searchFlag = true;
                                System.out.println("your number found at ["+m+"]");
                                break;
                        }
                        else if ( num > a[m] ){
                         l = m+1;
                        }
                        else{
                        h = m-1;
                        }
                }
                if( !searchFlag ) {
        System.out.println("your number not found");
        }       }       }
```

Note:

1) In linear search numbers can be random order and duplicate.
2) In Binary search numbers must be in ascending / descending order and unique value.

How to sort an Array in Java

ArraySort.java

```
package com.nrit.mnrao.test;

import java.util.Arrays;

public class Test {
        public static void main(String[] args) {
        int [] a={10,8,17,6,100,50,23,65,36,29};
        System.out.println("Before Sorting ");
        for (int i = 0; i < a.length; i++) {
                System.out.println(a[i]);
        }
        Arrays.sort(a);
                System.out.println("After Sorting ");
                for (int i = 0; i < a.length; i++) {
                        System.out.println(a[i]);
                } } }
package com.nrit.mnrao.test;
import java.util.Arrays;
public class Test {
        public static void main(String[] args) {
        String [] names={"unix","linux","java","hadoop","scala","python","html","oracle"};
        System.out.println("Before Sorting ");
        for (int i = 0; i < names.length; i++) {
                System.out.println(names[i]);
```

```
		}
		Arrays.sort(names);
		System.out.println("After Sorting ");
		for (int i = 0; i < names.length; i++) {
			System.out.println(names[i]);
} } }
```

2-Dimensional Array :

It is with rows and columns

Declaration:

```
int [][] a = new int[rows][cols];

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

High Level Memory Map:

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

Accessing elements:

a[0][0]; 1st row , 1st element

a[0][1]; 1st row , 2nd element

a[0][2]; 1st row , 3rd element

a[1][0]; 2nd row , 1st element
a[1][1]; 2nd row , 2nd element
a[1][2]; 2nd row , 3rd element
a[2][0]; 3rd row , 1st element
a[2][1]; 3rd row , 2nd element
a[2][2]; 3rd row , 3rd element

Assinging values :

```
a[0][0]=1;
a[0][1]=2;
a[0][2]=3;
a[1][0]=4;
a[1][1]=5;
a[1][2]=6;
a[2][0]=7;
a[2][1]=8;
a[2][2]=9;
```

displaying two dimensional arrays Using for loop :

```
package com.durga.mnrao.x;
public class Test {
	public static void main(String[] args) {
		int [][] a = new int[3][3];
		a[0][0]=1;
		a[0][1]=2;
		a[0][2]=3;
```

```
a[1][0]=4;
a[1][1]=5;
a[1][2]=6;

a[2][0]=7;
a[2][1]=8;
a[2][2]=9;
for(int i = 0; i<3; i++){
        System.out.println();
        for(int j = 0; j<3; j++){
                System.out.print(a[i][j] +"  ");
        }   }  } }
```

Initializations :

Compile time inialization :

```
int [][] a = {{1,2,3},{4,5,6},{7,8,9}};
        for(int i = 0; i<3; i++){
                System.out.println();
                  for(int j = 0; j<3; j++){
                        System.out.print(a[i][j] +"  ");
                }   }
```

Runtime initialization :

new keyword , always allocates memory at run time.

```
int [][] a = new int[][]{{1,2,3},{4,5,6},{7,8,9}};
```

Array rows with different no of columns :

```
int [][] a = { {1,2,3},  {4,5,6,7},   {8,9},  {10,11,12,13,14} };
```

how to find no of row in two dimensional array ?

how to find no of columns in each row of two dimensional array ?

No of rows in two dimensional array

a.length no of rows

No of columns in each row of two dimensional array

a[0].length no of elements in 1st row
a[1].length no of elements in 2nd row
a[2].length no of elements in 3rd row
a[3].length no of elements in 4th row

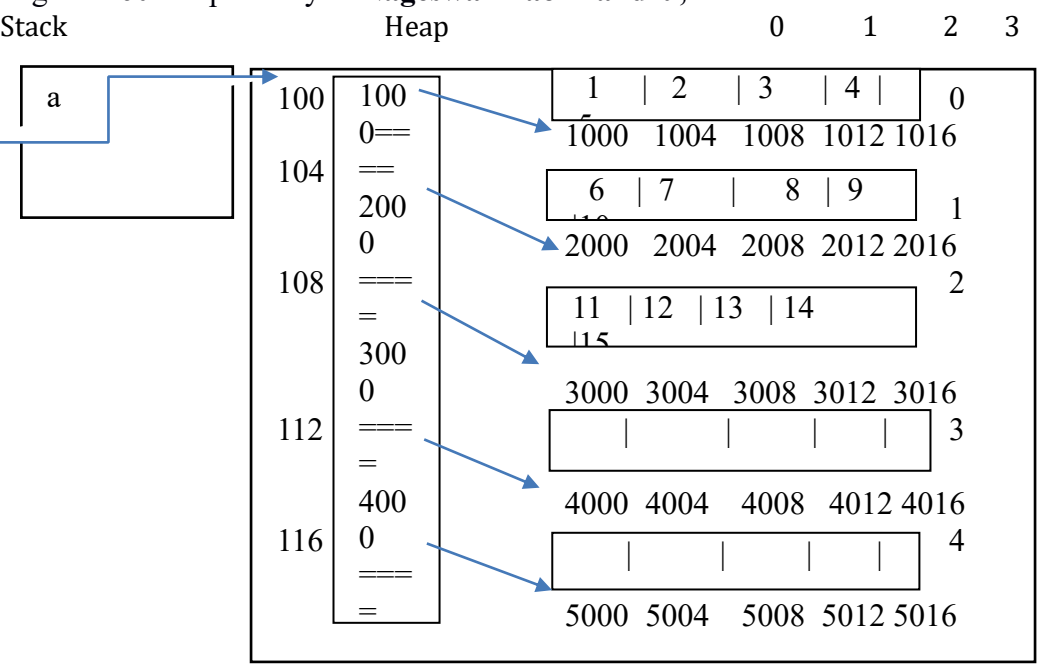
```
package com.durga.mnrao.x;
public class Test {
        public static void main(String[] args){
                int [][] a = {{1,2,3},{4,5,6,7},{8,9},{10,11,12,13,14}};
                for(int i = 0; i<a.length; i++){
                        System.out.println();
                        for(int j = 0; j<a[i].length; j++){
                                System.out.print(a[i][j] +"  ");
}   }  }  }
```

Memory Map of Two – Dimensional Arrays

Dynamic Memory Management for Two Dimensional Arrays :

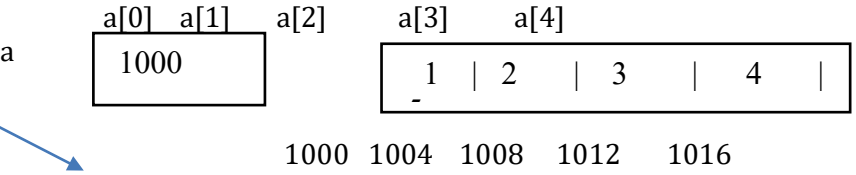
Eg:

```
int [][] a=new int[][]{{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20},{21,22,23,24,25}};
```



Single dimension array

int [] a = new int [5];



No of elements = a.length 1000.length (base address). length

Two Dimensional arrays

a[0] 1000[0] *(1000 + 0 * 4) *(1000) 1

a[1] 1000[1] * (1000 + 1 * 4) *(1004) 2

a[2] 1000[2] *(1000 + 2 * 4) *(1008) 3

a[3] 1000[3] *(1000 + 3 * 4) *(1012) 4

a[4] 1000[4] * (1000 + 4 * 4) * (1016) 5

base address [0] 1st element

base address [1] 2nd element

base address [2] 3rd element

base address [3] 4th element

base address [4] 5th element

to access value 4 in the 1st row

1000[3] 4

100[0][3] 4

a[0][3] 4

to access value 15 in the 3rd row

3000[4] 15

100[2][4] 15

a[2][4] 15

==========;

No of elements in each row

No of elements in 1st row 1000.length 100[0].length a[0].length

No of elements in 2nd row 2000.length 100[1].length a[1].length

No of elements in 3rd row 3000.length 100[2].length a[2].length

No of elements in 4th row 4000.length 100[3].length a[3].length

No of elements in 5th row 5000.length 100[4].length a[4].length

No of rows in the two dimensional array ? base address . length ? 100.length ? a.length

Java follows , ANSI - C++ Standards ,
Java reference is same as C++ pointer ,
Size of any type of pointer in C++ is **4 bytes**
Size of java reference variable = **4 bytes**

Program For the following output :

```
1       2       3
4       5       6
7       8       9
```

```
public class Test {
        public static void main(String[] args) {
                int [][]a = {{1,2,3},{4,5,6,},{7,8,9}};
                for(int i=0; i<a.length;i++){
                        System.out.println ();
                        for(int j=0;j<a[i].length;j++){
                                System.out.print(a[i][j]+"\t");
                        } } } }
```

```
public class Test {
        public static void main(String[] args) {
                int [][]a = new int [][]{{1,2,3},{4,5,6},{7,8,9}};
                for(int i=0; i<a.length;i++){
                        System.out.println ();
                        for(int j=0;j<a[i].length;j++){
                                System.out.print(a[i][j]+"\t");
                        }  }  }  }
```

Program for variable no of elements in each row :

```
public class Test {
        public static void main(String[] args) {
                int [][] a = {{1,2,3},{6,7},{11,12,13,14,15},{16},{21,22,24,25}};
                for(int i = 0; i< a.length ; i++){
                        System.out.println();
                        for(int j = 0 ; j<a[i].length ; j++){
                                System.out.print(a[i][j]+" ");
                        }  }  }  }
```

Program For the following output:
Without wassage of memory space

```
1
1       2
1       2       3
1       2       3       4
1       2       3       4       5
```

1st step:

```
int [][]a; ? declaration.
                a=new int[5][];// memory allocation for array of references(rows)
                //memory allocation for columns in each row.
                for(int i=0;i<a.length;i++){
                        a[i]= new int[i+1];
                }
//checking for no of rows
                System.out.println ("no of rows : "+a.length);
//checking for no of cols
                System.out.println ("1st row elements : "+a[0].length);
                System.out.println ("2nd row elements : "+a[1].length);
                System.out.println ("3rd row elements : "+a[2].length);
                System.out.println ("4th row elements : "+a[3].length);
                System.out.println ("5th row elements : "+a[4].length);
```

2nd step:
assigning values:

```
        for(int i=0;i<a.length;i++) {
                for(int j=0;j<a[i].length;j++) {
                        a[i][j]=j+1;
                }   }
```

Display:

```
for (int i = 0; i < a.length; i++) {
        System.out.println ();
        for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + "\t");
        }  }
```

Program for the above

```
public class Test {
        public static void main(String[] args) {
                int [][] a;
                a = new int[5][];
                for(int i = 0 ; i<a.length; i++){
                        a[i] = new int[i+1];
                }
                for(int i =0; i<a.length; i++){
                  for(int j = 0; j<a[i].length; j++)
                  {
                        a[i][j]= j+1;
                  }
                }
                for(int i =0; i<a.length; i++){
                        System.out.println();
                  for(int j = 0; j<a[i].length; j++){
                        System.out.print(a[i][j] +" ");
                  }  }  }  }
```

Eg:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
public class Test {
        public static void main(String[] args) {
                int [][] a;
                a = new int[5][];
                for(int i = 0 ; i<a.length; i++){
                        a[i] = new int[i+1];
                }
                for(int i =0; i<a.length; i++){
                  for(int j = 0; j<a[i].length; j++)  {
                        a[i][j]= i+1;   // here logic changes
                  }}
                for(int i =0; i<a.length; i++){
                  System.out.println();
                  for(int j = 0; j<a[i].length; j++){
                        System.out.print(a[i][j] +" ");
                  }  }  }  }
```

Eg:

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

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

```
        int [][]a;
        a = new int[5][];
        for(int i=0; i<a.length; i++){
                a[i] = new int[a.length-i];
        }
        for(int i=0; i<a.length ; i++){
         for(int j=0; j<a[i].length ; j++) {
                a[i][j]= j+1;
         }
        }
        System.out.println("Array elements are ");
        for(int i=0; i<a.length ; i++){
                System.out.println();
         for(int j=0; j<a[i].length ; j++) {
                System.out.print(a[i][j]+"  ");

          }  }  } }
```

Eg:

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

```
public class Test {
        public static void main(String[] args) {
                int [][]a;
                a = new int[5][];
                for(int i=0; i<a.length; i++){
                        a[i] = new int[a.length-i];
                }
                for(int i=0; i<a.length ; i++){
                 for(int j=0; j<a[i].length ; j++) {
                        a[i][j]= a.length-i; // here logic changes
                 }
                }
                System.out.println("Array elements are ");
                for(int i=0; i<a.length ; i++){
                        System.out.println();
                 for(int j=0; j<a[i].length ; j++) {
                        System.out.print(a[i][j]+"  ");
                  }
                } }  }
```

Java-Working with Methods

Working with Methods

Method is set of statements to perform some task. Every statement of method should be terminated with semicolon (;)

syntax:

```
Access_specifier   Return_type  method_name ( type  arg1,  type arg2,  type arg3,  .....)
{
   ============;
   ============;
  return value;
}
```

Access_specifier :

1) public
2) protected
3) private.
4) nothing (default)

Return_type:

depends on return value.
Eg:
return 10; --> return type is **int.**
Return 10.5; --> return type is **double.**
No return value --> return type is **void**

defining a method :

```
public  void  display(){
   ============;
   ============;
}
```

Calling a method:

```
public static void main(String[] args) // calling method
{
          ============;
         display(); // calling a method.
          ============;
         ============;
}
public  void display()// called method.
{
         ============;
          ============;
 }
```

There are four types of methods

1) Method with no args and no return value
2) Method with no args and return value
3) Method with args and no return value
4) Method with args and return value

1)Method with no args and no return value

eg:

```
public class Test{
        public static void main(String[] args){
                        System.out.println ("main start");
                        display();
                        System.out.println ("main end");
        }
        public static void display(){
                System.out.println ("I am in display");
        } }
```

O/p:
main start I am in display main end

eg:

```
public class Test{
        public static void main(String[] args){
                System.out.println ("Main start");
                System.out.println ("Before display");
                display();
                System.out.println ("After display");
                System.out.println ("Before show");
                show();
                System.out.println ("after show");
                System.out.println ("Main End");
        }
        public static void display(){
                System.out.println ("I am in display");
        }
        public static void show(){
                System.out.println ("I am in show");
        }   }
```

Main start, Before display, I am in display, After display, Before show, I am in show, after show, Main End

```
public class Test{
        public static void main(String[] args){
                        System.out.println ("main start");
                        System.out.println ("before display");
                        display();
                        System.out.println ("after display");
                        System.out.println ("before  show");
                        show();
                        System.out.println ("after show");
                        System.out.println ("main end");
        }
        public static void display(){
                System.out.println ("display start");
                show();
                System.out.println ("display end");
        }
        public static void show(){
                System.out.println ("I am in show");
}   }
```

O/P:

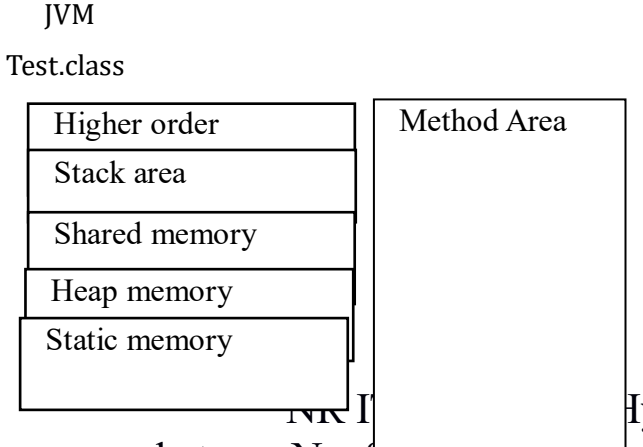
Main start, Before display, I am in display, After display, Before show, I am in show, after show, Main End

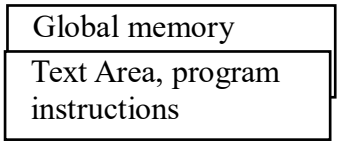
example to explain about method stack

```
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args) {
                System.out.println("main start");
                display();
                System.out.println("after display");
                show();
                System.out.println("after show ");
                demo();
                System.out.println("main end");     }
        public static void display() {
        System.out.println("display start");
                show();
                System.out.println("display end");
        }
        public static void show(){
                System.out.println("show start");
                demo();
                System.out.println("Show end ");        }
        public static void demo(){
                System.out.println("I a in demo");       }
}
```

Java Memory Management :

Child Process and it's memory concept





Text Area (with program instructions)

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

		System.out.println("main start");
		display();
		System.out.println("after display");
		show();
		System.out.println("after show ");
		demo();
		System.out.println("main end");
	}		public static void display()	{
		System.out.println("display start");
		show();
		System.out.println("display end");
	}
	public static void show()
	{	System.out.println("show start");
		demo();
		System.out.println("Show end ");
	}
	public static void demo()
	{
		System.out.println("I a in demo");
	}
```

Recursive method:

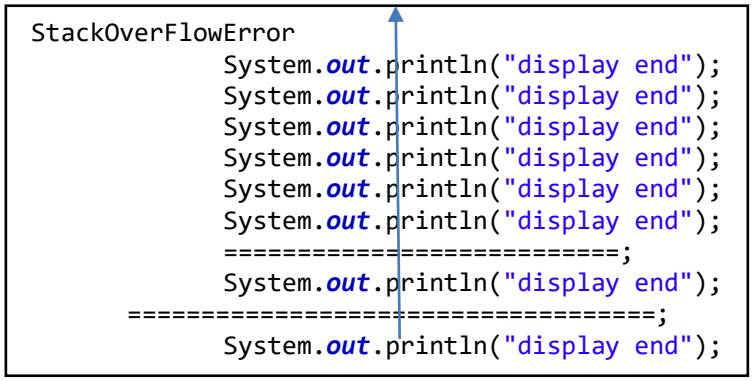
method calling itself is a recursive method:

eg:

```
public void display(){
	System.out.println ("I am in display");
	display();
	System.out.println ("display end");
}
```

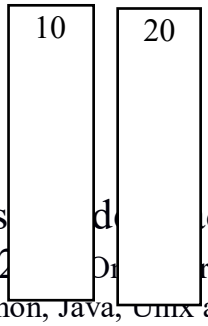
Recursive methods are not recommended. There is a chance of raising stack StackOverFlowError

Method Area (stack):

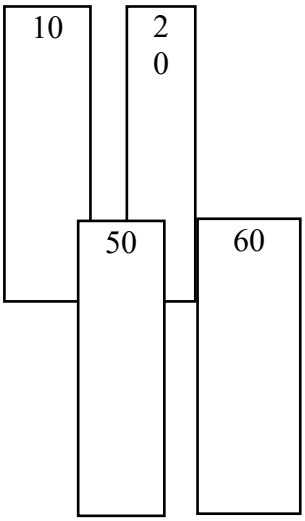


2)Method with parameters /arguments and no return value

```
public class Test{
	public static void main(String[] args){
		int a=10;		a		b
		int b=20;
		System.out.println ("Main start");
		display( a , b);  // here a and b are parameters
		System.out.println ("After display");
```



```
                System.out.println ("Before show");
                show(50,60);    // here 50 and 60 are parameters
                System.out.println ("after show");
                System.out.println ("Main End");
        }//main  close
                        // here x and y are argument
        public static void display(int  x,   int  y) {
                                  X                  y
                System.out.println ("I am in display");
                System.out.println ("args are "+x+"\t"+y);
        }//display close
        public static void show(int  x,  int y)
        {
                                    x                  y
                System.out.println ("I am in show");
                System.out.println ("args are "+x+"\t"+y);
                //show close
        }
}//class close
```




```
Main start    I am in display
args are 10       20
After display  Before show  I am in show
args are 50       60
after show Main End
```

Note : arguments are local to method

3)Method with Arguments and return values :

```
public class Test{
        public static void main(String[] args){
                int a=10;
                int b=20;
                long c = addNum(a, b);
                System.out.println ("addition "+c);
                 c = subNum(a, b);
                System.out.println ("substraction "+c);
                 c = mulNum(a,b);
                System.out.println ("multiplication "+c);
        }//main  close
        public static long addNum(int x, int y){
                long temp=x+y;
                return (temp);
        }
        public static long subNum(int x, int y){
                long temp = x-y;
                return temp;
        }
        public static long mulNum(int x ,int y){
                long temp = x*y;
                return temp;
        }
}//class close
```

```
addition 30    substraction -10    multiplication 200
```

Return key word with no return value:

This is to terminate the method.

Eg:

```
        public  void display(int x,  int y){
                ================
                ================
                if(cond){
                  return; // if condtion is true , return key word executes and terminates the method
```

```
        }
        ================
        ================
    }
```

Purpose of return statement :

1) **To return value from method**
2) **To terminate the method**

Sample program.

```
public class Test{
    public static void main(String[] args){
        int a;
        int b;
        System.out.println ("Main start");
        a=30;
        b=20;
        greater(a,b);
        System.out.println ("after greater one");
        a=20;
        b=30;
        greater (a,b);
        System.out.println ("after greater two");
        System.out.println ("Main end");
    }//main  close
    public static void greater (int x,  int y){
        System.out.println ("greater start");
        if(x>y){
            return ;
        }
        System.out.println ("greater end");
    }
}//class close
```

O/P;

Main start *greater* start after *greater* one
greater start *greater* end // only one time as condition is true for the first time.
after *greater* two Main end

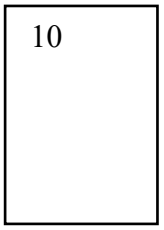
4)Method with no args and retun value:

```
package com.nrit.mnrao.test;
public class Test {
    public static void main(String[] args) {
        System.out.println("main start");
        String str = getMessage();
        System.out.println("return value from getMessage = "+str);
        System.out.println("main end");
    }
    public static String getMessage(){
        return "Hello Java";
    }  }
```

A method can have multiple return statements but only return statement executes .

```
    public void display()
    {
    =============;
    =============;
        if<cond1>
        {
         return ;
        }
    ============;
    ===========;
        if<cond2>
```

```
        {
          return ;
        }
    ============;
    ============;
        if<cond3>
        {
          return ;
        }
     ============;
    ============;
    }
```



Call by value and call by reference methods:

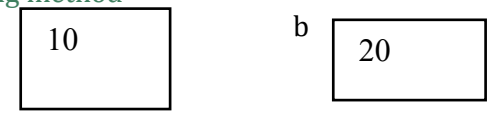
1) **Call by value**
   It is a calling method by passing value of a variable
2) **call by Reference**
   It is a calling method by passing address (reference) of a variable

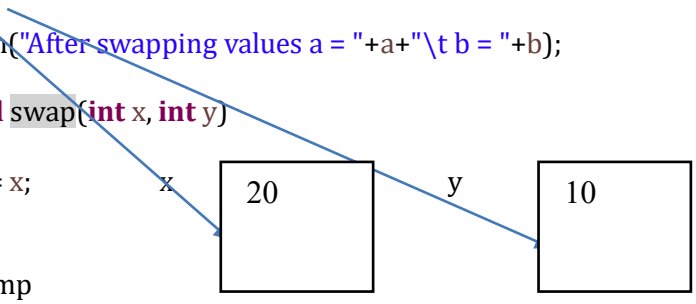
In Java

1) passing variable to method is always call by value , no call by reference for variable
2) passing Object to method is always call by reference, no call by value for variable

Call by value :

```
public class Test {
        public static void main(String[] args) {        //calling method
                int a=10;   a
                int b=20;
System.out.println("Initial values a = "+a+"\t b = "+b);
                swap(a,b);
        System.out.println("After swapping values a = "+a+"\t b = "+b);
        }
        public static void swap(int x, int y)
        {
                int temp = x;
                 x = y;
                 y = temp;
        }                temp
}
```





in call by value, method arguments are local variables, local copies creates for every argument.
Any changes made on arguments , will not be reflected actual copies .

Method Overloading

1. Method overloading takes place between the same method with different signature
2. Method signature defines, name of the method, no. of args , type of args and return type of the method.
3. Method overloading is a compile time process.

First compiler checks no. of arguments, if no. of args are matched between methods then checks for type of arguments, if type is also matched then ambiguity b/w methods and compiler shows error massage.
Method over loading is compile time binding (or) static binding.

1) Method Overloading based on no of arguments
2) Method Overloading based on type of arguments

Overloading based of no.of arguments :

```
public class Sample{
        public void display(){
                System.out.println ("No args");
        }
        public void display(int x){
                System.out.println ("one args");
        }
        public void display(int x, int y){
```

```
                System.out.println ("two args");
        }
        public void display(int x, int y, int z){
                System.out.println ("three args");
        }
}
public class Test{
        public static void main(String[] args){
                Sample s1 = new Sample();
                s1.display();
                s1.display(10);
                s1.display(20, 30);
                s1.display(10, 20, 30);
        } }
```

Method overloading with data :

```
public class Sample{
        private  int a;
        private  int b;
        private  int c;
        public void setData() {
                a=0;
                b=0;
                c=0;
        }
        public void setData(int x) {
                a = x;
                b = 0;
                c = 0;
        }
        public void setData(int x, int y) {
                a = x;
                b = y;
                c = 0;
        }
        public void setData(int x, int y, int z) {
                a = x;
                b = y;
                c = z;
        }
        public void display() {
                System.out.println ("a="+a+"\tb="+b+"\tc="+c);
        }
}
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                Sample s4 = new Sample();
                s1.setData();
                s2.setData(15);
                s3.setData(10,20);
                s4.setData(15,25,30);
                System.out.println ("First Object");
                s1.display();
                System.out.println ("Second Object");
                s2.display();
                System.out.println ("Third Object");
                s3.display();
```

```
                System.out.println ("Fourth Object");
                s4.display();
        } }
```

Overloading based on type of arguments (parameters) :

```
public class Sample{
        private  int a;
        private  float b;
        private  char  ch;
        public void setData(){
                a=0;
                b=0.0f ;
                ch='\0';
        }
        public void setData(int x, float y){
                a=x;
                b=y;
        }
        public void setData(int x, char y){
                a=x;
                ch=y;
        }
        public void setData(float x, char y ){
                b=x;
                ch=y;
        }
        public void display(){
                System.out.println ("a="+a+"\tb="+b+"\tch="+ch);
        } }
public class Test{
        public static void main(String[] args){
                        Sample s1 = new Sample();
                        Sample s2 = new Sample();
                        Sample s3 = new Sample();
                        Sample s4 = new Sample();
                        s1.setData();
                        s2.setData(10,20.0f);
                        s3.setData(50.5f, 'A');
                        s4.setData(100, 'B');
                        s1.display();
                        s2.display();
                        s3.display();
                        s4.display();
        }   }
```

Method Overloading and Type Promotion

1) Small type of data passing big type of argument :

It is an implicit process

```
public class Sample {
        public void setData(long x){
        }   }
public class Test {
        public static void main( String [] args) {
                byte b = 10;
                short s = 100;
                int i = 1000;
                long l = 10000;
                Sample s1 = new Sample();
                s1.setData(b); // valid , small type passing to big type
                s1.setData(s);// valid , small type passing to big type
                s1.setData(i); // valid , small type passing to big type
```

```
            s1.setData(l); // valid , same type passing
        } }
```

Integer type of data passing real number type of arguments

It is an implicit process

```
public class Sample {
        public void setData(double x){
        } }
public class Test {
        public static void main( String [] args) {
                byte b = 10;
                short s = 100;
                int i = 1000;
                long l = 10000;
                float f = 10.5f;
                Sample s1 = new Sample();
                s1.setData(b); // valid , byte type passing to double type
                s1.setData(s);// valid , short type passing to double type
                s1.setData(i); // valid , int type passing to double type
                s1.setData(l); // valid , long type passing to double type
                s1.setData(f); // valid, float type passing to double type
        } }
```

Reverse :

1) **Big type to small type**
2) **Real number to integer**

For the above explicitly casting required

1) **Big type to Small type :**

```
public class Sample {
        public void setData(byte x){
        } }
public class Test {
        public static void main( String [] args){
                short s = 10;
                int i = 100;
                Sample s1 = new Sample();
                s1.setData( (byte) s); // big to small, explicitly casting required
                s1.setData( (byte) i ); // big to small, explicitly casting required
        } }
```

Another example :

```
public class Sample {
        public void setData(float x){
        } }
public class Test {
        public static void main( String [] args) {
                double d = 10.5;
                Sample s1 = new Sample();
                s1.setData( (float) d ); // big to small type, casting requred
        } }
```

Real number to interger

```
public class Sample {
        public void setData(int x){
        } }
public class Test {
        public static void main( String [] args) {
                float f = 10.5f;
                double d = 20.5;
                Sample s1 = new Sample();
                s1.setData( (int) f ); // real  to int type, casting requred
                s1.setData( (int) d); // real  to int type, casting requred
```

```
    } }
```

implicitly passing for below one

1) small type to big type
2) integer to real number

explicitly type casting required for below

1) big type to small type
2) real number to integer

Important Note :

Method Overloading checking for only **no of args and type of args** of the method, never check for return type of the method. Below one not valid (return type is different)

```
package com.durga.mnrao.test;
public class Sample {
        public void setData(int x, float y ){
        }
        public int setData(int x, float y ){
                return 10;
        }}
```

Q1.What is data encapsulation and what's its significance?

Ans: Encapsulation is a concept in Object Oriented Programming for combining properties and methods in a single unit. Encapsulation helps programmers to follow a modular approach for software development as each object has its own set of methods and variables and serves its functions independent of other objects. Encapsulation also serves data hiding purpose.

OOP's Concept

Object Oriented Programming was introduced to overcome the dis-advantages of C-Language.

C-Language dis-advantages:

In C-Language every thing is global. Data and functions are global. Data can be accessed from anywhere in the application. Any function can be called from anywhere. Since everything is global, there is no security for data. Chance of corrupting data in C-Language. OOP's Concept was introduced to achieve data security .

OOP (Object Oriented Programming) is a concept, it is not a language. it provides following features.

1) Encapsulation
2) Data Abstraction
3) Method overloading
4) Constructors
5) Destructors
6) Operator Overloading
7) Inheritance
8) Method Overriding
9) Polymorphism
10) Templates

powerful features are

1) Encapsulation
2) Data Abstraction
3) Method overloading
4) Constructors
5) Inheritance
6) Polymorphism

Any language which supports all the above powerful features, that language is called as Object Oriented Programming Language.

Eg:

C++, Python (Data Processing), VC++, Java, .Net C# , Scala (Analytics)

History of OOP languages:

C - Language	--> 1972	--> by Dennis Ritchie
C++ and OOP	--> 1981	--> by Bjarne Strou-strups
Python	--> 1989	--> Guido Van Rossum
VC++	--> 1991	--> from Microsoft (it is with GUI Components)
Java	--> 1995	--> James Gosling
.net C#	--> 1998	--> from Microsoft to compete with java
Scala	--> 2009	--> from Apache software foundation

C++ supports all features of OOP's Concept

Java supports only below features	Python Support below features
1) Encapsulation	1. Encapsulation
2) Data Abstraction	2. Data Abstraction
3) Method overloading	3. Method overloading
4) Constructors	4. Constructor
5) Inheritance	5. Inheritance
6) Method Overriding	6. Method Overriding
7) Polymorphism	7. Polymorphism
8)	8. Operator Overloading

Java Features

Encapsulation :

It is a binding (wrapping)of data and methods in a single unit (called as class).Purpose of Encapsulation is to achieve data abstraction.

Data abstraction :

It is a hiding data from the outside environment; it can be achieved through the encapsulation. Purpose of data abstraction is for data security.

Method Overloading :

Class with same methods with different signature.

Constructors:

It is way of initializing an Object with required values

Inheritance:

It an acquiring properties from parent to child. It is use for code reusability.

Polymorphism:

Same one behaving differently for different purpose.

Classes and Objects

Classes and Objects are required to implement Encapsulation and Data Abstraction

class : class is an abstract idea or blue print of some thing. class is a logical one

Defining a class :

```
public class Sample {
        private  int a;
        private  float b; // both are data members (or) instance variables.
        public void setData(){
        -------------------
        -------------------
        }// end of setData method
        public void display() {
        -------------------
        -------------------
        }/ end of display method
} // end of the class
```

Data member :

to store and maintain data of an item and called as field of a record.

Method :

It is a set statements to perform some task on the data Methods are to perform some transaction on data. Methods are to perform CRUD Operations on data.

CRUD : Create, Read , Update and Delete.

class members does not acquire memory with out creating an object.

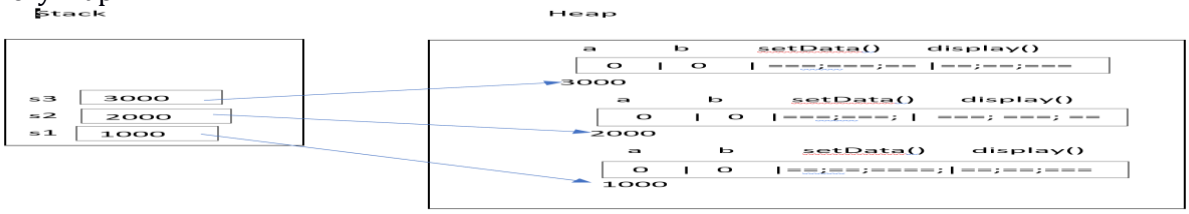
Object :-

It is an instance of the class. When object is created all the members of the class acquires memory. It is collection of data members and methods to manipulate or update the data. It is a physical one.

Creating an object :

```
Sample  s1 ; // Just it is reference of an Object, not an Object
s1 = new Sample(); // it creates an object
Sample s2=new Sample();
Sample s3=new Sample();
```

Memory Map:



Invoking methods :

```
s1.setData();
s2.setData();
s1.display();
s2.display();
```

if any method is invoking through object, then it must be a member of that class.

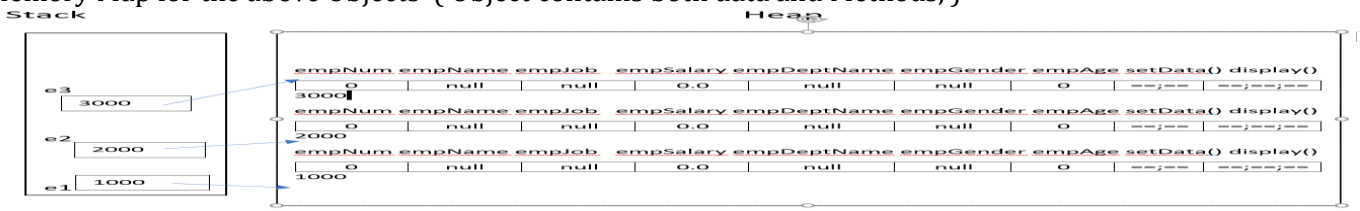
Eg:

```
public class Employee {
 private int empNum; // Data Members
 private String empName; // Data Members
 private String empJob; // Data Members
 private double empSalary; // Data Members
 private String empDeptName; // Data Members
 private String empGender; // Data Members
 private int empAge; // Data Members
 public void setData()  // Method
 {
 }
 public void display()  // Method
 {
 }  }
```

Creating Object :

```
Employee e1 = new Employee();
Employee e2 = new Employee();
Employee e3 = new Employee();
```

Memory Map for the above Objects (Object contains both data and Methods,)



Program to invoke methods:

```
public class Sample {
	public void display() {
		System.out.println("I am in display");
	}   }
public class Test {
	public static void main(String[] args) {
		Sample s1 = new Sample();
		s1.display();
	}
}
```

If any method invoking through object ,that method should me a member of the class.

Eg:

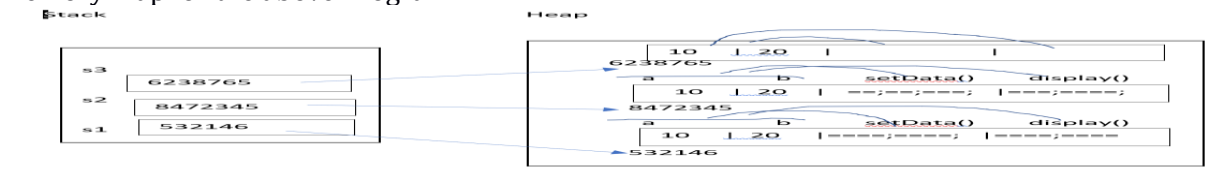
```
package com.nrit.mnrao.test;
public class Sample {
	private int a;
	private int b;
	public void setData(){
		a=10;
		b=20;
```

```
        }
        public void display(){
                System.out.println("a = "+a+"\t b = "+b);
        } }
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData();
                s2.setData();
                s3.setData();
                s1.display();
                s2.display();
                s3.display();
        } }
```

O/P:

a = 10 b = 20 a = 10 b = 20 a = 10 b = 20

Memory Map for the above Program



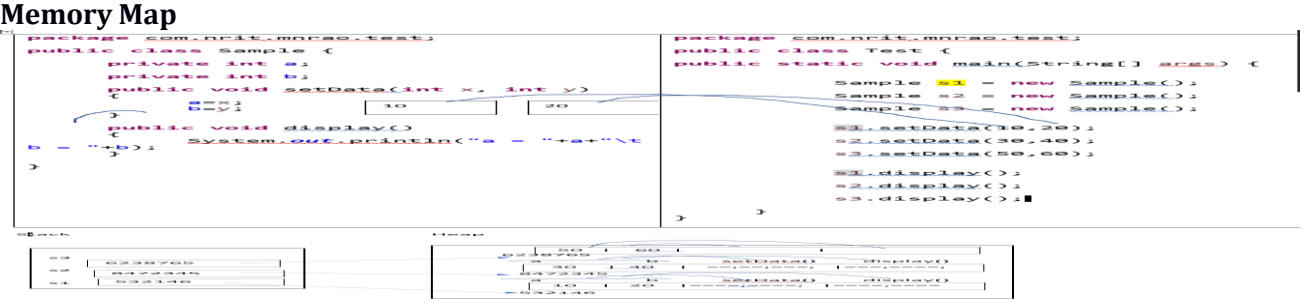

```
public class Sample{
        private int a;
        private float b;
        public void setData(){
                a=10;
                b=30.5f;
        }
        public void display(){
                System.out.println ("a="+a+"\tb="+b);
        } }
public class Test{
        public static void main(String[] args){
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData();
                s2.setData();
                s3.setData();
                s1.display();
                s2.display();
                s3.display();
        } }
```

Class methods with arguments :

```
public class Sample{
        private int a;
        private int b;
        public void setData(int x, int y)       {
                a=x;
                b=y;
}
        public void display()  {
                System.out.println ("a="+a+"\tb="+b);
}   }
public class Test{
```

```
        public static void main(String[] args){
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData(10,20);
                s2.setData(30,40);
                s3.setData(50,60);
                System.out.println ("First Object Details are ");
                s1.display();
                System.out.println ("Second Object Details are ");
                s2.display();
                System.out.println ("Third Object Details are ");
                s3.display();                } }
```

Memory Map



Students information:

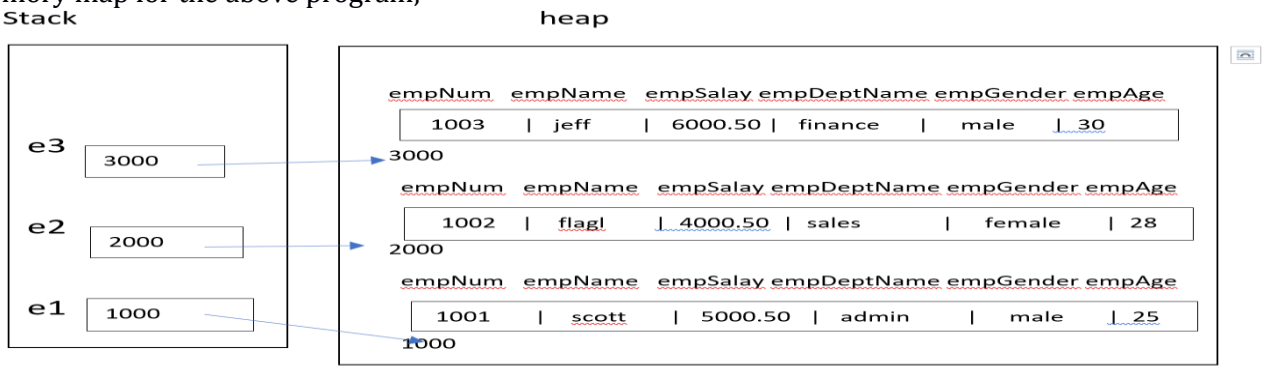
```
public class Student{
        private  int studentId;
        private  String studentName;
        private  int studentAge;
        private  double studentFee;
        public void setData(int id, String name, int age, double fee){
                studentId = id;
                studentName = name;
                studentAge = age;
                studentFee = fee;
        }
        public void display(){
                System.out.println ("STUDENT DETAILS ARE");
                System.out.println ("ID="+studentId);
                System.out.println ("NAME="+studentName);
                System.out.println ("AGE="+studentAge);
                System.out.println ("FEE="+studentFee);
        }  }
public class Test{
        public static void main(String[] args){
                Student st1 = new Student();
                Student st2 = new Student();
                Student st3 = new Student();
                st1.setData(1001,"xyz",23,1500);
                st2.setData(1002,"abc",24,2000);
                st3.setData(1003, "ijk",25,2500);
                st1.display();
                st2.display();
                st3.display();
        }  }
```

Emplyee information:

```
package com.durga.mnrao.test;
public class Employee {
                private int empNum;
                private String empName;
                private double empSalary;
```

```
        private String empDeptName;
        private String empGender;
        private int empAge;
        public void setData(int eno, String ename,  double sal, String  dname, String gender, int age  ){
                empNum = eno;
                empName = ename;
                empSalary = sal;
                empDeptName = dname;
                empGender = gender;
                empAge = age;
        }
    public void display(){
    System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);
    } }
public class Test  {
    public static void main(String[] args)  {
        Employee e1 = new Employee();
        Employee e2 = new Employee();
        Employee e3 = new Employee();
        e1.setData(1001, "Scott", 5000.50, "admin", "male", 25);
        e2.setData(1002, "Flag", 4000.50, "sales", "female", 28);
        e3.setData(1003, "Jeff", 6000.50, "finance", "male", 30);
        System.out.println ("Emp1 details are");
        e1.display();
        System.out.println ("Emp2 details are");
        e2.display();
        System.out.println ("Emp3 details are");
        e3.display();
    } }
```

Memory map for the above program,



How to choose name of the Object :

1) Naming conditions
2) Naming convention

Naming conditions :

These are same for all identifiers

These are as per the compiler.

1) It contains only alphabets (a-z, A-Z), digits (0 – 9) and under score (_)
2) Should not be used spaces and special chars, keywords.
3) Max length can be up to 255 chars

Naming conventions:

These are coding standards

Naming conventions of an Object :

1) name of the object should be same as class name
2) object name should start with lower case alphabet.
3) if object name contains multiple words, then first word start with lower case alphabet and next words start with upper case alphabet

eg:

```
public class ContractEmployee {
```

```
}
```

Object Name :

```
ContractEmployee contractEmployee = new ContractEmployee();
0Real time example :
public class ContractEmployee {
		private int empNum;
		private String empName;
		private String empJob;
		private double empSalary;
		private String empDeptName;
		private String empGender;
		private int empAge;
public void setData(int eno, String ename, String job, double salary,String dname, String gender, int age ){
			empNum = eno;
			empName = ename;
			empJob = job;
			empSalary = salary;
			empDeptName = dname;
			empGender = gender;
			empAge = age;
		}
		public void display(){
System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+em
pAge);
			} }
public class Test {
	public static void main(String[] args) {
ContractEmployee contractEmployee = new ContractEmployee();
contractEmployee.setData(1001,"mnrao", "Architect",606006.60,"it","male",40);
	contractEmployee.display();
	}  }
```

Updating data (Modification of data)

Class with setter methods:

```
public class Employee {
	private int empNum;
	private String empName;
	private String empJob;
	private double empSalary;
	private String empDeptName;
	private String empGender;
	private int empAge;
	public void setEmpNum(int eno){
		empNum = eno;
	}
	public void setEmpName(String ename){
		empName = ename;
	}
	public void setEmpJob(String job){
		empJob = job;
	}
	public void setEmpSalary(double salary){
		empSalary = salary;
	}
	public void setEmpDeptName(String dname){
		empDeptName = dname;
	}
	public void setempGender(String gender){
		empGender = gender;
	}
```

```
	public void setempAge(int age){
		empAge = age;
	}
	public void display(){
	System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"
\t"+empAge);
	}
}
public class Test {
	public static void main(String[] args) {
		Employee employee = new Employee();
		employee.setEmpNum(1001);
		employee.setEmpName("mnrao");
		employee.setEmpJob("manager");
		employee.setEmpSalary(60606.50);
		employee.setEmpDeptName("admin");
		employee.setempGender("male");
		employee.setempAge(35);
		employee.display();
	} }
```

Reading form key board:

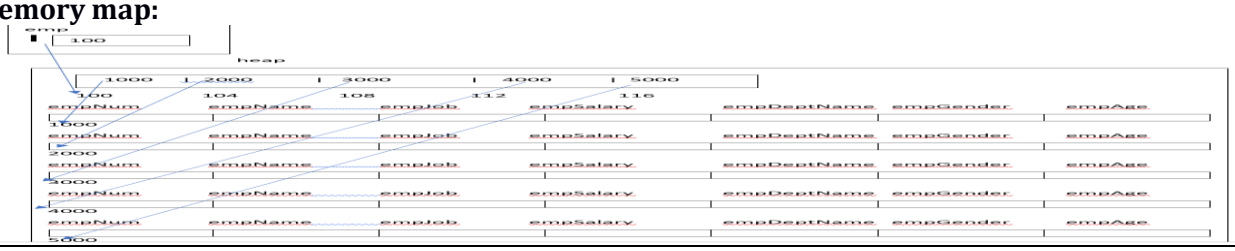
```
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
		Scanner sc = new Scanner( System.in );
		System.out.println("Enter emp number ");
		int eno = sc.nextInt();
		System.out.println("Enter emp name ");
		String ename = sc.next();
		System.out.println("Enter emp job ");
		String job = sc.next();
		System.out.println("Enter emp salary");
		double salary = sc.nextDouble();
		System.out.println("Enter emp dept name ");
		String dname = sc.next();
		System.out.println("Enter emp gender ");
		String gender = sc.next();
		System.out.println("Enter emp age ");
		int age = sc.nextInt();
		Employee employee = new Employee();
		employee.setEmpNum(eno);
		employee.setEmpName(ename);
		employee.setEmpJob(job);
		employee.setEmpSalary(salary);
		employee.setEmpDeptName(dname);
		employee.setempGender(gender);
		employee.setempAge(age);
		employee.display();
	}  }
```

Array of Objects :

```
package com.durga.mnrao.xyz;
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
	Scanner sc = new Scanner(System.in);
	Employee [] emp = new Employee[5]; // creating array of refernces
		// below loop to store data into objects
		for(int i =0; i< emp.length; i++){
			System.out.println("enter emp details ");
```

```
                System.out.println("Enter emp number ");
                int eno = sc.nextInt();
                System.out.println("Enter emp name ");
                String ename = sc.next();
                System.out.println("Enter emp job ");
                String job = sc.next();
                System.out.println("enter emp salary ");
                double salary = sc.nextDouble();
                System.out.println("enter dept name ");
                String dname = sc.next();
                System.out.println("Enter gender ");
                String gender = sc.next();
                System.out.println("Enter age ");
                int age = sc.nextInt();
                emp[i] = new Employee();
                emp[i].setEmpNum(eno);
                emp[i].setEmpName(ename);
                emp[i].setEmpJob(job);
                emp[i].setEmpSalary(salary);
                emp[i].setEmpDeptName(dname);
                emp[i].setempGender(gender);
                emp[i].setEmpAge(age);
        }
        for(int i=0; i<emp.length; i++){
                emp[i].display();
        }   }  }
```

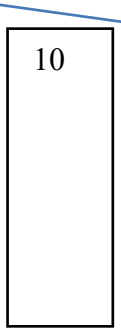
Memory map:



this keyword in java

What is output of Following program, ?

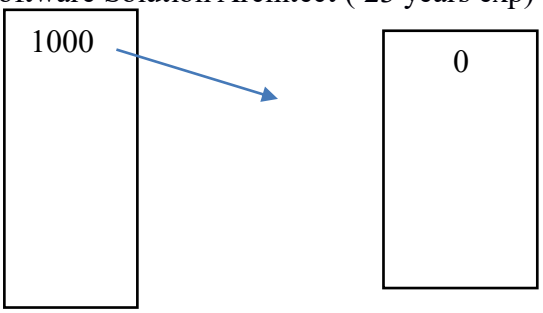
```
public class Sample
{
 private  int  x;
  public void setData(int  x)
 {                                  x
        x  =  x;  //overwriting
 }
  public void display()
 {
        System.out.println (x);
 } }
public class Test {

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



itself , but not assign to instance variable x.

s1 x

```
                Sample s1 = new Sample();
                s1.setData(10);
                s1.display();
        }
}
```



O/p: 0

Using this keyword we can get correct output

What is this in java ?

In java, this is a reference variable (pointer), which refers to the current object. this key word to refer to Current object.
When object is created, for each and every object this pointer generates and stores address of same object. Hence this pointer refer to the current object. **this** reference is also part of Object

Usage of java this keyword

1) this keyword can be used to refer current instance variable.
2) if instance variable name and method argument name are same , in that case , to access instance variable we can use this keyword.
3) if instance variable name and local variable name is same ,in that case also, to access instance variable we can use this keyword.
4) this keyword can also be used to return the current class instance.
5) this() can be used to invoke current class constructor.
6) this keyword can be used to invoke current class method (implicitly)
7) this can be passed as an argument in the method call.

When object is created, for each and every object, this reference variable will be created.
It is a part of object.
Accessing :
This .member;
if class data member name and method argument name are same, to access class data member we can use this keyword.
below example , instance variable name and method argument name are same

```
public class Sample {
 private  int  x;   //  instance variable name
                                  // method argument name
 public void setData(  int   x  )
 {
          this.x = x;
 }
 public void display(){
          System.out.println (x);
 }   }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                s1.setData(10);
                s1.display();
        }  }
```

Another use of this.

if class instance variable name and local variable name is same, to access class data member we can use this keyword.
Below example , instance variable name and local variable name is same

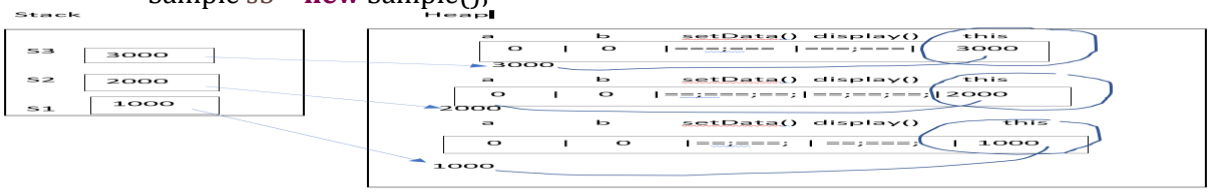
```
public class Sample
{       private  int a;   // instance variable
        public void setData(int  x)
        {
                a = x;
        }
        public void display() {
                int a = 50;  // local variable
                System.out.println ("local a  = " + a);
                System.out.println ("instance variable  a = " + this.a);
        }   }
public class Test {
```

```
        public static void main(String[] args) {
                Sample s1 = new Sample();
                s1.setData(10);
                s1.display();
}   }
```

When object is created, for each and every object this pointer generates and stores the address of same object.
i.e this pointer referring to current Object.

Memory map of this reference variable.

```
public class Sample {
        private  int  a;
        private  int b;
        public void setData()  {
                a=10;
                b=20;
        }
        public void display()  {
                System.out.println ("a="+a+"\tb="+b);
        }   }
        Sample s1 = new Sample();
        Sample s2 = new Sample();
                Sample s3 = new Sample();
```



Program to check address object and it's this reference :

```
public class Sample {
        private int a;
        private int b;
        public void setData(int x, int y){
                a = x;
                b = y;
        }
        public void display(){
                System.out.println("a= "+a+"\t b= "+b);
                System.out.println("this = "+ this);

        }  }
public class Test  {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData(10, 20);
                s2.setData(30, 40);
                s3.setData(50, 60);
           System.out.println(" s1 address = " + s1);
                s1.display();
                System.out.println(" s2 address = " + s2);
                s2.display();
                System.out.println(" s3 address = " + s3);
                s3.display();
        }   }
```

Output for the above :

```
======================;
s1 address = com.durga.mnrao.x.Sample@15db9742
a= 10     b= 20
this = com.durga.mnrao.x.Sample@15db9742
```

```
s2 address = com.durga.mnrao.x.Sample@6d06d69c
a= 30    b= 40
this = com.durga.mnrao.x.Sample@6d06d69c
 s3 address = com.durga.mnrao.x.Sample@7852e922
a= 50    b= 60
this = com.durga.mnrao.x.Sample@7852e922
```

class with setter and getter methods :

```
package com.durga.mnrao.x;
public class Employee {
	private int empNum;
	private String empName;
	private String empJob;
	private double empSalary;
	private String empDeptName;
	private String empGender;
	private int empAge;
	public int getEmpNum() {
		return empNum;
	}
	public void setEmpNum(int empNum) {
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public String getEmpJob() {
		return empJob;
	}
	public void setEmpJob(String empJob) {
		this.empJob = empJob;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpDeptName() {
		return empDeptName;
	}
	public void setEmpDeptName(String empDeptName) {
		this.empDeptName = empDeptName;
	}
	public String getEmpGender() {
		return empGender;
	}
	public void setEmpGender(String empGender) {
		this.empGender = empGender;
	}
	public int getEmpAge() {
		return empAge;
	}
	public void setEmpAge(int empAge) {
		this.empAge = empAge;
	}
}// end of Employee
```

```
package com.durga.mnrao.x;
public class Test {
	public static void main(String[] args) {
		Employee employee = new Employee();
		employee.setEmpNum(1001);
		employee.setEmpName("mnrao");
		employee.setEmpJob("manager");
		employee.setEmpSalary(50505.50);
		employee.setEmpDeptName("admin");
		employee.setEmpGender("male");
		employee.setEmpAge(35);
		int empNum = employee.getEmpNum();
		String empName = employee.getEmpName();
		String empJob = employee.getEmpJob();
		double empSalary = employee.getEmpSalary();
		String empDeptName = employee.getEmpDeptName();
		String empGender = employee.getEmpGender();
		int empAge = employee.getEmpAge();
	System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"
\t"+empAge);		}
}// end of Test class with main()
```

main() with array of Objects (Multiple Objects)

```
package com.durga.mnrao.x;
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
			Scanner sc = new Scanner(System.in);
			Employee [] emp = new Employee[5]; // creating array of references
			// below loop to store data into objects
			for(int i =0; i< emp.length; i++)  {
				System.out.println("enter emp details ");
				System.out.println("Enter emp number ");
				int eno = sc.nextInt();
				System.out.println("Enter emp name ");
				String ename = sc.next();
				System.out.println("Enter emp job ");
				String job = sc.next();
				System.out.println("enter emp salary ");
				double salary = sc.nextDouble();
				System.out.println("enter dept name ");
				String dname = sc.next();
				System.out.println("Enter gender ");
				String gender = sc.next();
				System.out.println("Enter age ");
				int age = sc.nextInt();
				emp[i] = new Employee();
				emp[i].setEmpNum(eno);
				emp[i].setEmpName(ename);
				emp[i].setEmpJob(job);
				emp[i].setEmpSalary(salary);
				emp[i].setEmpDeptName(dname);
				emp[i].setEmpGender(gender);
				emp[i].setEmpAge(age);
			}
			for(int i=0; i<emp.length; i++){
				int empNum = emp[i].getEmpNum();
				String empName = emp[i].getEmpName();
				String empJob = emp[i].getEmpJob();
				double empSalary = emp[i].getEmpSalary();
```

```
                String empDeptName = emp[i].getEmpDeptName();
                String empGender = emp[i].getEmpGender();
                int empAge = emp[i].getEmpAge();
        System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"
\t"+empAge);
                        }  }  }
```

Class with setter and getter methods is called POJO class or Bean class or model class.
POJO Plain Old Java Object.
POJO Core java related.
bean class or model class related to MVC frame works.
M Model , V View and C Controller

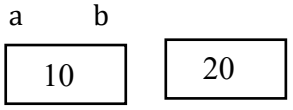
Java-Call by value and Call by reference in Java:

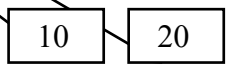
Call by value methods:

it is a calling method by passing value of a variable . in java passing variable to method only call by value for variables don't have call by reference .

Example for call by value:

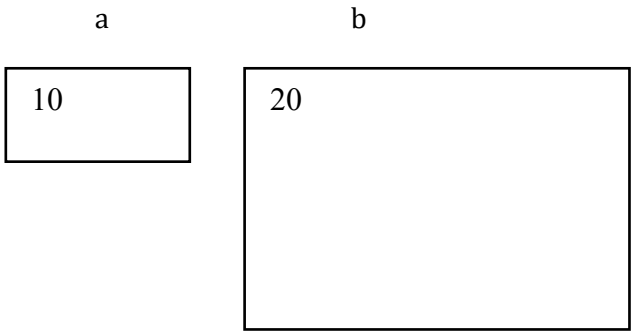
```
public class Test {
        public static void main(String[] args)
        {       System.out.println("main start ");
                int a = 10;
                int b = 20;
                display(a,b);
                System.out.println("main end");
        }
        public static void display(int x, int y)
        {
                System.out.println("x = "+x+"\t y = "+y);
        }  }
```

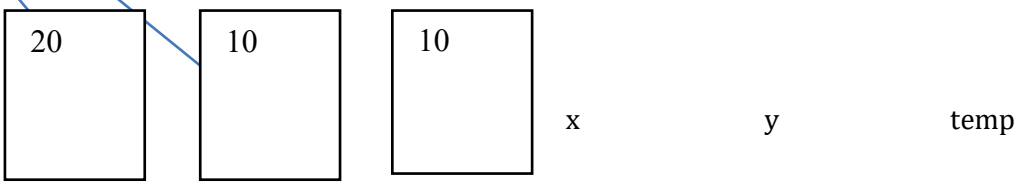




Swapping two variables:

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

                int a=10;
                int b=20;
System.out.println("before call to swap method");
        System.out.println(a+"\t"+b);    10  20
                swap(a,b);
System.out.println("After call to swap method");
        System.out.println(a+"\t"+b);  10  20
        }
        public static void swap(int  x,      int  y)
        {
                int temp;
                temp=x;
                x=y;
                y=temp;
}
```





O/P:

```
before call to swap method
10      20
After call to swap method
10      20
```

Reason :

In call by value, methods arguments are local variables. Hence local copies creates for arguments, if any changes made on the arguments will not be reflected on the actual variables (passing variables) as these are local copies.
In the above example, swapping operation works on local copies x and y.

Call by reference method:

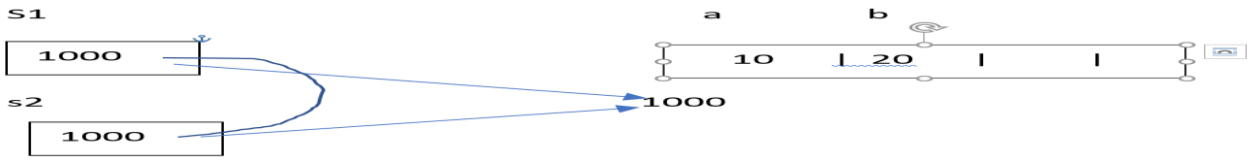
it is a calling method by passing reference of an object. in Java passing object to method is only call by reference for objects don't have call by value.

Before call by reference , example to understand references

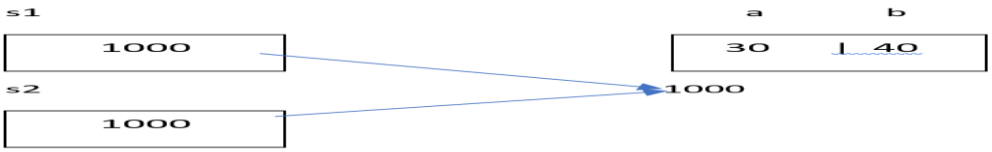
```
public class Sample {
	private int a;
	private int b;
	public void setData(int x, int y)  {
		a=x;
		b=y;
	}
	public void display()  {
		System.out.println("a= "+a+"\t b= "+b);
	}  }
public class Test {
	public static void main(String [] args)  {
		Sample s1 = new Sample();
		s1.setData(10, 20);
		Sample s2 = s1;
		System.out.println("initail values ");
		s1.display();
		s2.display();
		s2.setData(30, 40);
		System.out.println("after changing from s2 object");
		s1.display();
		s2.display();
	}  }
```

initial values
a= 10 b= 20
a= 10 b= 20
after changing from s2 object
a= 30 b= 40
a= 30 b= 40

initail values Memory map



After change from s2 memory map



Output is same for both, since they are referring to same location.

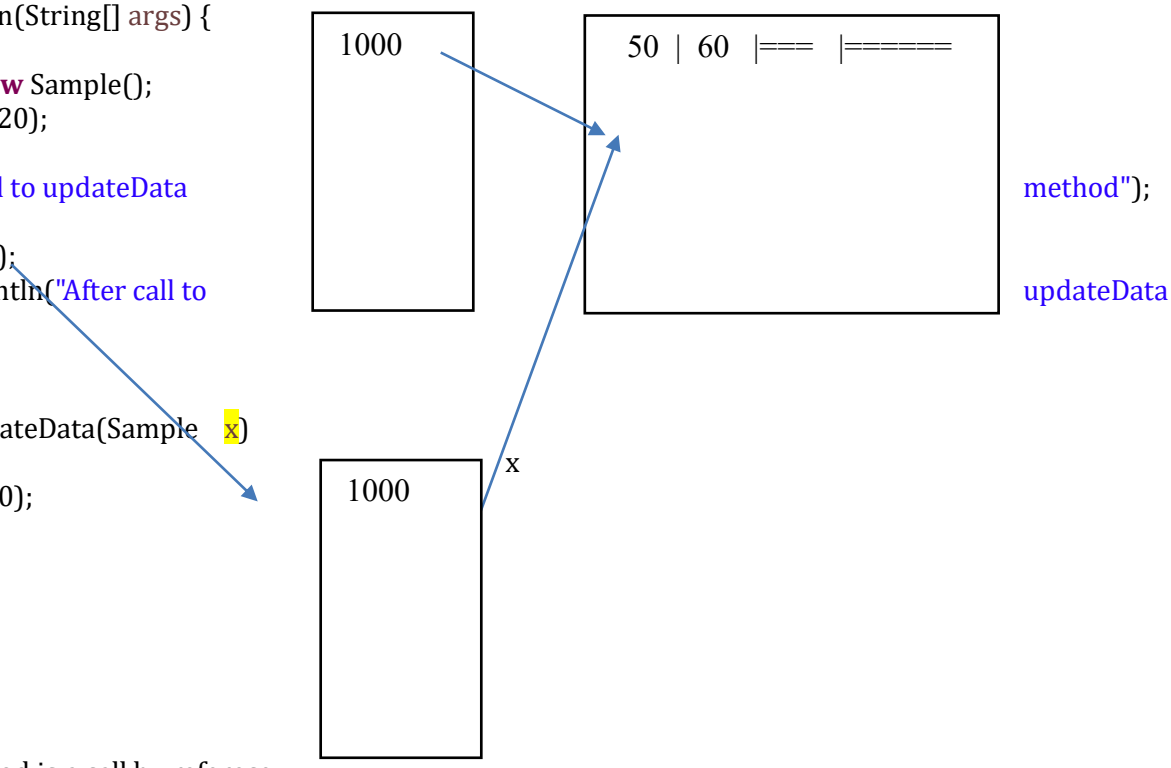
Call by reference example :

Passing Object to method

```
package com.nrit.mnrao.test;
public class Sample {
	private int a;
	private int b;
	public void setData(int x, int y){
		a=x;
		b=y;
	}
	public void display(){
		System.out.println("a = "+a+"b= "+b);
	}
}public  class Test{
```

s1 a b setdata() dis

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

                Sample s1 = new Sample();
                s1.setData(10, 20);
1000
System.out.println("before call to updateData                                    method");
                s1.display();
                updateData(s1);
                System.out.println("After call to                                updateData
method");
                s1.display();
        }
        public static void updateData(Sample   x)
        {
                x.setData(50, 60);
        } }
```



O/P:

```
before call to updateData
a = 10    b= 20
After call to updateData
a = 50    b= 60
```

In java, passing Object to method is a call by referece.
In call by reference, methods arguments refer to actual copy of the Object, if any changes made on the arguments, will reflect on actual copy of the Object .
When passing Object to method, reference of Object will move to method but not Object (Object remains at same location)

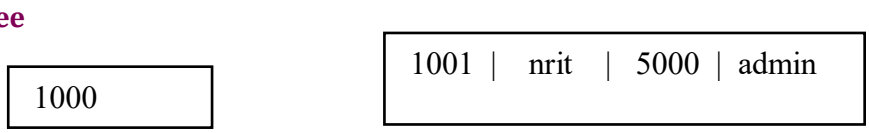
==========

Another eg:

```
public class Employee {
        private int empNum;
        private String empName;
        private double salary;
        private String empDept;
        public int getEmpNum() {
                return empNum;
        }
        public void setEmpNum(int empNum) {
                this.empNum = empNum;
        }
        public String getEmpName() {
                return empName;
        }
        public void setEmpName(String empName) {
                this.empName = empName;
        }
        public double getSalary() {
                return salary;
        }
        public void setSalary(double salary) {
                this.salary = salary;
        }
        public String getEmpDept() {
                return empDept;
        }
        public void setEmpDept(String empDept) {
                this.empDept = empDept;
        } }
```

Heap Memory

empNum empName empSalary empDept



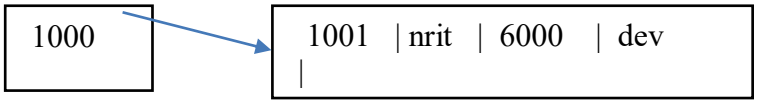

```
public  class Test
{
        public static void main(String[] args) {
        Employee  employee = new Employee();
                employee.setEmpNum(1001);
                employee.setEmpName("nrit");
                employee.setSalary(5000);
                employee.setEmpDept("admin");
                System.out.println("data delivering....");
                sendData( employee );
                System.out.println("data delivered");
        }
        public static void sendData(Employee  emp)
        {
                int empNum = emp.getEmpNum();
                String empName = emp.getEmpName();
                double salary = emp.getSalary();
                String empDept = emp.getEmpDept();
                System.out.println(empNum+"\t"+empName+"\t"+salary+"\t"+empDept);
        }  }
```

1000 emp 1000

empNum empName empSalary empDept

employee

Another eg:




```
public  class Test
{
        public static void main(String[] args) {
                Employee  employee = new Employee();
                System.out.println("data receiving....");
                recvData(employee);
                System.out.println("data received");
                int empNum = employee.getEmpNum();
                String empName = employee.getEmpName();
                double salary = employee.getSalary();
                String empDept = employee.getEmpDept();
                System.out.println(empNum+"\t"+empName+"\t"+salary+"\t"+empDept);
        }
public static void recvData(Employee  emp)
        {
                emp.setEmpNum(1001);
                emp.setEmpName("nrit");
                emp.setSalary(6000);
                emp.setEmpDept("dev");
        }
}
```

1000

emp 1000

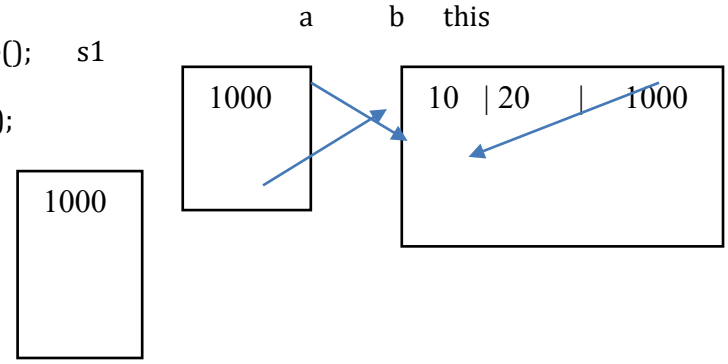
Another Eg:

```
public  class Test{
public static void main(String[] args) {
                Employee  employee = new Employee();
                System.out.println("data receiving....");
                recvData1(employee);
                System.out.println("data received");
                int empNum = employee.getEmpNum();
```

```
                String empName = employee.getEmpName();
                double salary = employee.getSalary();
                String empDept = employee.getEmpDept();
                System.out.println(empNum+"\t"+empName+"\t"+salary+"\t"+empDept);
        }
        public static void recvData1(Employee   employee){
                recvData2(employee);
                employee.setSalary(6000);
                employee.setEmpDept("dev");
        }
        public static void recvData2(Employee  employee){
                employee.setEmpNum(1001);
                employee.setEmpName("nrit");
        }  }
```

Method returning reference to current object:

```
public class Sample {
        private int a;
        private int b;
        public void setData(int x, int y){
                a=x;
                b=y;
        }
         public void  display() {
                System.out.println(a+"\t"+b);
         }
         public Sample  myCopy() {
                 return  this;
         }  }
public class Test {
        public static void main(String[] args) {

                Sample  s1 = new Sample();      s1
                s1.setData(10, 20);
                Sample  s2 = s1.myCopy();
        1000
                s1.display();
                                s2
                s2.display();


        }
}
```



O/p: 10 20
10 20

same for both, since they are referring to same location.

1.What is this keyword?

this keyword provides reference to the current object and it's mostly used to make sure that object variables are used, not the local variables having same name.

2.What is the default value of an object reference declared as an instance variable?

Null, unless it is defined explicitly.

Java static

Static is an access modifier in java

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be used with:

1) static Data member (also known as class variable)
2) static method (also known as class method)
3) static block
4) static inner classes

5) static import.

1) Class Data member as static

If you declare any data member as static, it is known static data member.

The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

The static variable gets memory only once in class area at the time of class loading. Default value of static data member is 0.

All static members of the class acquire memory without creating an object. These are not part of object.

For static data members only one copy will be created and that can be shared by all objects of the class.

It is a sharable memory.

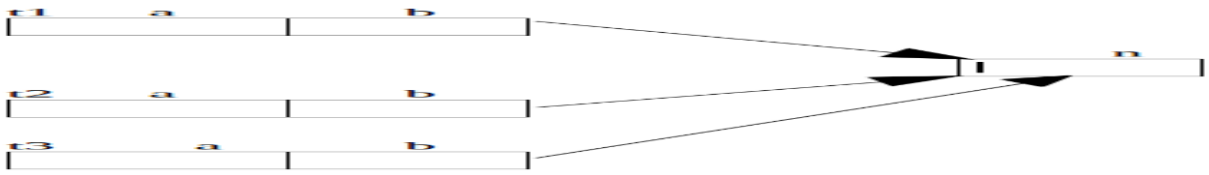
These members can be shared by all objects of the class. These can accessed eighther by class name or through the object.

Eg:

```
class Test  {
int a, b ;
static int n :
}
```

there are three objects created for the above class, for static data member 'n' only one copy is created but for non static member 'a' and 'b' separate copy is created for ever object (three copies).

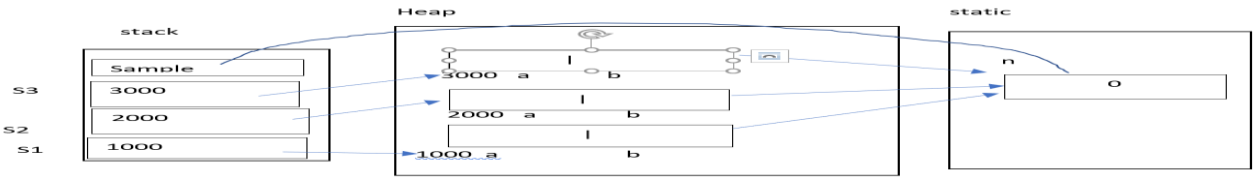
```
Test t1 = new Test();
Test t2 = new Test();
```




```
Test t3 = new Test();
```

```
public class Sample {
        private static int n;
        private  int a;
        private  int b;
}
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
```

Memory map for the above Objects

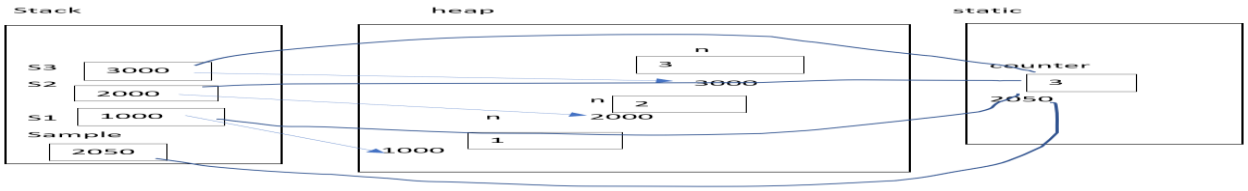


Program to count the number of objects created.

```
public class Sample {
        private  int n;
        private  static int counter;
        public void setData() {
                n = ++counter;
        }
        public void display() {
                System.out.println ("Current Object " + n);
                System.out.println ("Total Objects " + counter);
        } }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
```

```
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData();
                s2.setData();
                s3.setData();
                s1.display();
                s2.display();
                s3.display();
        }   }
```

current Object : 1 Total Objects : 3
current Object : 2 Total Objects : 3
current Object : 3 Total Objects : 3
Memory map for the above program



What is output of the following program.

```
public class Sample {
        private  int n;
        private  int counter;  // static not used
        public void setData()  {
                n=++counter;
        }
        public void display() {
                System.out.println ("Current Object : "+n);
                System.out.println ("Total no of  Objects : "+counter);
        }   }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData();
                s2.setData();
                s3.setData();
                s1.display();
                s2.display();
                s3.display();
        }   }
```

O/P:

Current Object : 1 Total no of Objects : 1
Current Object : 1 Total no of Objects : 1
Current Object : 1 Total no of Objects : 1

All static members of the class acquire memory with reference of class name without creating an object.
These members can be accessed either through object or by the class name. Accessing static data by the name of the class --> class name . member ,
Eg :
in the Test class
Test.n;
Since static data members acquire memory with reference of the class, these are called as class variable.
Static data member is class level. Instance variable is a instance level.

Accessing static data member without object.

```
public class Sample  {
        private  int n;
        public static int counter;
```

```
        public void setData() {
                n=++counter;
        }
        public void display()  {
                System.out.println ("Current Object : "+n);
                System.out.println ("Total no of  Objects : "+counter);
        }   }
public class Test {
        public static void main(String[] args) {
                System.out.println (Sample.counter);
                Sample.counter=Sample.counter+10;
                Sample.counter=Sample.counter+20;
                System.out.println (Sample.counter);
        }   }
```

O/p: 0 30

Types of variables :

there are three types of variables:

1) local variables
2) instance variables
3) class variables

```
public class Sample{
        int a; // instance variable
        static int n; //class variable
        public void display(){
                int  x; // local variable.
        }   }
```

Static data members are used in the following situations/scenarios :

1) To maintain common data for all the instances
2) To provide auto increment in application
3) To act like a counter variable in the application (at the server side)
   eg: to count no. of clients login into a server.

In real time applications data members are private to achieve data security .
To access these private members, public methods are introduced.
Public methods are acting as interface between private data and outside environment.

```
public class Sample {
        private int a;
        private int b;
        public void setData(int x, int y) {
                a = x;
                b = y;
}
        public void display() {
                System.out.println (a + "\t" + b);
        }
}
public class Test {
        public static void main(String[] args) {
                // below data is a public data.
                int a=10;
                int b=20;
                Sample s1 = new Sample();
                // below statement is to make public data as private.
                //here setData() takes the public data and stores into object.
                s1.setData(a,b);
                s1.display();
        }   }
```

Java-Static method :

Static methods also like a static data members, it acquires memory without creating an object. All static members acquires

memory by the name of the class without creating an object.

Declaration of static method :

```
class Test{
        public static void display(){
                =============;
                =============;
        }   }
```

static method acquires memory as soon as class gets loaded into the memory (without any class instance) Static method can be called by the reference of class name or though the object.

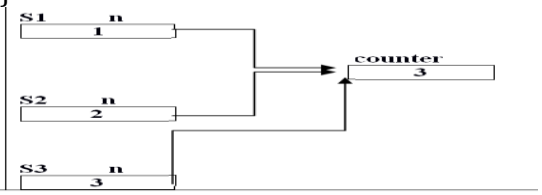
calling by the class name --> Test.display();

```
public class Sample {
        private static int count;
        public static void display(){
                count++;
        System.out.println( count);
        }
}
public class Test {
        public static void main(String[] args) {
                Sample.display();   // invoking static method by class name.
                Sample.display();
                Sample.display();
        }   }
```

main purpose of static method is to access private static data member.

Counting no.of objects created using static method

```
public class Sample {
        private int n;
        private  static int counter;
        public void setData(){
                n = ++counter;
        }
        public void display(){
                System.out.println ("Current Object " + n);
        }
        public static void showCount(){
                System.out.println ("Total Objects " + counter);
        }
}
public class Test{
public static void main(String[] args){
                Sample.showCount();
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3 = new Sample();
                s1.setData();
                s2.setData();
                s3.setData();
                s1.display();
                s2.display();
                s3.display();
                Sample.showCount();
        }
}
```



1) instance method (non-static method)	2) class method (static method)
1. it acquires memory with reference of object. for this method object required 2. it is invoking through the object (instance) instance method can access any data 3. it can make a call to any method , static as well non-static methods.	1. it acquires memory with reference of class name. for this method object is not required 2.it is invoking by the class name static method can access only static data. 3.static method can make a call to only static methods.

Static method can access only static data members and static methods Reason: Static method acquires memory without any object but for non static members object is required. non static method can access any member (static or non static members)

methods two types:

instance method (non-static method) -->

invoking the through the instance and it can access all members

class method (static method)

invoking by the class name and it can access only static members.

This keyword (pointer)

should not be used in the static methods, because static method acquires memory without creating an object but for **"this"** reference object required.

Since static method do not have current object, this key word should not be used with static methods.

```
public class Sample {
	private int a;
	public void setData(int x) {
		a = x;
	}
	public void display() {
		System.out.println(a);
		this.a = 10;
	}
	public static void showData() {
		int a = 50;
		System.out.println(this.a);// invalid
	} }
```

```
package com.durga.mnrao.x;
public class Sample {
	private static int n = 10;
	public static void display(){
		int n = 20;
		System.out.println("local n = "+n);
		System.out.println("class level = a " + Sample.n);
		System.out.println("class level = a " + this.n); // invalid
	}
}
```

different ways of invoking methods :

1) using object instance method (non-static method)
2) using class name class method (static method)
3) directly calling without any object or class name local method, same class method

1) using object:

==========

to invoke non-static method object is required
create an object and call to non-static method
eg:

```
Sample s1 = new Sample();
s1.setData(); //non-static method
s1.display(); //non-static method
```

2) using class name

static methods can be invoked by the class name. if any method is invoking by the class name, then it is a static method.

```
Sample.showCount();  // it is a static method.
```

3) directly calling without any object or class name :

==========

if method is defined in the same class,
then it can be called directly without any object or class name

Java-static import :

If any class imported as static , all of it's static members can access directly with out using class name

Syntax

```
import static <package_name.class_name.*>
```

```
package com.nrit.mnrao.y;
public class Sample {
	public static int a=10;
	public static int b=20;
	public static void display(){
		System.out.println("Sample display");
	}
	public static void show(){
	System.out.println("Sample show");
	}
}
```

```
package com.nrit.mnrao.x;
// below is a static import
import static com.nrit.mnrao.y.Sample.*;
public class Test {
	public static void main(String[] args) {
		System.out.println(a);
		System.out.println(b);
		display();
		show();
	}
}
```

Individual static import

```
package com.nrit.mnrao.x;
import static com.nrit.mnrao.y.Sample.a;
import static com.nrit.mnrao.y.Sample.b;
import static com.nrit.mnrao.y.Sample.display;
import static com.nrit.mnrao.y.Sample.show;
public class Test {
	public static void main(String[] args) {
		System.out.println(a);
		System.out.println(b);
		display();
		show();
	}  }
```

```
//System class importing as satic
package com.nrit.mnrao.x;
import static java.lang.System.*;
import java.util.Scanner;
public class Test {
	public static void main(String[] args) {
	System.out.println("Hello world");
	System.err.println("error");
	Scanner sc = new Scanner(in);
		out.println("enter a number");
		int n = sc.nextInt();
		out.println("your number ="+n);
	}  }
```

Constructors

1. Constructor is a member of the class, it is like a method, which is invoked automatically when object is created.
2. The purpose of the constructor is to initialize objects with required values. Constructor invoking implicitly when object is created.

3. Whereas method to be invoked explicitly by the developer.

Rules to define a constructor :

1) Name of the constructor should be the same as the class name.
2) Constructor must be a public.
3) Constructor should not have any return type not even void also. Implicitly it returns its own type.
4) constructor can have arguments, like a method.
5) constructor can be overloaded, like a method.
6) constructor should not be static, abstract, final, synchronize, transient, native and volatile.
7) constructors can not be Inherited.
8) constructor cannot be overridden.

Types of constructors :

There are two types of constructors in java
1) default constructor --> it is a constructor without any parameters
2) parameterized constructor --> it is a constructor with parameters .

Working with default constructor :

```
public class Sample {
        public Sample(){
                System.out.println ("Constructor called");
        } }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3=null; //it is only reference, object not created.
        }  }
```

new keyword is responsible to create object as well calling a constructor

O/P; Constructor called
Constructor called

Initializing an object :

```
public class Sample {
        private  int a;
        private  float b;
        private  char ch;
        public Sample(){
                a=10;
                b=20.6f;
                ch='a';
        }
        public void display(){
                System.out.println (a);
                System.out.println (b);
                System.out.println (ch);
        }
}
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                Sample s2 = new Sample();
                Sample s3=null;
                s1.display();
                s2.display();
                s3.display();//throws NullPointerException.
        } }
```

Method should not be invoked through the null reference. It throws NullPointerException
default constructors are used to initialize all objects with same values.

Eg: min balance in bank account and min age of employment is same for all. Company name is same for all employees
Another example :

empNum and salary auto initialization.

```
public class Employee {
	private int empNum;
	private String empName;
	private double empSalary;
	private char empGender;
	private static int counter=1000;
	public Employee(){
		empNum=++counter; // assigning empnum automatically
		empSalary=5000;// default salary for any employee.
	}
	public int getEmpNum(){
		return empNum;
	}
	public String getEmpName(){
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSalary(){
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public char getEmpGender() {
		return empGender;
	}
	public void setEmpGender(char empGender) {
		this.empGender = empGender;
	}
	@Override
	public String toString() {
		return "Employee [empNum=" + empNum + ", empName=" + empName + ", empSalary=" + empSalary + ",
empGender="+ empGender + "]";
	} }
public class Test {
	public static void main(String[] args) {
		Employee employee1 = new Employee();
		employee1.setEmpName("nrit1");
		employee1.setEmpGender('M');
		Employee employee2 = new Employee();
		employee2.setEmpName("nrit2");
		employee2.setEmpGender('F');
		Employee employee3 = new Employee();
		employee3.setEmpName("nrit3");
		employee3.setEmpGender('M');
		Employee employee4 = new Employee();
		employee4.setEmpName("nrit4");
		employee4.setEmpGender('F');
		System.out.println(employee1);
		System.out.println(employee2);
		System.out.println(employee3);
		System.out.println(employee4);
	} }
```

Another example:

```
public class DatabaseServer {
	private String dbHost;
```

```
        private String dbName;
        private String dbUid;
        private String dbPasswd;
        public DatabaseServer() {
                dbHost="localhost";
                dbName="nrit";
                dbUid="scott";
                dbPasswd="tiger";
        }
        public String getDbHost() {
                return dbHost;
        }
        public void setDbHost(String dbHost) {
                this.dbHost = dbHost;
        }
        public String getDbName() {
                return dbName;
        }
        public void setDbName(String dbName) {
                this.dbName = dbName;
        }
        public String getDbUid() {
                return dbUid;
        }
        public void setDbUid(String dbUid) {
                this.dbUid = dbUid;
        }
        public String getDbPasswd() {
                return dbPasswd;
        }
        public void setDbPasswd(String dbPasswd) {
                this.dbPasswd = dbPasswd;
        }
        public void display() {
                System.out.println ("Server  :" + getDbHost());
                System.out.println ("Database : " + getDbName());
                System.out.println ("Db user id : " + getDbUid());
                System.out.println ("Db password " + getDbPasswd());
        }   }
public class Test {
        public static void main(String[] args) {
                DatabaseServer db = new DatabaseServer();
                System.out.println ("Connecting to server:");
                db.display();
                db.setDbHost("192.10.20.1");
                db.display();
                db.setDbUid("java");
                db.setDbPasswd("java123");
                db.display();
                System.out.println ("currecnt uid:"+ db.getDbUid() );
                System.out.println ("currecnt server location :"+ db.getDbHost() );
        }   }
```

Working with parameterized constructor :

```
public class Sample {
        private  int a;
        private  int b;
        public Sample() {
                a = 10;
                b = 20;
```

```
        }
        public Sample(int x, int y) {
                a = x;
                b = y;
        }
        public void display() {
                System.out.println (a);
                System.out.println (b);
        }   }
public class Test {
        public static void main(String[] args) {
                Sample s1= new Sample();
                Sample s2= new Sample(40,50);
                Sample s3= new Sample(70,80);
                s1.display();
                s2.display();
                s3.display();
        }   }
```

parameterized constructor is used to initialize the object with different values .

Class with both default and parameterized constructors.

```
class Test  {
 private  int a, b ;
        public Test() {
                a=0;
                b=0;
        }
        public Test(int x, int y) {
                a=x;
                b=y;
        }
        void display() {
                System.out.println ( " a= " + a + " b= " +b);
        }
}
public class MultiConstructorTest {
        public static void main( String [] args ) {
        Test t1 = new Test(); // if default constructor is not defined then it is a compilation error.
        Test t2 = new Test( 10,15);
                 t1.display();
                t2.display();
        }   }
```

1. if class don't have constructors, then compiler generates a default constructor for every class.
2. If class contains at least one user defined constructor, then compiler will not generate default constructor.
3. If class contains constructors then one of the constructors must be a default constructor.
4. when you defining a class with constructors, then one of the constructors must be a default constructor.

Eg:

```
public class Sample {
        private int a;
        private int b;
        public Sample(int x, int y){
                a=x;
                b=y;
        }
                public void display(){
                System.out.println(a+"\t"+b);
        }    }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample(); // compile time error as compiler not generating any default constructor.
                Sample s2 = new Sample(10,20);
```

```
        Sample s3 = new Sample(50,60);
        s1.display();
        s2.display();
        s3.display();
    }  }
```

Constructor overloading :

This is similar to method overloading, it takes place based no. of arguments and type of arguments. At compile time, compiler first check for the no. of args , if no. of args are matched then checks for the type of args, if type is also matched between constructors then ambiguity between the constructors and compilation error.

Overloading is based on number of arguments.

```
public class Test {
    private  int a;
    private  int b;
    private  int c ;
    public Test() {
        a=0;
        b=0;
        c=0;     }
    public Test(int x) {
        a=x;
        b=0;
        c=0;
    }
    public Test(int x, int y) {
        a=x;
        b=y;
        c=0;
    }
    public Test(int x, int y, int z) {
        a=x;
        b=y;
        c=z;
    }
    void display() {
        System.out.println ( " a= " + a + " b= " +b+" c= " +c);
    }  }
public class ConstructorOverLoadTest {
    public static void main( String [] args ) {
        Test t1 = new Test();
        Test t2 = new Test( 10, 15);
        Test t3 = new Test(1,2,3);
         t1.display();
        t2.display();
        t3.display();
    }  }
```

Constructor Overloading based on type of arguments

```
class Test {
    private  int a;
    private  float b;
    private  char ch;
    public Test() {
        a=0;
        b=0.0f;
        ch=0;
    }
    public Test(int x, float y ) {
        a=x;
        b=y;
```

```
        }
        public Test(int x, char y) {
                a=x;
                ch=y;
        }
        public Test(float x, char y) {
                b=x;
                ch=y;
        }
        void display() {
                System.out.println (" a= "+a"+ b= "+b+" ch= "+ch );
        }
}// end of class Test
public class ConstructorOverLoadTypeTest {
        public static void main(String [] args ) {
                Test t1 = new Test();
                 int i=10;
                float f = 11.5f;
                char c1 = 'a';
                Test t2 = new Test(i,f);
                Test t3 = new Test(f,c1);
                t1.display();
                t2.display();
                t3.display();
        }  }
```

An Object can be initialized in three ways

1) Class Level
2) Initialize block / instance block
3) Constructor

class level initialization :

```
class Test {
        private  int a = 10;
        private  float b = 15.5f;
public test()
{}
}
Test t1 = new Test();
```

Initialize / instance block :

```
class Test {
{
          ======;
          ======;
}
}
```

Initialize block / instance block executes before going to execute a constructor.

```
public class Sample   {
        {
                System.out.println ("Initialize block");
        }
        public Sample()
        {
                System.out.println ("constructor");
        }  }
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
        }
}
```

Class with multiple initialize blocks.

```
public class Sample  {
        {
                System.out.println ("Initialize block1");
        }
        public Sample(){
                System.out.println ("constructor");
        }
        {
                System.out.println ("Initialize block2");
        }
        public void display(){
                System.out.println ("I am in display");
        }
        {
                System.out.println ("Initialize block3");
        }
}
public class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
        }
}
```

Initialize block executes for every Object.

There is no order of writing initialize blocks, it can be written anywhere in the class. A class can have multiple initialize blocks If class contains multiple initialize blocks then order of execution is top to bottom. Purpose of initialize block is to initialize constants; Purpose of Multiple initialize blocks are to segregate constants for easy understanding code (as per coding standards) Segregate grouping.

Static block :

```
class Test {
        static {
                ========;
                ========;
        }}
class Test {
        static
        {
                System.out.println ( " Static block ");
        }
        {
                System.out.println ( " Init block ");
        }
        public Test() {
                System.out.println (" Constructor  ");
        }
}
public class StaticInitBlockTest {
        public static void main( String [] arg ) {
                Test t1 = new Test();
                Test t2 = new Test();
        }
}
```

O/p :Static block. Init block constructor init block constructor

Static block is executed only for the first time created object. Because it is common for all the objects.

Static block can be defined any where in the class. A class can have multiple static blocks. The order of execution is from top to bottom.

The purpose of static block is to initialize the static constants. Purpose multiple static blocks are to segregate multiple static constants.

Invoking current class constructor:

```
public class Sample {
```

```
        public Sample(){
                System.out.println("default constructor");
        }
        public Sample(int x, int y){
                this();
                System.out.println("two args");
        }
        public Sample(int x, int y, int z){
                this(10,20);
                System.out.println("three args");
        } }
public class Test {
        public static void main(String[] args){
                Sample s1 = new Sample(1,2,3);
        } }
```

O/P: default constructor two args three args

the statement, which makes a call to another constructor must be 1st statement inside the constructor.

```
public class Sample {
        public Sample(){
                System.out.println("default constructor");
        }
        public Sample(int x, int y){
                System.out.println("two args");
                this();// invalid compile time error.
        }
        public Sample(int x, int y, int z){
                System.out.println("three args");
                this(10,20);// invalid compile time error.
        }
}
```

Method can be a recursive but constructor can not be called recursively.

```
public class Sample {
        public Sample(){
                System.out.println("default constructor");
        }
        public Sample(int x, int y){
                System.out.println("two args");
        }
        public Sample(int x, int y, int z)
        {
                this(10,20,30); // Recursive caling is not valid for constructors.
                System.out.println("three args");
        }    }
```

Constructor can make a call to method, (**but)** method can not make a call to constructor

```
public class Sample {
        public Sample(){
                System.out.println("no args");
        }
        public Sample(int x ){
                this();
                System.out.println("One args");
        }
        public Sample(int x , int y){
                this(x);
                System.out.println("Two args");
        }
        public Sample(int x , int y, int z){
            display()// this is valid
                System.out.println("three args");
```

```
    }
    public void display(){
        this(x,y,z);   // this is invalid .
        System.out.println("I am in display");
    }  }
```

1. When the constructor of a class is invoked?

Ans: The constructor of a class is invoked every time an object is created with new keyword.

2. Can a class have multiple constructors?

Ans: Yes, a class can have multiple constructors with different parameters. Which constructor gets used for object creation depends on the arguments passed while creating the objects.

3. How objects of a class are created if no constructor is defined in the class?

Ans: Even if no explicit constructor is defined in a java class, objects get created successfully as a default constructor is implicitly used for object creation. This constructor has no parameters.

4. Can we call the constructor of a class more than once for an object?

Ans: Constructor is called automatically when we create an object using new keyword. It's called only once for an object at the time of object creation and hence, we can't invoke the constructor again for an object after its creation.

5. Can we have two methods in a class with the same name?

Ans: We can define two methods in a class with the same name but with different number/type of parameters. Which method is to get invoked will depend upon the parameters passed.

Inheritance

It is an acquiring properties from parent into child.

Def. :

It is a creation of new classes from existing classes. It as an acquiring old class properties into new class.

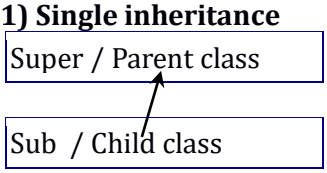
Old class is a parent class and new class is child class. Purpose of Inheritance is code reusability. As per Java , Parent class is called as super class And child class is called as sub class.

Types of inheritance :

Java provides three types of inheritance:

1. Single inheritance
2. Multi level inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance
6. Multipath Inheritance
7. Cyclic Inheritance

1) Single inheritance



extends is a keyword to create child class from the parent class :

Syntax :

```
class Parent {
=============;
}
class Child extends Parent {
=============;
}
```

A class can extend from only one class, since java does not support multiple inheritance Dis-advantage of multiple inheritance, chance creating duplicate copies at the child class level.

```
public class Sample {
     int a;
     int b;
    public void setAB(int x, int y){
        a=x;
        b=y;
    }
    public void dispAB(){
        System.out.println (a+"\t"+b);
```

```
        }
}
public class Test extends Sample {
        int c;
        int d;
        public void setData(int x, int y, int z, int k){
                setAB(x,y);
                c=z;
                d=k;
        }
        public void display(){
                dispAB();
                System.out.println (c+"\t"+d);
        }
}
public class InheritTest {
        public static void main(String[] args) {
                Test t1 = new Test();
                t1.setData(10,20,30,40);
                t1.display();
        }
}
```

Super class and sub class with same methods :

Parent class and child class contains same methods with same signature then method Overriding takes place.

Method overriding is a redefining parent class method in the child class. Purpose of method overriding is to changing or extending functionality of existing methods (parent class methods)

```
public class Sample {
        public void display(){
                System.out.println ("Sample");
        }
}
public class Test extends Sample {
        public void display(){
                System.out.println ("Test");
        }
}
public class InheritTest {
        public static void main(String[] args) {
                Test t1 = new Test();
                t1.display();
        }
}
```

o/p : Test

reason, method overriding .

Invoking super class method : Super is a key word to invoke parent class methods.

```
public class Sample {
        public void display(){
                System.out.println ("Sample");
        }
}
public class Test extends Sample {
        public void display(){
                super.display(); // to invoke parent method.
                System.out.println ("Test");
        }
}
public class InheritTest {
        public static void main(String[] args) {
                Test t1 = new Test();
```

```
                t1.display();
        }
}
```

Super class method can be called from anywhere from the child class Method.

```
        public void display(){
                System.out.println ("Test");
                super.display();
        }
```

Another example :

Invoking current class methods and parent class methods.

```
public class Sample {
        public void show() {
                System.out.println("Sample show");
        }
        public void display() {
                System.out.println("Sample display");
        }
}
public class Test extends Sample {
        public void show() {
                System.out.println("Test show");
        }
        public void display() {
                super.show();// to call to parent class method
                this.show();// to call to same class method
                show();// to call to same class method
                System.out.println("Test display");
        }
}
public class IheritTest {
        public static void main(String[] args) {
                Test  t = new Test();
                t.display();
        }
}
```

Another example

```
public class Sample {
         int a;
         int b;
        public void setData(int x, int y){
                a=x;
                b=y;
        }
        public void display(){
                System.out.println (a+"\t"+b);
        }
}
public class Test extends Sample {
         int c;
         int d;
        public void setData(int x, int y, int z, int k){
                setData(x,y);     // in this case super key word not required as the signature  //is not matching
                c=z;
                d=k;
        }
        public void display(){
                super.display();
                System.out.println (c+"\t"+d);
        }  }
```

Method Overloading	Method Overriding
Passing of same message for different functionality	Redefining parent class method in child class
It is between the same methods with different signature	It is between the same methods with same signature
Method overloading is in the same class	Method overriding is between parent and child classes
It doesn't check for the return type	It checks for the return type
It is a static binding (compile time)	It is dynamic binding (run time)

```
public class InheritTest {
        public static void main(String[] args) {
                Test t1 = new Test();
                t1.setData(1,2,3,4);
                t1.display();
        }   }
```

Difference between method overloading and method overriding :

```
class Test {
        void display() {
                System.out.println ( " Test display " );
        }
}
class TestOne extends Test {
        void display() {
                // display(); it is recursive calling, calling itself, it thorws StackOverFlowException
                super.display();
                System.out.println ( " TestOne display " );
        }
}
public class SuperMethodTest {
        public static void main ( String [] args ){
                TestOne t1 = new TestOne();
                  t1display();
        }
}
```

Super class and sub class with same data members :

```
class Test {
        int a;
        int b;
}
class TestOne extends Test{
        int a;
        int b;
        void setData(){
                super.a=10;
                super.b=20;
                a=30;
                b=40;
        }
        void display(){
                System.out.println ( " Super class : a =  "+super.a+" b = " +super.b );
                System.out.println ( " Sub class : a =  "+a+" b = " +b );
        }
}
public class SuperDataTest {
        public static void main ( String [] args ){
                TestOne t1 = new TestOne();
                  t1.setData();
                  t1.display();
        }
}
```

Directly we can't access super class data members using object, if super class and sub class has same data members.

t.super.a --> is not valid.

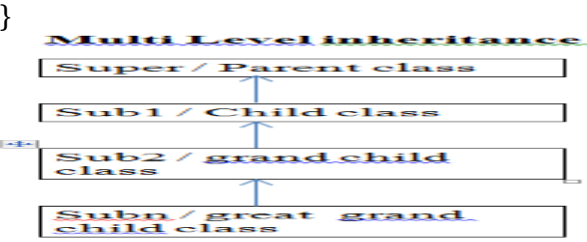
```
class Person {
        int id;
        int age ;
        char sex;
        void setData(){
           pid=1001;
           age = 25;
           sex = 'M"
        }
        void display(){
                System.out.println ( " Id : " +pid );
                System.out.println ( " Age : "+age);
                System.out.println ( "Sex : " +sex);
        }
}
class Emp extends Person{
        float salary;
        void setData() {
                super.setData();
                salary = 20000.0f
        }
        void display() {
                super.display();
                System.out.println ( " salary : "+sal );
        }
}
public class EmpPersonTest {
        public static void main( String [] args ) {
                Emp e1 = new Emp();
                  e1.setData();
                  e1.display();
        }
}
```

Private access specifier.

Private members can be accessed in the same class only . Can not be accessed from sub / child class.

```
public class Sample {
        private  int a;
        private int b;
        public void setData(int x, int y){
                a=x;
                b=y;
        }
        public void display(){
                System.out.println(a+"\t"+b);
        }
}
public class Test extends Sample{
        private int a;
        private int b;
        public void setData(int x, int y , int z, int k){
                setData(x,y);
                //super.a=x; invalid, since these are private members
                //super.b=y; invalid, since these are private members
                a=z;  //its own members
                b=k; //its own members
        }
        public void display(){
                super.display();
```

```
                System.out.println(a+"\t"+b);
        }
}
```



Multi Level Inheritance Example:

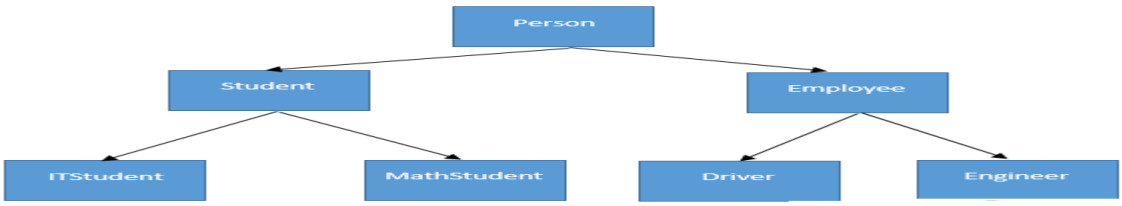
```
public class Test{
        public void display(){
                System.out.println ("Test display");
        }
}
public class TestOne extends Test{
        public void display(){
                System.out.println ("TestOne display begining");
                super.display();
                System.out.println ("TestOne display end");
        }
}
public class TestTwo extends TestOne{
        public void display(){
                System.out.println ("TestTwo Display Beg");
                super.display();
                System.out.println ("TestTwo Display end");
        }
}
public class InheritanceTest {
        public static void main(String[] args) {
                TestTwo t = new TestTwo();
                t.display();
        }
}
```

Multilevel Inheritance with data members

```
public class Test {
        private  int a;
        private  int b;
        public void setData(int x, int y){
                a=x;
                b=y;     }
        public void display()   {
                System.out.println(a+"\t"+b);
        }   }
public class TestOne extends Test {
        private  int c;
        private  int d;
        public void setData(int x, int y, int z, int k ) {
                setData(x,y);
                c=z;
                d=k;
 }
        public void display()   {
                super.display();
                System.out.println(c+"\t"+d);   }  }
public class TestTwo extends TestOne {
        private   int i;
```

```
        private   int j;
        public void setData(int x, int y, int z, int k, int l, int m){
                setData(x,y,z,k);
                i=l;
                j=m;      }
        public void display()   {
                super.display();
                System.out.println(i+"\t"+j);
        }}
        public class MultiLevelTest {
        public static void main(String[] args) {
                TestTwo t = new TestTwo();
                t.setData(10, 20);
                t.display();
                System.out.println("================");
                t.setData(1, 2, 3, 4);
                t.display();
                System.out.println("================");
                t.setData(5, 15, 25, 35, 45, 55);
                t.display(); }  }
```

Hierarchial Inheritance:



eg:

```
package com.nrit.mnrao.hierarchial;
public class Person {
        private int pid;
        private String name;
        private String gender;
        private int age;
        public void setData(int  pid, String  name, String  gender, int age) {
                this.pid = pid;
                this.name = name;
                this.gender = gender;
                this.age = age;
        }
        public void display() {
                System.out.print(pid + "\t" + name + "\t" + gender + "\t" + age+"\t");
        }    }
package com.nrit.mnrao.hierarchial;
public class Student extends Person {
        private String course;
        private  double feePaid;
        private  double feeDue;
        private  char grade;
public void setData(int pid, String name, String gender, int age, String course, double feePaid, double feeDue, char grade) {
                setData(pid, name, gender, age);
                this.course = course;
                this.feePaid = feePaid;
                this.feeDue = feeDue;
                this.grade = grade;
        }
        public void display() {
                super.display();
```

```
                System.out.println(course + "\t" + feePaid + "\t" + feeDue + "\t" + grade);
        }
}
package com.nrit.mnrao.hierarchial;
public class Employee extends Person {
        private String dept;
        private double salary;
        private String desg;
        public void setData(int pid, String name, String gender, int age, String dept, double salary, String desg) {
                setData(pid, name, gender, age);
                this.dept = dept;
                this.salary = salary;
                this.desg = desg;
        }
        public void display() {
                super.display();
                System.out.println(dept + "\t" + salary + "\t" + desg);
        }    }
package com.nrit.mnrao.hierarchial;
public class HierachialTest {
        public static void main(String[] args) {
                Student student = new Student();
                student.setData(1001, "abc", "male", 25, "java", 10000, 5000, 'A');
                student.display();
                Employee employee = new Employee();
                employee.setData(101, "Mohan", "Male", 40, "IT", 25000, "admin");
                employee.display();
        }  }
```

if method names are different then we can call from main method by using object.

Inheritance with constructors:

Constructors invokes from **bottom to top BUT** executes from **top to bottom.**

Default constructors are invoked automatically **(implicitly)**

```
public class Test{
public Test(){
                System.out.println ("Test");
        }  }
public class TestOne extends Test{
        public TestOne(){
                System.out.println ("Test One");
        } }
public class TestTwo extends TestOne{
        public TestTwo(){
                System.out.println ("Test Two");
        }  }
public class InheritConstructorTest{
public static void main(String[] args) {
                TestTwo t = new TestTwo();
        } }
```

o/p:

Test Test One Test Two

Invoking parent class constructors with parameters.

```
public class Test {
        int a;
        int b;
        public Test(){
                a=0;
                b=0;
        }
        public Test(int x, int y){
```

```
                a=x;
                b=y;
        }
        public void display(){
                System.out.println(a+"\t"+b);
        }
}
public class TestOne extends Test {
        int c;
        int d;
        public TestOne(){
                c=0;
                d=0;
        }
        public TestOne(int x, int y, int z, int k){
                super(x,y);      // invoking parent class constructor
                c=z;
                d=k;
        }
        public void display(){
                super.display();
                System.out.println(c+"\t"+d);
        }
}
public class TestTwo extends TestOne {
        int i;
        int j;
        public TestTwo(){
                i=0;
                j=0;
        }
        public TestTwo(int x, int y, int z, int k, int l, int m){
                super(x,y,z,k);  // invoking parent class constructor
                i=l;
                j=m;
        }
        public void display(){
                super.display();
                System.out.println(i+"\t"+j);
        }
}
public class MultiLevelTest {
        public static void main(String[] args) {
                TestTwo t1 = new TestTwo();
                t1.display();
                TestTwo t2 = new TestTwo(1,2,3,4,5,6);
                t2.display();
        } }
```

The statement, which makes a call to parent class constructor must be a first statement in the child class constructor.

Calling same class constructors

```
public class TestOne extends Test{
        private  int c;
        private  int d;
        public TestOne(){
                c=0;
                d=0;
        }
        public TestOne(int x, int y){
                c=x;
```

```
                d=y;
        }
        public TestOne(int x, int y, int z, int k){
                //calling a same class constructor.
                this(z,k);
                a=x;
                b=y;
        }
        public void display(){
                super.display();
                System.out.println (c+"\t"+d);
        }
}
```

Invoking Parent class constructor as well own constructor :

```
public class TestTwo extends TestOne{
        private int i;
        private int j;
        public TestTwo(){
                i = 0;
                j = 0;
        }
        public TestTwo(int x, int y, int z, int k){
                System.out.println("Test Two Four args");
        }
        public TestTwo(int x, int y, int z, int k, int m, int n){
                this(x,y,z,k);
                super(x,y,z,k); // this is invalid, it must be a first statement
                i = m;
                j = n;
        }
        public void display(){
                super.display();
                System.out.println("i = "+i+"\t j = "+j);
        }
}
```

For the above, alternative soultions .

```
public class TestTwo extends TestOne{
        private int i;
        private int j;
        public TestTwo(){
                i = 0;
                j = 0;
        }
        public TestTwo(int x, int y, int z, int k){
                super(x,y,z,k);
                System.out.println("Test Two Four args");
        }
        public TestTwo(int x, int y, int z, int k, int m, int n){
                this(x,y,z,k);  // same class constructor.
                i = m;
                j = n;
        }
        public void display(){
                super.display();
                System.out.println("i = "+i+"\t j = "+j);
        }
}
```

Access Specifiers

It defines scope of classes and its members. There are four types of access specifiers public, protected, default and private. Default access specifier is default Public is having more scope;

Descending order is

public, protected, default, private

Scopes :

1) Outside the Java application world 2) From other packages (sub class)

3) Same package (sub class) 4) Within the same class

Scope ▲ / Access Specifier ▼	Outside the Java application world	From other packages (sub class)	Same package (sub class)	Within the same class
public	Yes	Yes	Yes	Yes
protected	No	Yes	Yes	Yes
default	No	N o	Yes	Yes
private	No	N o	No	Yes

Other package sub class:

==========

Same Package sub class:

```
package com.igrow.mnrao.x;

public class Sample
{
        public int a;
        protected int b;
        int c;
        private int d;
}
```

```
package com.igrow.mnrao.x;

public class Test extends Sample{
        public void display()
        {
                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
                System.out.println(d);
        }
}
```

public is recommanded for **static data** members, as it is using by the class name. to access private members, public methods are required . For every private member , public setter and getter methods are required .

Setter method is to store data into object. Getter method is to retrieve data from the object.

With in the same class:

```
public class Sample
{
        public int a;
        protected int b;
        int c;
        private int d;
        public void display()
        {
                System.out.println(a);  ---> valid
                System.out.println(b);  ---> valid
                System.out.println(c);  ---> valid
                System.out.println(d);  ---> valid
        }
}
```

Access Modifiers

It defines behaviour of classes and it's members.

1) static 2) final 3) abstract 4) synchronize. 5) transient. 6) native 7) volatile

Static:

static can be used with following

1) class data members.
2) methods
3) static blocks.
4) static Inner classes or Nested classes.
5) static import

final :

It can be used with following

1. classes.
2. class data members (instance variable / class variable)
3. methods
4. local variables

abstract:

It can be used with classes and methods.

Final:

local variable as final (constant):

this is to declare local variable as constant. it's value can not be changed. it takes only one time assignment.

Naming conventions :

1) name of the constant must be in upper case alphabets
2) it constant contains multiple words, then every word should be separated with underscore

```
eg:    public class Sample {
       public void display(){
              final int MIN_BUFF_SIZE = 100;
              final int MAX_BUFF_SIZE = 1000;
       }   }
```

Eg1:

```
public class Sample{
       public void display(){
              //below is the local variable.
              final int A=10;
              A=20;// invalid, since A is a constant.
       } }
```

eg2:

```
public class Sample{
       public void display(){
              final int A;
              A=20;// valid sinced A is not initialized.
              A=30;// invalid, final variable doesn't take second time assignment.
       }   }
```

class data member as final:

```
public class Sample{
       final int A=10;
       public void display(){
              A=20;//invalid,final data member can not be modified from the methods.
              System.out.println(A); // valid, since just it is a reading
       }    }
```

Final data member must be initialized. There are three ways to initialize final data member

1) class level
2) using constructor
3) using initialize block.

Class Level:

```
public class Sample{
       final int A=10;
}
```

Below is not valid one. Since method may be invoking many times.

```
public class Sample {
       final private  int A;
       public void setData(){
              A=10;
       }
}
```

using initialize block / instance block.

```
public class Sample {
       final private  int A ;
       {
       A = 10;
```

```
        }   }
public class Sample {
        final private int A;
        {
                A=10;
                A=20;  // invalid , as final variable does not take second time assignment
        }
}
public class Sample {
        final private  int A = 10;
        {
                A = 20;   // invalid, second time assignment
        }
}
```

Using constructor:

eg1:

```
public class Sample {
        final private  int A ;
        public Sample(){
        A = 10;
        }
}
```

Eg2:

```
public class Sample {
        final private int A;
        public Sample(){
                A=10;
                A=20;  // invalid, as it is a second time assignment
        }   }
public class Sample{
        final int A=10;
        public Sample(){
                A=20;// Invalid as final data member takes only one time assignment.
        }
}
```

Using both initialize block and constructor.

```
public class Sample {
        final int A=5;{
                A=10; // not valid
        }
        public Sample(){
                A=10; // not valid
        }   }
public class Sample{
        final int A;{
                A=10;//valid as it is a first time assignment.
                // valid since initialize block executes first it is valid.
        }
        public Sample(){
                A=20;//Invalid as final data member takes only one time assignment.
                //invalid since constructor executes after initialize block
        }   }
```

In the above first initialize bock executes and then constructor.

```
public class Sample {
        final int A;
        {
                A=10;
        }
        {
```

```
                A=20;// Invalid
        } }
```

Purpose of constructor is to initialize **variable data members** (its value can be changed)
Purpose of initialize block is used to initialize **final data members** (its value can not be changed)

```
public class DataBase {
        private String hostName;
        private String userId;
        priv-ate String password;
        final private String DB_PRODUCT;
        final private int PORT_NUM;
        // initialize block for constants .
        {
                DB_PRODUCT = "mysql";
                PORT_NUM = 3306;
        }
        // constructors for varaiable
        public DataBase(){
                 hostName ="192.168.56.10";
                 userId = "root";
                 password = "admin";
        }
        public String getHostName() {
                return hostName;
        }
        public void setHostName(String hostName) {
                this.hostName = hostName;
        }
        public String getUserId() {
                return userId;
        }
        public void setUserId(String userId) {
                this.userId = userId;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        public String getDB_PRODUCT() {
                return DB_PRODUCT;
        }
        public int getPORT_NUM() {
                return PORT_NUM;
        }
}
```

Setter methods to change values.
Getter methods are to get the value (returning value)
Since Constants value can not be changed , don't have setter methods, only getter methods. A Class can have **multiple initialize** blocks.
Purpose of multiple initialize blocks is used to **segregate** the constants, in the scenario where Java application is connecting to different external servers (resources).

Class data member as final and static :

```
public class Sample {
        final private int A=10;
        private int x;
        private int y;
}
```

In the above case, every object contains variables x, y and final data member A The difference is x and y values can be changed, where as value of A can not be changed (it is a constant) For every instance separate copy of A creates.

static and final

```
public class Sample {
        static final private  int A=10;
        private int x;
        private int y;
}
```

In the above case, only x and y part of Objects, **final and static** data member A, is not a part of object, it is a separate copy, it can be shared by all objects of the class (only read can not modify).

static and final, is a read only sharable data.
Initializing static and final data member.

1) Class level
2) Using static block.

1) Class Level

```
public class Sample {
        static  final int A=10;
}
```

Using static block:

```
public class Sample {
        static  final int A;
        static
        {
                A=20;
        }
}
public class Sample {
        static  final int A=10;
        static
        {
                A=20;// invalid as final data member does not take second time assignment.
        }
}
```

Purpose of static block is used to initialize final and static data members.
A Class can have multiple static blocks.

Purpose of multiple static blocks are used to segregate the static constants, in the scenario where Java application is connecting to different external servers (resources).

Purpose of Constructor, instance block and static block

Constructor :
to initialize variables

instance block :
to initialize **final** variables

static block :
to initialize **final and static** variable

method as final:
it prevents from overriding in the child class:

```
public class Sample{
        final  public void display()     {
        }
}
public class Test extends Sample  {
        //below method is not valid ( error ), can not override in the child class.
        public void display()  {
        }
}
```

static method can override **but** it should be static in **both classes**, parent as well as child.
static and final should not **override , both are static and final in both classes, parent as well as child.**
class as final :
it prevents from the inheritance. Final class members can not be inherited into child class.

```
final public class Sample
{
}
// in the below, extends Sample is not valid
public class Test extends Sample
{
}
```

Final class don't have sub class / child class

Abstract :

It is an access modifier
It can be used with following
1) classes
2) methods

Abstract class:
classes are of two types
1) Abstract class it does not allow to create an instance
2) Concrete class it allows to create an instance

Declaring class as an abstract:

```
public abstract class Sample{
        public void display(){
        }
        public void show(){
        }
}
```

Abstract class does not allow to create instance.
Abstract class can have all abstract methods (or) all concrete methods (or) it can have both.

Concrete class allows to create instance. Concreate class **should** contain all concrete methods only Not Abstract Method Abstract it is a partially implemented class.
abstract class does not allow to create an object but allows to declare reference .
purpose of abstract class reference variable is to refer to child class instance (used in polymorphism)
Eg:
Sample s1 = **new** Sample(); ---> Invalid, does not allow to create an object
Sample s1; valid since, it is only reference.
purpose of abstract class is to provide members for the child classes.

Method as an abstract :
Methods are of two types :
1) abstract method does not have functionality (definition)
2) concrete method it is a method with functionality (definition)

1) abstract method:
declaration.

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

it does not allow to define (does not have body)
it is only declaration . don't have definition
Below definition is invalid.

```
public abstract void display() {
}
```

2) concrete method

```
public void display(){
}
```

if class contains at least one abstract method, then class also becomes as an abstract class.

```
public abstract class Sample{
        public abstract void display();
        public void show(){
        }
```

```
}
```
implementing abstract method in child class.
```
public class Test extends Sample
@Override
        public void display() {
        }
}
```
Abstract method should be implemented in the child class, otherwise child class also become as an abstract class.
An abstract class can have all concrete methods. But It is not mandatory to have an abstract method.

Below abstract class contains all concrete methods. It is valid one.
```
public abstract class Sample{
        public void display(){
        }
        public void show(){
        }
}
class Test extends Sample{
        // overriding in child class
        public void display(){
        }
        // overriding in child class
        public void show(){
        } }
Test t = new Test();
t. display();
t. show();
```
Abstract:
========
it can be used with classes and methods there are two types of classes
1) Concrete class
2) Abstract class

1) Concrete class:
it allows to create an Object it should contains all concrete methods only it will not allow abstract methods

2) Abstract class:
it does not allow to create an Object it allows do declare only reference but object is not valid. an abstract class, can have all concrete methods. abstract class, need not to have an abstract method compulsory.
it is not compulsory to have an abstract method. an Abstract class can have **all concrete methods**, or can have **all abstract methods** , or it can have **both**. purpose of abstract class is to provide the members for child classes. It is partially implemented class. it is used in polymorphism

Method as an Abstract :
There are two types of methods
1) Concrete Method
2) Abstract Method

1) Concrete Method:
it is a method with defination

2) Abstract Method:
It is a declaration , don't have defination. if class contains at least one abstract method ,then class converts into abstract .all abstract methods should be implemented (defined)in child class, otherwise child becomes as an abstract. If method functionality is not known , then declare method as abstract .

Note :
Abstract classes and abstract methods are used in polymorphism

Java-Interface

interface is similar to abstract class it allows only reference declaration it does not allow to create instance
default properties of interface :

1) interface by default, it is a **public** interface data members are by default **public , static and final**.
2) Interface methods are by default **public and abstract.**

Defining interface:

```
public interface MyInterface{
        public static final int A=10; // here public static final,  is an optional.
        public static final int B=20;
        public abstract void display();  // here public and abstract , is an optional .
        public abstract void show();
}
```

Interface does not allow to create an object.

```
MyInterface i1 = new MyInterface();// invalid.
```

implements is a keyword to inherit interface members into child class.

```
public  class Sample implements MyInterface{
        @Override
        public void display(){
        }
        @Override
        public void show(){
        }       }
```

all methods of an interface should be implemented in the child class, otherwise child class becomes as an abstract class.

```
public interface MyInterface {
        public static final int A=10;  // public static final are optional
        public static final int B=10;
        public abstract void display();
        public abstract void show();   // public abstract is an optional
}
public  class Sample implements MyInterface{
        @Override
        public void display() {
                // TODO Auto-generated method stub
                System.out.println("I am in display");
                System.out.println("A = "+A+"\t B= "+B);
        }
        @Override
        public void show() {
                // TODO Auto-generated method stub
                System.out.println("I am in show");
                System.out.println("A = "+A+"\t B= "+B);
        }
}
public  class Test {
        public static void main(String[] args) {
                Sample s1 = new Sample();
                s1.display();
                s1.show();
        }  }
```

Concrete class Vs Abstract class Vs Interface

1) concrete class, is a fully implemented class (fully qualified class)
2) abstract class, is a partially implemented (partially qualified class)
3) interface is a fully abstract

an interface can refer to child class instance . a parent can refer to child class instance **but** child **can not** refer to parent.

parent reference and child class instance

```
public interface MyInterface {
        public static final int A=10;  // public static final are optional
        public static final int B=10;
        public abstract void display();
        public abstract void show();   // public abstract is an optional
}
public  class Sample implements MyInterface{
```

```
	@Override
	public void display() {
		// TODO Auto-generated method stub
		System.out.println("I am in display");
		System.out.println("A = "+A+"\t B= "+B);
}
	@Override
	public void show() {
		// TODO Auto-generated method stub
		System.out.println("I am in show");
		System.out.println("A = "+A+"\t B= "+B);
} }
public  class Test{
	public static void main(String[] args) {
		MyInterface i1 = new Sample(); //parent reference and child class instance
		i1.display();
		i1.show();
	}
}
```

o/p

I am in display A = 10 B= 20

I am in show A = 10 B= 20

Child class own methods, should not be invoked through the parent reference.

```
public  class Sample implements MyInterface{
	@Override
	public void display() {
		// TODO Auto-generated method stub
		System.out.println("I am in display");
		System.out.println("A = "+A+"\t B= "+B);
	}
	@Override
	public void show() {
		// TODO Auto-generated method stub
		System.out.println("I am in show");
		System.out.println("A = "+A+"\t B= "+B);
	}
	// it's own method , not from the parent
	public void demo(){
		System.out.println("I am in demo ");
		System.out.println("A = "+A+"\t B= "+B);
	}
}
public  class Test{
	public static void main(String[] args) {
		MyInterface i1 = new Sample();
		i1.display();
		i1.show();
		i1.demo();	// Invalid statement
	} }
```

@Override , means that method from the parent class

To invoke **child class method**, through the **parent reference**, it should be declared in the **parent interface.**

```
public interface MyInterface {
	public static final int A=10;  // public static final are optional
	public static final int B=20;
	public abstract void display();
	public abstract void show();   // public abstract is an optional
	public abstract void demo();  // this declaration required for the above example
}
```

An interface can extends of another interface **but not** implements

```
public interface MyInterface1 {
        public abstract void display();
        public abstract void show();
}
public interface MyInterface2 extends MyInterface1{
        public void demo();
        public void put();
}
```

Child interface contains both parent interface members as well it's own members Child class for above MyInterface2
Below child class has to implement both MyInterface1 as well MyInterface2 methods, otherwise it becomes as an abstract class.

```
public  class Sample implements MyInterface2{
        @Override
        public void display() {
        }
        @Override
        public void show() {
        }
        @Override
        public void demo() {
        }
        @Override
        public void put() {
        }   }
```

Inheritance between classes and interface:

Parent -->	class	interface	interface	class
	extends	**implements**	**extends**	**Not Applicable**
child -->	class	class	interface	interface

a class can implement multiple interfaces but extends of only one class.

```
public interface MyInterface1 {
        public abstract void display();
        public abstract void show();
}
public interface MyInterface2 {
        public void demo();
        public void put();
}
// child class implementing multiple interfaces
public  class Sample implements MyInterface1, MyInterface2{
        @Override
        public void display() {
}
        @Override
        public void show() {
        }
        @Override
        public void demo() {
        }
        @Override
        public void put() {
        // TODO Auto-generated method stub
        }
}
```

Class with **extends** as well **implements** both

final syntax of the class:

```
public class MyChildClass extends MyParentClass  implements MyInterface1, MyInterface2, MyInterface3 {
}
```

If all Parent interfaces contains same data member, then duplicate copies creates at child class level, then it is an ambigious to access.

To access parent interface members , use interface name.
Eg:

```
public interface MyInter1 {
        public static final int A=10;
}
public interface MyInter2 {
        public static final int A=20;
}
public interface MyInter3 {
        public static final int A=30;
}
public class Sample implements MyInter1, MyInter2, MyInter3 {
    public void display() {
    System.out.println( A );// invalid, there is an ambiguity in accessing parent members, as creating multiple copies.
        System.out.println(MyInter1.A);//valid
        System.out.println(MyInter2.A); //valid
        System.out.println(MyInter3.A); //valid
    }  }
```

Interface data members are public , since these are public , these behaves like a Global data, these can be used any where in the project by importing interface.
If it is same package , importing is not necessary .

```
package com.durga.mnrao.x;
public interface MyInterface1 {
    public static final int A=10;
package com.durga.mnrao.x;
public interface MyInterface2 {
    public static final int A=20;
package com.durga.mnrao.x;
public interface MyInterface3 {
    public static final int A=30;
package com.durga.mnrao.y;
// importing interfaces from another package
import com.durga.mnrao.x.MyInterface1;
import com.durga.mnrao.x.MyInterface2;
import com.durga.mnrao.x.MyInterface3;
public  class Test{
    public static void main(String[] args) {
        System.out.println( MyInterface1.A );
        System.out.println( MyInterface2.A );
        System.out.println( MyInterface3.A );
}  }
```

All points about Interface:
implements is a keyword , to inherit members of interface into child class
interface methods should be defined at child class level, otherwise child class converts into abstract class.
an interface can refer to it's child class instance parent reference and child class instance (polymorphism)
if any method is invoking through the interface reference it should be declared inside the interface;
an interface can extend another interface for inheritance but not implements
Relationship between classes and interfaces :

Parent --->	class	interface	interface	class
	extends	**implements**	**extends**	**Not** Apllicable
Child ---->	class	class	interface	Interface

a class can implement multiple interfaces but extends of only class.
final Syntax of class :
access_specifier class child_class_name extends Parent_class_name implements Inter1, Inter2, Inter3...
{
}
Adapter class:
below interface using at project level
package com.durga.mnrao.x;

```
public interface MyInterface {
public abstract void m1(); public abstract void m2(); public abstract void m3(); public abstract void m4(); public abstract void m5(); public abstract void m6(); public abstract void m7(); public abstract void m8(); public abstract void m9(); public abstract void m10();
}
```

module1 with class A
only m1() and m2() required to implement

```
public class A implements MyInterface {
	@Override
	public void m1() {
		// TODO Auto-generated method stub
}
	@Override
	public void m2() {
	// TODO Auto-generated method stub
	}
}
```

module2 with class B
only m3() and m4() required to impelment

```
public class B  implements MyInterface{
	@Override
	public void m3() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m4() {
	// TODO Auto-generated method stub
	}
}
```

module3 with class C
only m5() and m6() required to impelment

```
public class C implements MyInterface{
	@Override
	public void m5() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m6() {
		// TODO Auto-generated method stub
	}
}
```

mudule4 with class D
only m7() and m8() required to impelment

```
public class D implements MyInterface{
	@Override
	public void m7() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m8() {
		// TODO Auto-generated method stub
	}
}
```

module5 with class E
only m9() and m10() required to impelment

```
public class E implements MyInterface{
	@Override
	public void m9() {
		// TODO Auto-generated method stub
```

```
	}
	@Override
	public void m10() {
		// TODO Auto-generated method stub
	}
}
```

since all the above classes implementing MyInerface ,
these becomes abstract as these are implementing only required methods not all methods of interface.
Solution for the above requirement.

Below is the adapter class.
It is class with empty implementation of interface method

```
public class Temp implements MyInterface{
	@Override
	public void m1() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m2() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m3() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m4() {
	// TODO Auto-generated method stub
	}
	@Override
	public void m5() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m6() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m7() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m8() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m9() {
		// TODO Auto-generated method stub
	}
	@Override
	public void m10() {
		// TODO Auto-generated method stub
	}
}
```

Below A , B, C, D and E are the actual classes to implement different methods for different usage.

```
public class A extends  Temp{
	@Override
	public void m1() {
		// TODO Auto-generated method stub
```

```
		System.out.println("I am in m1");
	}
	@Override
	public void m2() {
		// TODO Auto-generated method stub
		System.out.println("I am in m2");
	}
}
public class B extends  Temp{
	@Override
	public void m3() {
		// TODO Auto-generated method stub
		System.out.println("I am in m3");
}
	@Override
	public void m4() {
		// TODO Auto-generated method stub
		System.out.println("I am in m4");
	}
}
public class C extends  Temp{
	@Override
	public void m5() {
		// TODO Auto-generated method stub
		System.out.println("I am in m5");
}
	@Override
	public void m6() {
		// TODO Auto-generated method stub
		System.out.println("I am in m6");
	}
}
public class D extends  Temp{
	@Override
	public void m7() {
		// TODO Auto-generated method stub
	System.out.println("I am in m7");
	}
	@Override
	public void m8() {
		// TODO Auto-generated method stub
		System.out.println("I am in m8");
	} }
public class E extends  Temp{
	@Override
	public void m9() {
		// TODO Auto-generated method stub
		System.out.println("I am in m9");
}
	@Override
	public void m10() {
		// TODO Auto-generated method stub
		System.out.println("I am in m10");
	}  }
```

marker interface

it is an empty interface , which has no data members and methods

```
public interface MyInterface{
}
```

main purpose of abstract classes, abstract methods and interfaces, is to achieve the polymorphism:

Java-Polymorphism

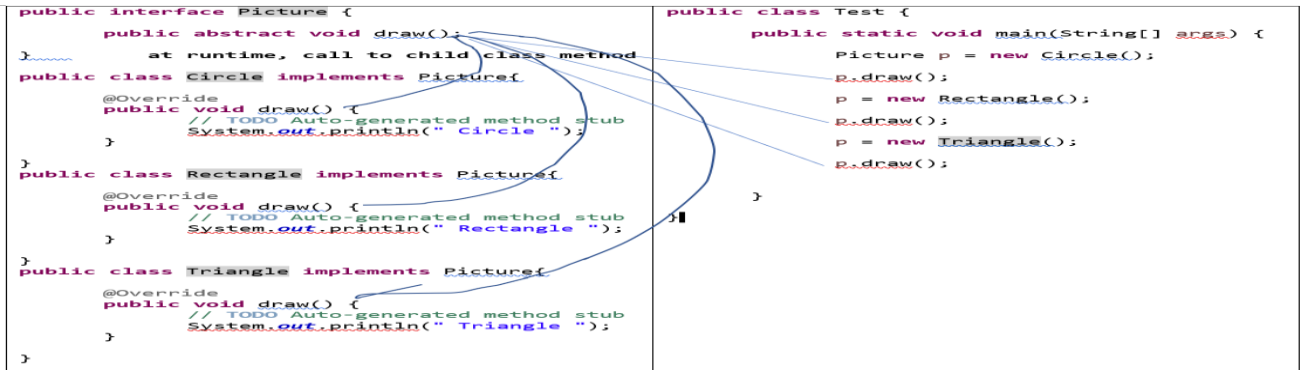
Poly --> many
Morphism --> forms
def: Same method, defining and behaving differently for different purpose is called as polymorphism.
Eg:

Picture (interface) ? Parent
draw()

Circle | Rectangle | Triangle ? child classes
draw() | draw() | draw()

```
public interface Picture{
	public abstract  void draw();
}
public class Rectangle implements Picture{
	@Override
	public void draw() {
		System.out.println ("Rectangle");
}
public class Square implements Picture {
	@Override
	public void draw() {
		System.out.println ("Square");
}   }
public class Triangle implements Picture {
	@Override
	public void draw() {
		System.out.println ("Triangle");
	}
}
public class PolyTest{
	public static void main(String[] args){
		Picture p1;
		p1=new Rectangle();
		p1.draw();
		p1=new Triangle();
		p1.draw();
		p1=new Square();
		p1.draw();
	}
}
```

Dynamic Binding : Compile time links to parent but run time call to child class methods



Memory map of Objects at run time

Another way of main() ,

Anthor way of implementing ploymorphism

```
public class PolyTest{
	public static void main(String[] args){
		dispaly( new Rectangle() );
		dispaly( new Triangle() );
		dispaly( new Square() );
```

```
        }
        // if type of method argument, is parent type, then you can pass any child class instance
        // parent reference and child class instance
        public static void dispaly(Picture  x){
                x.draw();
        }
}
// if type of method argument, is parent type, then you can pass any child class instance
        // parent reference and child class instance
```

Another example:

```
package com.visix;
public interface Bank {
        public  int  getRateOfInterest();
}
```

The Bank interface has a method to calculate Rate Of Interest.

CitiBank.java:

```
package com.visix;
public class CitiBank implements Bank {
        public int getRateOfInterest() {
        return 15;
        }  }
```

HdfcBank.java:

```
package com.visix;
public class HdfcBank implements Bank {
  public int getRateOfInterest() {
    return 13;
  } }
```

The CitiBank and HdfcBank classes implement the Bank interface and the classes generate interest rate related to CitiBank and HdfcBank.

```
public class BankApplication {
  public static void main(String as[]) {
                Bank bank = new CitiBank();
                System.out.println (bank.getRateOfInterest());
                bank = new HdfcBank();
                System.out.println (bank.getRateOfInterest());
  } }
```

Polymorphism implementation with abstract class

```
package com.durga.mnrao.bank;
public abstract class ReserveBankOfIndia {
        public int getMinRateOfInterest(){
                return 6;
        }
        public int getMaxRateOfInterest(){
                return 13;      }
        public abstract int getBankRateOfInterest(); }
package com.durga.mnrao.bank;
public class AXISBank extends ReserveBankOfIndia{
        @Override
        public int getBankRateOfInterest() {
                // TODO Auto-generated method stub
                return 9;       }}
package com.durga.mnrao.bank;
public class HDFCBank extends ReserveBankOfIndia{
        @Override
        public int getBankRateOfInterest() {
                // TODO Auto-generated method stub
                return 10;
        } }
package com.durga.mnrao.bank;
```

```
public class ICICIBank extends ReserveBankOfIndia{
	@Override
	public int getBankRateOfInterest() {
		// TODO Auto-generated method stub
		return 11;	}
}
package com.durga.mnrao.bank;
public class Test {
	public static void main(String[] args) {
		ReserveBankOfIndia rbi= new  HDFCBank();
		System.out.println(rbi.getMinRateOfInterest());
		System.out.println(rbi.getMaxRateOfInterest());
		System.out.println(rbi.getBankRateOfInterest());
		rbi= new  AXISBank();
		System.out.println(rbi.getMinRateOfInterest());
		System.out.println(rbi.getMaxRateOfInterest());
		System.out.println(rbi.getBankRateOfInterest());
		rbi= new  ICICIBank();
		System.out.println(rbi.getMinRateOfInterest());
		System.out.println(rbi.getMaxRateOfInterest());
		System.out.println(rbi.getBankRateOfInterest());	} }
```

Another way of main() , implementation

Method argument type is parent, then we can pass any child class instance.

```
public class Test {
	public static void main(String[] args) {
		ReserveBankOfIndia  rbi = new AxisBank();
		System.out.println("Axis Bank ");
		getInterestDetails( new AxisBank() );
		System.out.println("HDFC  Bank ");
		getInterestDetails( new HDFCBank() );
		System.out.println("SBI  Bank ");
		getInterestDetails( new SBIBank() );
	}
	public static void getInterestDetails( ReserveBankOfIndia  rbi ){
		System.out.println("min Interest = "+rbi.getMinRateOfInterest());
		System.out.println("Max Interest = "+rbi.getMaxRateOfInterest());
		System.out.println("Actual Interest = "+rbi.getActualRateOfInterest());
	}
}
```

static binding	dynamic binding
if program executed as per compiler linking , that is called as static binding	if program not executed as per compiler linking , that is called as dynamic binding
if program executed as per compiler linking , that is called as static binding	if compiler don't know about run time process, that is called as dynamic binding
it is a method overloading it is called as compile time polymorphism	it is a Method Overriding it is called as run time polymorphism
it happens in the same class	it happens between parent and child

Java-Garbage-Collection

it is a clearing all unused objects. JVM provides GarbageCollector to clean all unused objects (garbage)

invoking Garbage Collector :

System.gc();

Example:

```
package com.durga.mnrao.gc;

public interface Picture {
	public void draw();
}
package com.durga.mnrao.gc;
public class Rectangle implements Picture{
	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Rectangle");
	}
	@Override
	protected void finalize() throws Throwable {
		// TODO Auto-generated method stub
		System.out.println("Rectangle Object destroyed ");
	} }
package com.durga.mnrao.gc;
public class Triangle implements Picture{
	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Triangle");
	}
	@Override
	protected void finalize() throws Throwable {
		// TODO Auto-generated method stub
		System.out.println("Triangle Object destroyed ");
	}  }
package com.durga.mnrao.gc;

public class Circle implements Picture{
	@Override
	public void draw() {
		// TODO Auto-generated method stub
		System.out.println("Circle");
	}
	@Override
	protected void finalize() throws Throwable {
		// TODO Auto-generated method stub
		System.out.println("Circle Object destroyed ");
	} }
package com.durga.mnrao.gc;
public class Test {
	public static void main(String[] args) {
		Picture p = new Rectangle();
		p.draw();
		p = new Triangle();
		p.draw();
		p = new Circle();
		p.draw();
		System.gc();
	} }
```

In the above Rectangle and Triangle are unreferenced , treated as garbage .

When called to Garbage collector ,these objects get cleared from Heap Memory by the Garbage collector.

Nested Classes :

```
package com.durga.mnrao.nested;

public class Employee {
	private int empNum;
	private String empName;
	private double empSalary;
	private String empDeptName;
	private String empGender;
	private int empAge;
	public class DateOfBirth{
		private int day;
		private int month;
		private int year;
		public int getDay() {
			return day;
	}
		public void setDay(int day) {
			this.day = day;
	}
		public int getMonth() {
			return month;
	}
		public void setMonth(int month) {
			this.month = month;
	}
		public int getYear() {
			return year;
	}
		public void setYear(int year) {
			this.year = year;
	}	};
	public int getEmpNum() {
		return empNum;
	}
	public void setEmpNum(int empNum) {
		this.empNum = empNum;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSalary() {
		return empSalary;
	}
	public void setEmpSalary(double empSalary) {
		this.empSalary = empSalary;
	}
	public String getEmpDeptName() {
		return empDeptName;
	}
	public void setEmpDeptName(String empDeptName) {
		this.empDeptName = empDeptName;
```

```
	}
	public String getEmpGender() {
		return empGender;
	}
	public void setEmpGender(String empGender) {
		this.empGender = empGender;
	}
	public int getEmpAge() {
		return empAge;
	}
	public void setEmpAge(int empAge) {
		this.empAge = empAge;
	}
}
package com.durga.mnrao.nested;
public class Test {
	public static void main(String[] args) {
	Employee employee  = new Employee();
	employee.setEmpNum(1001);
	employee.setEmpName("mnrao");
	employee.setEmpSalary(50050.50);
	employee.setEmpDeptName("admin");
	employee.setEmpGender("male");
	employee.setEmpAge(35);
	Employee.DateOfBirth  dob = employee.new DateOfBirth();
	dob.setDay(10);
	dob.setMonth(3);
	dob.setYear(2010);
	int empNum = employee.getEmpNum();
	String empName = employee.getEmpName();
	double empSalary = employee.getEmpSalary();
	String empDeptName = employee.getEmpDeptName();
	String empGender = employee.getEmpGender();
	int empAge = employee.getEmpAge();
	int day = dob.getDay();
	int month = dob.getMonth();
	int year = dob.getYear();

System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge+"\t"+day
+"-"+month+"-"+year);
	}    }
```

Java- Strings

String is a class from java.lang package. It is not a basic data type. It is user defined type (non-premitive) Java String provides a lot of concepts that can be performed on a string such as compare, concat , equals, split, length, replace, compareTo , intern, substring etc.

Different ways of Creating a string

String Assignment:

```
String s;
s="hello";
```

String initialization :

```
String s1="hello";
```

Using Constructor :

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

Creating String from Another String

```
String s3 = new String(s1);
```

String reference assignment :

```
String s4 = s1;
```

From byte array:

```
byte [] b = {65,66,67,68,69,70};
String s5 = new String(b);
```

From char array:

```
char [] ch = {'h','e','l','l','o'};
String s6 = new String(ch);
```

Null String :

String s7=**null**; ? string reference is null, it can not be used, it throws NullPointerException.

Empty String :

String s8= **new** String(); ? s2 is reference String object, which contains nothing, empty object (zero no of chars)

The java.lang.String class implements Serializable, Comparable and CharSequence interfaces
The java String is immutable i.e. it cannot be changed but a new instance is created. For every time of assignment it will create new location. For mutable class, you can use StringBuffer and StringBuilder class.
How to create String object?

There are two ways to create String object:
By string literal
By new keyword
1) String Literal
Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
String s1="Welcome";
String s2="Welcome";//will not create new instance
```

In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool.

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).
2) By new keyword

```
String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).
Java String Example

```
public class Test {
        public static void main(String[] args)  {
                String s1="hello";
                System.out.println(s1);
                String s2 = new String("hello");
                System.out.println(s2);
                String s3 = new String(s1);
                System.out.println(s3);
                String s4 = s1;
                System.out.println(s4);
                byte [] b = {65,66,67,68,69,70};
                String s5 = new String(b);
                System.out.println(s5);
                char [] ch = {'h','e','l','l','o'};
                String s6 = new String(ch);
                System.out.println(s6);
        }   }
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1.	char charAt(int index)	returns char value for the particular index
2.	int length()	returns string length
3.	getBytes()	to convert string to byte array.
4.	String substring(int beginIndex)	
5.	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
6.	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
7.	boolean equals(Object another)	checks the equality of string with object
8.	boolean isEmpty()	checks if string is empty
9.	String replace(char old, char new)	replaces all occurrences of specified char value
10.	String replace(CharSequence-old,CharSequence new)	replaces all occurrences of specified CharSequence
11.	String trim()	returns trimmed string omitting leading and trailing spaces
12.	String [] split(String regex)	returns splitted string matching regex
13.	int indexOf(int ch)	returns specified char value index
14.	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
15.	int indexOf(String substring)	returns specified substring index
16.	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
17.	String toLowerCase()	returns string in lowercase.
18.	String toUpperCase()	returns string in uppercase

1. charAt()

```
String s1="hello";
char ch = s1.charAt(0);
System.out.println(ch);
ch = s1.charAt(5);                    // throws StringIndexOutOfBoundsException
```

2. length()

```
String s1="hello";
int len = s1.length();
System.out.println(len);
```

3. getBytes()

```
String s1="hello";
byte[] b = s1.getBytes();// to convert string to byte array.
for (int i = 0; i < b.length; i++) {
        System.out.println(b[i]);
}
```

String to char array :
toCharArray()

```
String s1 = "abcdef";
char [] ch = s1.toCharArray();
```

4. indexOf()

```
String s1="hello";
int index = s1.indexOf('l');
System.out.println(index);
```

5. isEmpty() return true if string is empty.

```
String s1="";
if(s1.isEmpty()){
        System.out.println("yes");
}
else{
        System.out.println("No empty");
}
String s1=null;
if(s1.isEmpty())  // NullPointerException{
        System.out.println("yes");
}
else{
        System.out.println("No empty");
}
```

6.replace()

```
String s1="this is java";
String s2 = s1.replace("java", "kava");
System.out.println(s1);// no change, since string is immutable object.
O/P: this is java
System.out.println(s2);
O/P: this is kava
```

String is immutable object, it can not be modified .

Types of objects in java.

There are two types

1) mutable
2) immutable

7. toUpperCase()

```
String s1="hello";
String s2 = s1.toUpperCase();
System.out.println(s1);
System.out.println(s2);
```

For every attempt to modify, it will create new location and return reference to new location.

8.toLowerCase()

```
String s1="HeLlO12345";
String s2 = s1.toLowerCase();
System.out.println(s1);
System.out.println(s2);
```

9.trim()

```
String s1=" hello ";
System.out.println(s1.length()); // 7
String s2 = s1.trim();
System.out.println(s1.length()); // 7
System.out.println(s2.length()); // 5
```

10. contains()

```
String s1="this is java";
if(s1.contains("is")){
        System.out.println("yes");
}
else{
        System.out.println("no");
}
```

11.indexOf()

```
String s1="this is java";
int index = s1.indexOf("java");
System.out.println(index);
String s1="this is java";
int index = s1.indexOf("java", 0);
System.out.println(index);
```

Java String comparison

We can compare string in java on the basis of content and reference.

there are three ways to compare strings in java:

1. By equals() method
2. By = = operator
3. By compareTo() method

It is used in authentication (by equals() method),
sorting (by compareTo() method),
reference matching (by == operator) etc

1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

public boolean equals(Object another)
compares this string to the specified object.

public boolean equalsIgnoreCase(String another)
compares this String to another string, ignoring case.

```
public class Test {
  public static void main(String [] args) {
   String s1="hello";
   String s2="hello";
  if( s1.equals(s2) ) {
        System.out.println ("both are equal");
   }
   else  {
        System.out.println ("not equal");
   } } }
```

o/p :

both are equal

```
public class Test {
  public static void main(String [] args)  {
   String s1="hello";
   String s2="Hello";
     if(s1.equals(s2))  {
        System.out.println ("both are equal");
   }
   else  {
        System.out.println ("not equal");
   }  } }
```

O/P ;

not equal

equals() ? it is a case sense

equalsIgnoreCase() ? ignoring case sense

```
public class Test {
  public static void main(String [] args) {
   String s1="hello";
   String s2="Hello";
   if(s1.equalsIgnoreCase(s2)) {
        System.out.println ("both are equal");
   }
   else {
        System.out.println ("not equal");
   }
  } }
```

O/p : both are equal

Program to validate userid and password.

```
import java.util.Scanner;
public class Test {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
```

```
                System.out.println("Enter user id");
                String userId = sc.next();
                System.out.println("Enter Passwd :");
                String pwd = sc.next();
                if (userId.equalsIgnoreCase("mnrao11@gmail.com") && pwd.equals("java12345")) {
                System.out.println("valid user ");
                } else {
                        System.out.println("In valid user ");
                } } }
```

2) String compare by == operator

The = = operator compares references (locations) not values.

```
public class Test {
        public static void main(String[] args) {
                String s1 = new String("hello");
                String s2 = new String("hello");
                if(s1==s2){
                        System.out.println("same location");
                }
                else {
                        System.out.println("different location");
                } } }
```

o/p : different location

```
public class Test {
        public static void main(String[] args) {
                String s1 = "hello";
                String s2 = new String("hello");
                if (s1 == s2) {
                System.out.println("same location");
                } else {
                        System.out.println("different location");
                } } }
```

o/p : different location

String s2 = new String("hello");

How may objects are created

Ans : 1 or two

If "hello" already exist in the string pool, then only one creates inside the heap memory.

If "hello" does not exist in the string pool, then one object with "hello" placing inside the string pool and another is created inside the heap memory.

```
public class Test{
  public static void main(String []  args)   {
    String s1="hello";
    String s2="hello";
    if(s1==s2)     {
          System.out.println ("Same location");
    }
    else    {
          System.out.println ("Different location");
    }   }   }
```

o/p; Same location

```
public class Test {
        public static void main(String[] args) {
                String s1 = "hello";
                if (s1 ==new String( "hello")) {
                System.out.println("same location");
                }
                else {
                   System.out.println("different location");
                }    }  }
```

different location

```
public class Test {
        public static void main(String[] args) {
                if (new String("hello") == "hello") {
                        System.out.println("same location");
                } else {
                        System.out.println("different location");
                } } }
```

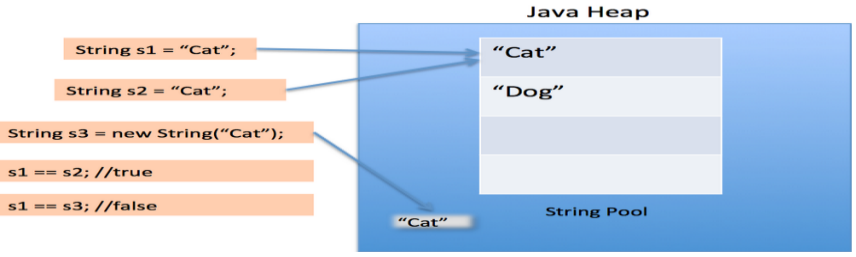
different location

```
public class Test {
        public static void main(String[] args) {
                if ("hello" == "hello") {
                System.out.println("same location");
                } else {
                        System.out.println("different location");
                } } }
```

o/p: same location

Java String Pool?

Here is a diagram which clearly explains how String Pool is maintained in java heap space and what happens when we use different ways to create Strings.



String Pool is possible only because **String is immutable in Java** and it's implementation of **String interning** concept. String pool is also example of **Flyweight design pattern**.
String pool helps in saving a lot of space for Java Runtime although it takes more time to create the String.

When we use double quotes to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference.

However using *new* operator, we force String class to create a new String object in heap space. We can use intern() method to put it into the pool or refer to other String object from string pool having same value.

```
public class Test {
  /**
   * Java String Pool example
   * @param args
   */
  public static void main(String[] args) {
    String s1 = "Cat";
    String s2 = "Cat";
    String s3 = new String("Cat");
    System.out.println("s1 == s2 :"+(s1==s2));
    System.out.println("s1 == s3 :"+(s1==s3));
  }  }
```

s1 == s2 :true
s1 == s3 :false
how many string are getting created in below statement;
String str = new String("Cat");

In above statement, either 1 or 2 string will be created. If there is already a string literal "Cat" in the pool, then only one string "str" will be created in the pool. If there is no string literal "Cat" in the pool, then it will be first created in the pool and then in the heap space, so total 2 string objects will be created.

Advantage of String pool is to save the heap memory.

```
public class Test{
  public static void main(String [] args)  {
   String s1=new String("hello");
   String s2=new String("hello");
   if(s1==s2)    {
        System.out.println ("Same location");
   }
   else   {
        System.out.println ("Different location");
   }  }  }
```

o/p: Different location

```
public class Test{
  public static void main(String [] args)  {
   String s1=new String("hello");
   String s2=s1;
   if(s1==s2)    {
        System.out.println ("Same location");
   }
   else    {
        System.out.println ("Different location");
   }   }  }
```

O/P: Same location

```
public class Test{
  public static void main(String [] args)  {
   String s1=new String("hello");
   String s2=new String(s1);
   if(s1==s2)  {
        System.out.println ("Same location");
   }
   else  {
        System.out.println ("Different location");
   }  }  }
```

O/P; Different location

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string. It compares ASCII value of the String.

Suppose s1 and s2 are two string variables.

```
String s1=new String("hello");
String s2=new String("hello");

int diff = s1.compareTo(s2);
```

1) returns 0 if both Strings contains same value
2) returns +ve value if s1 > s2
3) returns -ve value if s1<s2

```
public class Test {
  public static void main(String [] args) {
   String s1=new String("hello");
   String s2=new String("hello");
   if(s1.compareTo(s2)==0){
        System.out.println ("Same");
   }
   else  {
        System.out.println ("Not Same");
   }  }  }
```

O/P: Same

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

```
    String s1=new String("hello");
    String s2=new String("Hello");
    if(s1.compareTo(s2)>0)  {
            System.out.println ("S1 > S2");
    }
    else  {
            System.out.println ("S2 > S1");
    }   }   }
```

O/P: S1 > S2

```
public class Test {
   public static void main(String []  args)   {
    String s1=new String("Hello");
    String s2=new String("hello");
        if(s1.compareToIgnoreCase(s2)==0)   {
            System.out.println ("Same");
    }
    else  {
            System.out.println ("Not Same");
    }  }  }
```

O/P: same

compareTo() is used sort the strings in ascending order or descending order.

Sorting Strings:

```
public class Test  {
 public static void main(String []  args)   {
    String []names=new String[]{"java","hadoop","linux","unix","servlet","jsp","html"};
    System.out.println ("Before Sorting ");
     for (int i = 0; i < names.length; i++)   {
                    System.out.println (names[i]);
    }
    for (int i = 0; i < names.length; i++)    {
            for(int j=0; j<names.length-1-i;j++)    {
                    if((names[j].compareTo(names[j+1]))>0)
                    {
                            String temp = names[j];
                            names[j]=names[j+1];
                            names[j+1]=temp;
                    }    }      }
     System.out.println ("After Sorting ");
     for (int i = 0; i < names.length; i++)
    {
                    System.out.println (names[i]);    }
  }  }
```

String Concatenation in Java

+ is an operator to concatenate the Strings.

```
public class Test  {
   public static void main(String []  args)  {
    String s1="hello";
    String s2= s1+"java";
     String s3 = new String("world");
     System.out.println (s2);
     String s4 = s2+"\t"+s3;
      System.out.println (s4);
  }  }
```

O/P:

hellojava

hellojava world

eg:

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

```
        int eno=1001;
        String ename="mnrao";
        String job="manager";
        double sal = 50000.50;
        String dept="admin";
        String gender="male";
        int age = 25;
        String record = eno+","+ename+","+job+","+sal+","+dept+","+gender+","+age;
        System.out.println(record);
    }  }
```

Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
public class Test{
  public static void main(String []  args)   {
        String s1="Java is a programming language. Java is a platform. Java is an Island.";
          String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
          System.out.println (s1);
          System.out.println (replaceString);
  }  }
```

Output:

Java is a programming language . Java is a platform. Java is an Island.
Kava is a programming language. Kava is a platform. Kava is an Island.

replace(), will change value of the String as String is a immutable object.
It returns replaced value.
String value can not be changed for every attempt on the String it creates new location.

```
public class Test{
  public static void main(String []  args)   {
        String s1="Java is a programming language. Java is a platform. Java is an Island.";
         String replaceString=s1.replaceAll("Java","Kava");//replaces all occurrences of "Java" to "Kava"
          System.out.println (s1);
          System.out.println (replaceString);
    }   }
```

Output:

Java is a programming language. Java is a platform. Java is an Island.
Kava is a programming language. Kava is a platform. Kava is an Island.

Split() of String class:

It return array of Strings;

```
public class Test{
  public static void main(String []  args)   {
        String s1="java linux unix hadoop html";
        String [] s2 =s1.split(" ");
         for (String str : s2)      {
                System.out.println (str);
         }  }   }
```

O/P;

Java Linux Unix Hadoop html

```
public class Test {
        public static void main(String[] args) {
                String record = "1001:nrit:25:5000:male";
                String [] fields = record.split(":");
                System.out.println(fields[0]+"\t"+fields[1]+"\t"+fields[4]);
        }   }
```

1001   nrit   male

How to create Immutable class.

To create immutable class in java, you have to do following steps.

1. Declare the class as final so it can't be extended.
2. Make all fields private so that direct access is not allowed.
3. Don't provide setter methods for variables
4. Make all **immutable fields (final)** so that it's value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

Program to create immutable class

```
package com.nrit.mnrao.test;
final public class User {
	final private int uid;
	final private String uname;
	public User(){
		uid=0;
		uname=null;
	}
	public User(int uid, String uname){
		this.uid=uid;
		this.uname=uname;
	}
	public User(User  x){
		uid=x.uid;
		uname=x.uname;
	}
	public int getUid()
	{
		return uid;
	}
	public String getUname(){
		return uname;
	}
	public User  clone(){
		User temp = new User(uid,uname);
		return temp;
	}
	@Override
	public String toString() {
		return uid + "\t" + uname;
	}
}
public class Test {
	public static void main(String[] args) {
		User u1 = new User(1001,"mnrao");
		User u2 = u1.clone();
		User u3 = new User(u1);
		if( u1 != u2 ){
		System.out.println("successfully cloned ");
		}
		else {
			System.out.println("cloning failed");
		}
		if( u1!=u3 ){
			System.out.println("Successfully copy generated ");
		}
		else{
			System.out.println("copy failed ");
		}
		System.out.println(u1.getUid()+"\t"+u1.getUname());
```

```
            System.out.println(u2.getUid()+"\t"+u2.getUname());
            System.out.println(u3.getUid()+"\t"+u3.getUname());
    } }
```

Why String is immutable or final in Java

Why String is immutable in Java?

benefits of String immutability

String pool is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool. If String would not have been immutable, then String interning would not have been possible because if any variable would have changed the value, it would have been reflected to other variables also.
If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in **socket programming** host and port details passed as String. Since String is immutable it's value can't be changed otherwise any hacker could change the referenced value to cause security issues in the application.

Since String is immutable, it is safe for multithreading and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.
Strings are used in **java** classloader and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.

StringBuffer

It is a class from java.lang package
Java String-Buffer class is used to create mutable (modifiable) string. The String-Buffer class in java is same as String class except, it is mutable i.e. it can be changed.
Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.
Important Constructors of StringBuffer class
StringBuffer() --> creates an empty string buffer with the initial capacity of 16 bytes.
StringBuffer sb = new StringBuffer() ;
StringBuffer(String str)--> creates a string buffer with the specified string.
StringBuffer sb = new StringBuffer("hello") ;
StringBuffer(int capacity)--> creates an empty string buffer with the specified capacity as length.
StringBuffer sb = new StringBuffer(10) ;
Important methods of StringBuffer class

public synchronized String-Buffer append(String s):
is used to append the specified string with this string.
The append()method is overloaded like append(char),append(Boolean),append(int),append(float), append(double) etc.

public synchronized StringBuffer insert(int offset, String s):
is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

public synchronized StringBuffer replace(int startIndex, int endIndex, String str):
is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer delete(int startIndex, int endIndex):
is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer reverse()
is used to reverse the string.
public int capacity():
is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)
is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index):
is used to return the character at the specified position.

public int length()

is used to return the length of the string i.e. total number of characters.

public String substring(int beginIndex)

is used to return the substring from the specified beginIndex.

public String substring(int beginIndex, int endIndex)

is used to return the substring from the specified beginIndex and endIndex.

Eg:

```
public class Test {
	public static void main(String[] args){
		StringBuffer sb = new StringBuffer();
		String record1 = "1001#mnrao11#5000#25#finance";
		String record2 = "1002#mnrao2#6004#26#dev";
		String record3 = "1003#mnrao32#7000#23#testng";
		String record4 = "1004#mnrao4#6500#25#admin";
		sb.append(record1);
		sb.append("\n");
		sb.append(record2);
		sb.append("\n");
		sb.append(record3);
		sb.append("\n");
		sb.append(record4);
		String str = sb.toString();
		String [] records = str.split("\n");
		for (String record : records)  {
			System.out.println (record);
		} } }
```

below program will display only names of the record:

```
public class Test {
	public static void main(String[] args) {
	String [] records = {
					"1001,ajay,manager,account,45000,male,38",
					"1002,aiswrya,clerk,account,25000,female,30",
					"1003,varun,manager,sales,50000,male,35",
					"1004,amit,manager,account,47000,male,40",
					"1005,kareena,executive,sales,15000,female,24",
					"1006,deepak,clerk,sales,23000,male,30",
					"1007,sunil,accountant,sales,13000,male,29",
					"1008,satvik,director,purchase,80000,male,45"     };
		StringBuffer sb = new StringBuffer(255);
		for(int i=0; i<records.length; i++){
		sb.append(records[i]+"\n");
		}
		String empData = sb.toString();
		sb = null;
		String[] empRecords = empData.split("\n");
		for(String empRecord :empRecords){
			String[] fields = empRecord.split(",");
			if(fields[2].equals("manager") && fields[3].equals("account")) {
				System.out.println(empRecord);
			} } } }
```

No.	String	StringBuffer
1.	String class is immutable.	StringBuffer class is mutable.
2.	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3.	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Difference between String-Buffer and StringBuilder

No.	StringBuffer	StringBuilder
1.	StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2.	StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.

StringTokenizer

The **java.util.StringTokenizer** class allows an application to break a string into tokens.

- This class is a legacy class that is retained for compatibility reasons although its use is discouraged in new code.
- Its methods do not distinguish among identifiers, numbers, and quoted strings.
- This class methods do not even recognize and skip comments.

Class declaration

Following is the declaration for **java.util.StringTokenizer** class:

public class StringTokenizer extends Object implements Enumeration<Object>

Class constructors

StringTokenizer(String str)

This constructor a string tokenizer for the specified string.

StringTokenizer(String str, String delim)

This constructor constructs string tokenizer for the specified string.

StringTokenizer(String str, String delim, boolean returnDelims)

This constructor constructs a string tokenizer for the specified string.

int countTokens()

This method calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.

“java:oracle:linux:unix:sql”

boolean hasMoreElements()

This method returns the same value as the hasMoreTokens method

boolean hasMoreTokens()

This method tests if there are more tokens available from this tokenizer's string.

Object nextElement()

This method returns the same value as the nextToken method, except that its declared return value is Object rather than String.

String nextToken()

This method returns the next token from this string tokenizer.

String nextToken(String delim)

This method returns the next token in this string tokenizer's string.

```
import java.util.StringTokenizer;
public class Test {
        public static void main(String []  args) {
                System.out.println ("Using Constructor 1 - ");
                StringTokenizer st1 = new StringTokenizer("Hadoop Java Html Oracle Linux", " ");
                while (st1.hasMoreTokens()) {
                        System.out.println (st1.nextToken());
                }
                System.out.println ("Using Constructor 2 - ");
                StringTokenizer st2 = new StringTokenizer("Hadoop:Java:Html:Oracle:Linux", " :");
                while (st2.hasMoreTokens())  {
                        System.out.println (st2.nextToken());
                }       }    }
```

Data Conversion in Java

It is a conversion of data from one type to another type. It is called as type casting.
There are two types in type casting
1)Basic to Basic **(primitive type casting)**
Below all are Non-primitive type casting

1)Primitive	1)Basic to Basic		
2)Non Primitive	2)Basic to Object 3)Object to Basic	4) Object to Object	5) Sting to Numeric 6) Numeric to String

1)Basic to Basic (primitive types)
These are system defined types. all primitive types are basic type
eg: int, float ,char....
this type of conversion is type casting.
Primitive data types casting (system defined types)
These are
1) Implicit Casting and 2) Explicit casting
1) implicit casting:
it is a conversion of data from small type to big type. It is an implicit process (automatically/internally):

```
byte b=10;
short s=b;
```

here byte type of data converting into short (type promotion)
data and type promotion as below.
byte ===> short ===> int ===> long ==>float ===> double
char ==>int ==>long==>float ===> double
eg1:

```
byte b=10;
short s = b ; --> valid
```

eg2:

```
int a=10;
long b=a; --> valid
```

eg3:

```
float a=10.5f;
double b=a; --> valid
```

Implicit casting is available for the below
1) Small size to bigsize
2) Integer to real number
converting from real number to integer, implicit casting is not applicable. To convert real to integer type explicit casting required.
2) Explicit casting:
It is a conversion of data from big type to small type. Java developer has to convert explicitly .
it is required for below one
1) big type to Small type
2) real number type to Integer type
1) big type to Small type

```
short a=10;
```

byte b = a; --> invalid, since size of variable " a" is 2 bytes , which is more than size of variable "b" (1 byte)
such of kind of scenarios, we use explicit type casting.
short a=10;
byte b = (byte) a ; --> it is valid
same scenario with following
short a=130;
byte b = (byte) a ; --> it compiles but at run time loss of data. It results in unexpected data (i.e leads to bugs)
here value of b is -126 .
hence explicit type casting is recommended is only for constants, not for variables.
float a= 10.5 ; --> invalid, since in java real numbers by default treats as double.
float a= 10.5f; --> valid
default data type.
integer data (numbers) is a **int** data type but not a short and byte.
Real numbers are treated as **double** type.
Result of mathematical expressions, with different data types.
1) byte + byte = int
2) short + short = int
3) int + int = int (compiles) but at runtime there is chance of data over flow.
Recommended one is (int+int = long)
4) long + long = long
5) char + char = int.
6) float + float = double
7) double + double = double
8) int + long = long
9) float + double = double
10) int + float = float

Division (/) operation:
int a =10;
float x = a/3 ;
result value is 3.0
reason in division, both operands are int type then result int.
to get real number, at leat one operand must be float / double
int a =10;
float x = a / 3.0f ;
result is 3.333333
eg:
int a =10;
int b = 3;
float x = a / b ;
result value is 3.0
reason in division, both operands are int type then result int.
solution for the above is, type casting .
float x = (float) a / b ;
(or)
float x = a / (float) b ;

Object Wrapper classes

2)Basic to Object

for every basic data type, java provides Object Wrapper class to convert basic data type to Object type.

Basic type key words	Object Wrapper class
Byte	Byte
Short	Short
Int	Integer
Long	Long
Float	Float
Double	Double
Char	Character
Boolean	Boolean

Note: All the above wrapper classes are from java.lang package. Wrapper classes are immutable and final.

eg1:
int a=10;
to convert into Object:
Integer i1= new Integer(a);// i1 contains 10.
eg2:
double b=100.50;
Double d1 = new Double(b);
3) Object to Basic Type:
Object Wrapper classes provides typeValue() to convert from Object to basic type.
type--> data type.
Eg:
for integer type is **int**.
For float type is **float**
eg1:
```
Integer i1= new Integer(10);
int a = i1.intValue();// value of a is 10.
```
eg2:
```
Double d1 = new Double(100.50);
double b = d1.doubleValue();
```
Auto Boxing and Unboxing:
it converts basic to object, object to basic automatically. This concept was introduced from jdk1.5 (tiger version) onwards.
Auto boxing:
it is a conversion of data from basic to Object type:
eg:
```
Integer i1 = 10;// directly can be assigned.
Double d1 = 300.506;// directly can be assigned.
```
Auto unboxing:
```
Double d1= new Double(25.8);
double b = d1; // directly can be assigned. typeValue() not required.
```
4) Object to Object :
These are of two types
1) Up casting
2) Down casting
1) Up casting :
It is a conversion of data from child to parent. It is an implicit process.
Eg:
Below is the parent.
```
package com.nrit.mnrao.test;
public interface Picture {
        public void draw();
}
```
Child class:
```
package com.nrit.mnrao.test;
public class Rectangle implements Picture {
        @Override
        public void draw() {
                System.out.println("Rectangle");
        }   }
```
Parent reference and child instance (child to parent)
```
Picture p = new Rectangle();  // it is an implicit process.
```
2) Down Casting :
It is conversion of data from parent to child. It is an explicit casting.
```
                Picture picture = new Rectangle();  // parent reference.
                Rectangle rectangle =  picture; //Not valid, since Parent reference assign        to child
```
to convert from parent child implicit in not valid. The way is explicit casting.
```
                Picture picture = new Rectangle();
                Rectangle rectangle =  (Rectangle)  picture;// it is an explicit casting.
```
Before going converting from parent child, type of instance should be checked .
Eg:

Parent :

```
package com.nrit.mnrao.test;
public interface Picture {
        public void draw();
}
```

Child1 :

```
package com.nrit.mnrao.test;
public class Rectangle implements Picture{
        @Override
        public void draw() {
                // TODO Auto-generated method stub
                System.out.println("Ractangle draw");
        }   }
```

Child2:

```
package com.nrit.mnrao.test;
public class Circle implements Picture{
@Override
public void draw() {
                // TODO Auto-generated method stub
                System.out.println("Circle draw");
}   }
```

Main method for testing :

```
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args) {
                Picture p = new Rectangle();
                System.out.println("before conversion");
                Circle c = (Circle) p;  // it raises Class Cast Exception at runtime
                System.out.println("before draw");
                c.draw();
        }   }
```

Solution for the above :

Before going converting from parent to child, type of instance should be checked .

```
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args) {
                Picture p = new Rectangle();
                if ( p instanceof  Circle )   {
                        Circle c = (Circle) p;
                        c.draw();
                }
                else  {
                        System.out.println("Not a Circle instance");
                }   }   }
```

Object class:

Object class is a top most parent of complete JDK hierarchy. It is a default parent for every class, including user defined classes. After compilation of every class, compiler makes Object class as a parent of every class.

Eg:;

Source code.

Sample.java as below.

```
public class Sample {
}
```

After compilation

Sample.class file contains following code.

```
public class Sample extends Object{
}
```

All the members of Objet class, gets inherited into Sample class. Hence Sample class object contains Object class members as well it's own members.

Eg:

Parent class:

```
package com.nrit.mnrao.test;
public class Example {
	public void display(){
		System.out.println("Exmaple display");
	}
}
```

Child class :

```
package com.nrit.mnrao.test;
public class Sample extends Example{
}
```

Main method for testing above one :

```
package com.nrit.mnrao.test;
public class Test {
	public static void main(String[] args) {
		Sample s1 = new Sample();
		s1.display(); // it is from the Parent class
	}   }
```

In the above example , display() methods is not from child class (Sample) , even it is not form it's own class it will work as display() inherited from the parent class (Example)

i.e if any method is invoking through the object , then it should be either it's own method or from the parent class.

Members of Object class:

equals(); □ to compare two Objects

getClass(); □ to get class name for the object.

Eg:

```
		Sample s1 = new Sample();
		Class<? extends Sample> c  = s1.getClass();
		System.out.println(c.getName() );
		O/P :
			Sample.
```

hashCode(); □ return address of object in form of unsigned iteger.

Eg:

```
		Sample s1 = new Sample();
		int hashCode = s1.hashCode();
		System.out.println(hashCode);
```

notify() and notifyAll() □ used with threads to send notification to threads

notify() □ to send notification to first waiting thread in waiting queue.

notifyAll() □ to send notification to all waiting threads in waiting queue.

toString() □ to convert any type of data into string type.

```
		Sample s1 = new Sample();
		System.out.println(s1);
```

Here println() □ expecting string, hence it makes to toString(), if toString() is available in the child class, then it makes call to child class method, if not avaialable then it makes a call to Obeject class toString() method.

Object class toString() returns address (reference) of the object.

Overriding toString() method.

```
package com.nrit.mnrao.test;
package com.nrit.mnrao.test;
public class User {
	private int uid;
	private String uname;
	public User() {
		uid=0;
		uname=null;
	}
	public User(int uid,String uname)  {
		this.uid=uid;
		this.uname=uname;
```

```
	}
	public int getUid() {
		return uid;
	}
	public void setUid(int uid) {
		this.uid = uid;
	}
	public String getUname() {
		return uname;
	}
	public void setUname(String uname) {
		this.uname = uname;
	}
	@Override
	public String toString() {
		return "User [uid=" + uid + ", uname=" + uname + "]";
	}
}
package com.nrit.mnrao.test;
public class Test {
	public static void main(String[] args)		{
		User u1 = new User(1001,"hello1");
		User u2 = new User(1002,"hello2");
		User u3 = new User(1003,"hello3");
		System.out.println(u1);
		System.out.println(u2);
		System.out.println(u3);
	}  }
```

Note: toString() works only on the objects but not with basic data type.

Converting Object class type to String class type.

```
		String str = "hello";
		Object obj = str;  // valid since child is assigining to parent
					( up casting)
		Object obj = new Sample();
		String str =(String) obj; //compiles but raises ClassCastException
		System.out.println(str);
		//below is the valid coding for coversion.
if (obj instanceof String)  {
		String str = (String) obj;
}
else  {
	System.out.println("invalid runtime instance, expected for String type ");
}
```

wait() ? to put the thread into waiting state for shared resources.

5) String to basic type:

==========

every wrapper class provides static parseX() to convert from String to basic type.

Eg1:

```
String str="101";
int a =Integer.parseInt(str);// here value of a is 101.
```

Eg2:

```
String str="100.50";
float f = Float.parseFloat(str);
```

parseX() ? throws NumberFormatException, if string contains non-numeric value.

Eg1: to use parseX()

```
package com.nrit.mnrao.test;
import java.util.Scanner;
public class Test {
```

```
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter EMPNO:");
                String inputData = sc.next();
                int eno = Integer.parseInt(inputData);
                System.out.println("Enter name ");
                String name = sc.next();
                System.out.println("Enter Salary ");
                 inputData = sc.next();
                 double salary = Double.parseDouble(inputData);
                 System.out.println("Enter Dept ");
                 String dept = sc.next();
                 System.out.println(eno+"\t"+name+"\t"+salary+"\t"+dept);
        }
}
```

O/P;

```
Enter EMPNO:1001
Enter name:abc
Enter Salary:5000
Enter Dept :dev
1001    abc     5000.0 dev
```

Eg2: to use parseX()

Record coverting into variables:

```
package com.durga.mnrao.test;
public class Test {
        public static void main(String [] args) {
                String record = "1001:mnrao:manager:15000.50:admin:male:30";
                String[] fields = record.split(":");
                 int empNum = Integer.parseInt(fields[0]);
                 String empName = fields[1];
                 String empJob = fields[2];
                 double  empSalary = Double.parseDouble(fields[3]);
                 String empDeptName = fields[4];
                 String empGender = fields[5];
                 int  empAge = Integer.parseInt(fields[6]);
        System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"
\t"+empAge);
        }   }
```

6)Numeric (any type) to String type:

==========

String class provides static overloaded valueOf() method to convert any basic type of data into String type.

Eg1:

```
int a=125;
String str = String.valueOf(a);
```

Eg2:

```
float a=30.56f;
String str = String.valueOf(a);
```

To convert any object to String type:

```
Sample s1 = new Sample();
String str = String.valueOf(s1);
System.out.println(str);
```

Here valueOf() argument type is Object class type.

It method argument type is Object class type , then any type of object can be passed to that method (parent reference and child object).

As Object class is a default parent for every class it can refer to any object.

Commandline parameters

D:\test>java Test hello java world ==> command line (upto enter key)

java ? JVM

Test □ .class file
hello java world ==> command line parameters
passing to main(String[] args)
public static void main(String [] args)
hello java world
args array generates with three element
args[0] ==> hello
args[1] ==> java
args[2] ==> world

String [] args --> to read command line parameters

D:\test>java Test **hello java world**

array generates as below
args[0] ==> hello args[1] ==> java args[2] ==> world

D:\test>java Test **hello java**

array generates as below
args[0] ==> hello args[1] ==> java D:\test>java Test **hello**

array generates as below
args[0] ==> hello

D:\test>java Test □ no parameters

empty array generates

no. of elements in args array = no of parameters
args.length □ parameter count.
all command line parameters passes to main(String [] args)
Windows:
D:\test> notepad Test.java

```
public class Test {
        public static void main( String [] args) {
                int n = args.length;
                System.out.println ("No.of elements="+n);
        }
}
```

compilation:

Windows:
D:\test> notepad Test.java

compilation:

```
D:\test> javac Test.java
D:\test> java Test  hello  java  world
No.of elements= 3
```

program to display command line parameters.

```
D:\test> notepad Test.java
public class Test{
        public static void main(String []args){
                for( int i=0;i<args.length;i++){
                  System.out.println (args[i]);
                }//for closing
           }//main closing
}//class closing
```

compilation:

```
D:\test> javac Test.java
```

execution:

```
D:\test> java Test  hello  java  world
hello
java
world
D:\test> java Test  “hello  java”  world
hello java
world
```

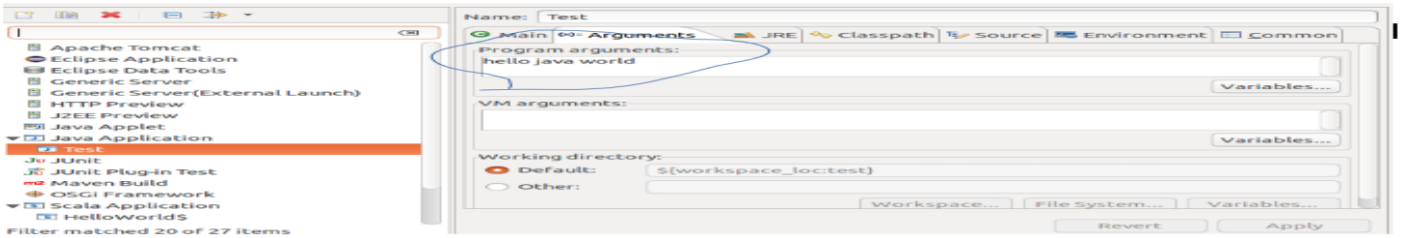
parmeter including space use double quote as above.

Passing command line parameters in eclipse:

rt.click on the main() program --> Runas --> RunConfiguration-->
double click on JavaApplication -->
select name of the class --> Test --> select arguments Tab -->
then give parameters as below (hello java world)
Apply and run
Displaying command line parameters using for-each loop:

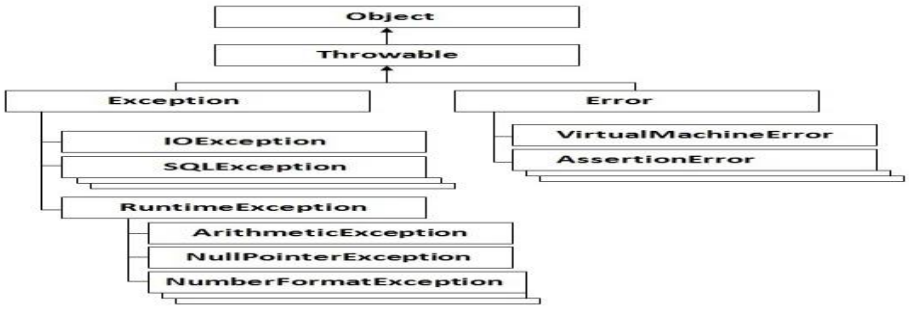
```
public class Test{
        public static void main(String[] args){
                for (String word : args) {
                        System.out.println (word);
                }  }}
```

execution:



Exception Handling

Error : It is a compile time one. Such as syntactical errors.
Exception : It is a runtime one, it is an unknown instruction to processor at run time. when exception raises application terminates abnormally. It can be handled by writing exception handling code.
Bug : It is an unexpected value at runtime. Even bug raises application never terminates abnormally. It can be fixed.
Defect : It can't be fixed. Some features may not be provided by the application software. It is manufacturing defect.
The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.
Hierarchy of Java Exception classes



What is an exception
Dictionary Meaning: Exception is an abnormal condition.
In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
What is exception handling.
Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.
Advantage of Exception Handling
The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling.
Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
```

Suppose there are 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the exception will be executed. That is why we use exception handling in java.

Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:
1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a= 50/0;//ArithmeticException
```

2) Scenario where NullPointerException occurs
default catch

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println (s.length());//NullPointerException
```

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="10ab2”;
int i=Integer.parseInt(s);//NumberFormatException
```

4) Scenario where ArrayIndexOutOfBoundsException occurs
If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[] = new int[5];
a[5]=50; //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords
There are 5 keywords used in java exception handling.
try , catch, finally, throw and throws
Java try-catch
Java try block is used to enclose the code that might throw an exception. It must be used within the method.
Java try block must be followed by either catch or finally block.
Syntax of java try-catch

```
try
{       ------------------------;
        ------------------------;  //code that may throw exception
}
catch(Exception_Type    arg)
{
        ---------------------;
        ---------------------;
}
```

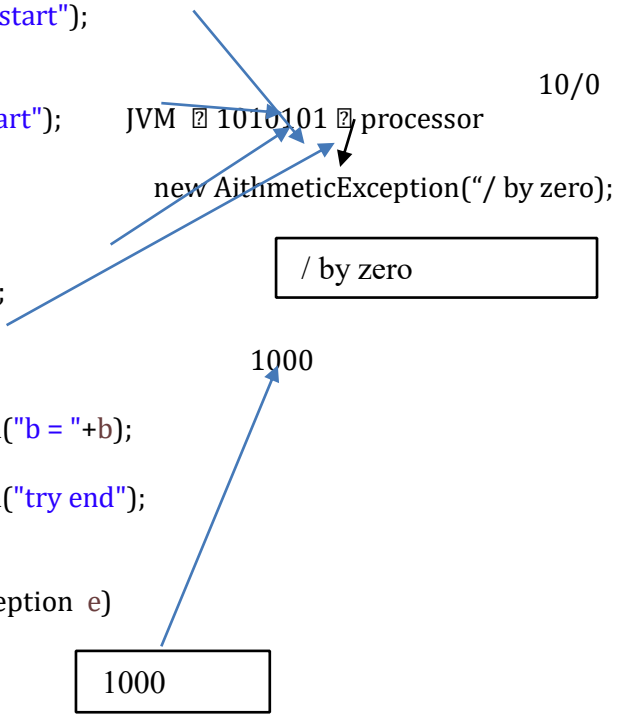

Example :

```
public class Test {
        public static void main(String[] args) {
                System.out.println("main start");
                Try  {
                        System.out.println("try start");
                        int a = 10;
                        int n = args.length;
                        int b = a/n;
                        System.out.println("b = "+b);
                        System.out.println("try end");
                }
                catch(ArithmeticException e)  {
```

```
                System.out.println("divide by zero error");
            }
            System.out.println("main end");
        } }
```

Run above program with Command line parameters:
RunAs → RunConfiguration →double click on JavaApplication → arguments tab → hello java
Try with parameters and also try without parameters.
For the above program , back end process as below

```
package com.durga.mnrao.xyz;
public class Test {
    public static void main(String[] args) {
            System.out.println("main start");
            try
            {
            System.out.println("try start");

                int a = 10;

                int n = args.length;

                int b = a/n;

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

                System.out.println("try end");

            }
            catch(NumberFormatException  e)
            {

                System.out.println("divide by zero ");
            }
            System.out.println("main end");
        } }
```



Example : to get JVM message on Exception

```
public class Test{
    public static void main( String  [] args ){
            System.out.println ("Main Start");
            try{
                System.out.println ("tray start");
                int a=10;
                int b;
                int n=args.length;
                b=a/n;
                System.out.println ("result "+b);
                System.out.println ("tray end");
            }
            catch(ArithmeticException  e)
            {
               String msg = e.getMessage();// this is to get message from JVM.
              System.out.println("divide by zero error "+msg);
              e.printStackTrace(); // gives location ( line number ) of exception.
            }
            System.out.println ("main end");
    }
}
```

Try with multiple catch blocks:

```
try
{
        ----------;
        ----------;
}
catch ( )
{ }
catch ( )
{ }
catch ( )
{ }
catch ( )
{ }
```

Example for try with multiple catch blocks:

```
public class Test{
    public static void main(String[] args){
        System.out.println ("Beg of main");
        try {
            int a = 10;
            int n = args.length;
            int b = a / n;
            System.out.println ("Result ; " + b);
            System.out.println ("First args:" + args[0]);
            System.out.println ("Second args:" + args[1]);
            System.out.println ("Third args:" + args[2]);
            System.out.println ("End of try");
        }
        catch(ArithmeticException   e)                  {
            String msg = e.getMessage();// this is to get message from JVM.
            System.out.println("divide by zero error "+msg);
            e.printStackTrace(); // gives location ( line number ) of exception.
        }
        catch(ArrayIndexOutOfBoundsException   e)  {
            String msg = e.getMessage();// this is to get message from JVM.
            System.out.println("invalid index "+msg);
            e.printStackTrace(); // gives location ( line number ) of exception.
        }
        catch(NumberFormatException   e)    {
            String msg = e.getMessage();// this is to get message from JVM.
            System.out.println("Non bumeric value"+msg);
            e.printStackTrace(); // gives location ( line number ) of exception.
        }
        System.out.println ("End of main");
    } }
```

Another example to submit emp information :

runAs runConfigiration arguments -> 1001 mnrao 15000 male 32

```
public class Test {
    public static void main(String[] args) {
        System.out.println("main start");
        try{
            System.out.println("try start");
            int a = 10;
            int n = args.length;
            int b = a/n;
            int empNum =  Integer.parseInt(args[0]);
            String empName = args[1];
            double  empSalary = Double.parseDouble( args[2] ) ;
            String empGender = args[3];
```

```
                int empAge =  Integer.parseInt(args[4]);
                System.out.println("below are emp details");
                System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empGender+"\t"+empAge);
        }
        catch(ArithmeticException e)  {
                System.out.println("emp recorfd missing, please submit");
        }
        catch(ArrayIndexOutOfBoundsException e)    {
                System.out.println("Missing some fileds");
        }
        catch(NumberFormatException e){
                System.out.println("non numeric value received ");
        }
        catch(NullPointerException e){
                System.out.println("null reference ");
        }
        System.out.println("main end");
    }  }
```

Try with default catch:

default catch :

```
catch (Exception   e ){
}
```

Example for default catch:

```
public class Test{
    public static void main(String[] args){
        System.out.println ("Beg of main");
        try{
                int a = 10;
                int n = args.length;
                int b = a / n;
                System.out.println ("Result ; " + b);
                System.out.println ("First args:" + args[0]);
                System.out.println ("Second args:" + args[1]);
                System.out.println ("Third args:" + args[2]);
                System.out.println ("End of try");
        }
        catch (ArithmeticException e) {
                System.out.println ("devide by zero exception");
        }
        catch (NullPointerException e){
                System.out.println ("Null  Pointer  Exception");
        }
        catch (Exception e ){
                System.out.println ("default exception");
        }

        System.out.println ("End of main");
    }  }
```

How **Exception** class, can handle any type of excepton.

Ans:

Since it is a parent of all Exception classes, it can hadle all kinds of exceptions.

Implementing Polymorphism.

default catch must be a last catch block.

below is the compile time error.

```
        try
        {
                ----------;
                ----------;
        }
```

```
        catch (ArithmeticException e  )
        {}
        catch (Exception e ) // default catch
        {}
        catch (NullPointerException e )  // compile time error, unreachable code
        {}
        catch (ArrayIndexOutOfBoundsException e )  )  // compile time error, unreachable code
        {}
```

Program to handle **NumberFormatException**

```
package com.nrit.mnrao.test;
import java.util.Scanner;
public class Test {
    public static void main(String [] args) {
        System.out.println("Employee details");
        Scanner sc = new Scanner(System.in);
        int eno;
        String empName;
        double salary;
        String deptName;
        while(true)    {
            try{
                System.out.println("Emp Num : ");
                String inputData = sc.nextLine();
                eno = Integer.parseInt(inputData);
                break;
            }
            catch(NumberFormatException   e)    {
                System.out.println("Non numeric input data");
            }
        }
        System.out.println("emp name ");
        empName = sc.nextLine();
        while(true)             {
            try                         {
                System.out.println("Emp Salary : ");
                String inputData = sc.nextLine();
                salary = Double.parseDouble(inputData);
                break;
            }
            catch(NumberFormatException   e)                    {
                System.out.println("invalid salary ");
            }
        }
        System.out.println("dept name ");
        deptName = sc.nextLine();
        System.out.println("Your input details are ");
        System.out.println(eno+"\t"+empName+"\t"+salary+"\t"+deptName);
        sc.close();
    } }
```

Input :

```
Employee details
Emp Num : 1001
emp name :abc
Emp Salary : 5000
dept name :dev
```

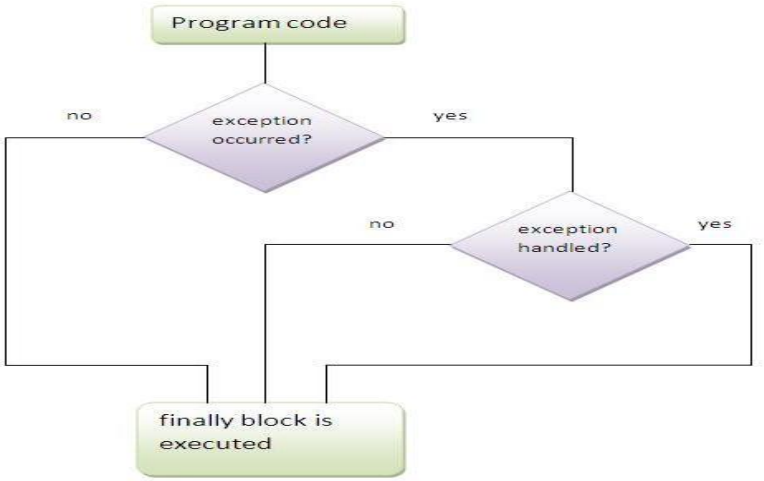
output:

```
Your input details are
1001    abc     5000.0 dev
```

Next time run, check the following:

Input data

Employee details
Emp Num : 10ab10
Non numeric input data
Emp Num : 10x20
Non numeric input data
Emp Num : 1001
emp name dev
Emp Salary : 5000,50
invalid salary
Emp Salary : 5000#50
invalid salary
Emp Salary : 5000.50
dept name :admin



Output:

Your input details are
1001 dev 5000.5 admin

Java finally block

Java finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not .Java finally block must be followed by try or catch block.

Finally block contain certain statement to be executed whether terminated normally or abnormally. These are compulsory statements. Mostly cleaning operations such as file closing, database connection closing, tcp sockets closing and etc.

Syntax of try-catch-finally block

Case 1:

```
try     {
}
catch (Exception e)
{                       // TODO: handle exception
}
finally {
}
```

Case 2:

try-finally

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

Below is not a valid one

```
try     {
}
finally {
}
catch (Exception e)
{                       // TODO: handle exception
}
```

Sequence must be **try, catch and finally**

Below is also not a valid one

```
catch (Exception e)     {
        // TODO: handle exception
}
finally{
}
```

To write catch / finally, try should be presented.

Multiple catch blocks can be placed but multiple finally blocks are not valid.

Example :

```
public class Test{
        public static void main(String[] args){
                System.out.println ("main start");
                try{
                        System.out.println ("Try start");
                        int a = 10;
                        int n = args.length;
                        int b = a / n;
                        System.out.println ("b= " + b);
                        System.out.println ("First param = " + args[0]);
                        System.out.println ("Second param = " + args[1]);
                        System.out.println ("Third param = " + args[2]);
                        System.out.println ("try end");
                }
                catch (ArithmeticException e) {
                        System.out.println ("arithmetic ");
                }
                catch (NumberFormatException   e)   {
                        System.out.println ("Array Index");
                }
                finally {
                        System.out.println ("I am in finally");
                }
                System.out.println ("main end");
        } }
```

Input :
Command line params are as below
hello java world
output (no exception and normally completed)

```
main start
Try start
b= 3
First param = hello
Second param = java
Third param = world
try end
I am in finally
main end
```

Input :
No command line params
output (exception raised and handled)

```
main start
Try start
arithmetic
I am in finally
main end
```

Input :
Command line params are as below
hello java
output (exception raised but not handled, terminated abnormally)

```
main start
Try start
b= 5
First param = hello
```

Second param = java
I am in finally
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
at com.nrit.mnrao.test.Test.main(Test.java:26)
there are two senarios

1) Normal termination
2) Abnormal termonation

Normal termination:
Any statements after finally block will also be executed.
Abnormal termination:
Any statements after finally block will not be executed.
Java Nested try block
The try block within a try block is known as nested try block in java.

Why use nested try block
Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
Syntax:

```
try{
        ----------;
        ----------;
        try     {
                ----------;
                ----------;
        }
        catch ( ){
        }
        catch ( ){
        }
        ----------;
        ----------;
}
catch ( ){
}
catch ( ){
}
----------;
---------;
```

Cases:
Case 1:
No exception in the out try as well as inner try, then no catch block executes and program completed till end, then terminates.
Case 2:
Exception in the outer try
Checks for the outer catch, if matched then, that catch blocks executes, if not matched, then termiates abnormally.
Case 3:
Exception in the inner try

Checks for the inner catch, if matched then, that catch blocks executes, then goes to outer try ending statements, then terminates normally.

If not matched with inner catch, then checks for outer catch blocks, if matched then, that catch blocks executes, then terminates normally. if not matched, then termiates abnormally.

```
public class Test {
        public static void main(String[] args){
                System.out.println ("main start");
                try     {
                        System.out.println ("outer try start");
                        int a=10;
                        int n=args.length;
```

```
                int b = a/n;
                System.out.println ("b="+b);
                try{
                        System.out.println ("Innner try start");
                        System.out.println ("First param "+args[0]);
                        System.out.println ("second param "+args[1]);
                        System.out.println ("third param "+args[2]);
                        System.out.println ("innner try end ");
                }
                catch(ArrayIndexOutOfBoundsException e)    {
                        System.out.println ("Inner try array index exception ");
                }
                finally {
                        System.out.println ("Inner try finally");
                }
                System.out.println ("outer try end ");
        }
        catch(ArithmeticException e) {
                System.out.println ("out side try divide by zero error");
        }
        finally {
                System.out.println ("outer try finally");
        }
        System.out.println ("main end");
} }
```

Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).

Why use java finally

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Nested try with Finally:

```
public class Test{
        public static void main(String[] args) {
                System.out.println("main start ");
                try{
                        System.out.println("outer try start");
                        int a = 10;
                        int n = args.length;
                        int b = a / n;
                        System.out.println(" b = "+b);
                        try{
                                System.out.println("Inner try start ");
                                System.out.println("First param = "+args[0]);
                                System.out.println("Second param = "+args[1]);
                                System.out.println("Third param = "+args[2]);
                                System.out.println("Inner try end");
                                }
                        catch(ArrayIndexOutOfBoundsException e)    {
                                System.out.println("inner try Invalid index");
                        }
                        catch(NumberFormatException e){
                                System.out.println("outer try non numeric value");
                        }
                        finally {
                                System.out.println("I am in inner finally");
                        }
                        System.out.println("outer try end ");
                }
                catch(ArithmeticException e) {
                        System.out.println("outer try Divide by zero");
```

```
                }
                catch(NullPointerException e){
                        System.out.println("Inner try null reference");
                }
                finally {
                        System.out.println("I am in outer finally");
                }
                System.out.println("main end");
        }   }
```

Java throw keyword

The Java **throw** keyword is used to throw an exception explicitly.

We can throw either checked or uncheked exception in java by throw keyword. throw keyword is mainly used to throw custom exception.

The syntax of java throw keyword is given below.

```
throw  new throwableInstance();
```

throwableInstance must be a child of **Exception** class.

Eg:

```
throw  new IOException("sorry device error”);
```

java throw keyword example

eg:

```
package com.nrit.mnrao.test;
public class Test {
        public static void main(String[] args){
                System.out.println ("Main Start");
                try{
                        System.out.println ("tray start");
                        int a=10;
                        int b;
                        int n=args.length;
                        if(n==0){
                                throw new ArithmeticException();
                        }
                        b=a/n;
                        System.out.println ("b="+b);
                }
                catch(ArithmeticException  e) {
                        System.out.println( e.getMessage() );
                }
                System.out.println ("main end");
        }  }
```

In the above example getMessage() **return value is null.**

If exception instance created by JVM implicitly, then JVM will initialize instance with text message. Then getMessage() returns text message initialized by the JVM.

In the above example**,** java developer**,** has created exception instance but not initialized with any text message. Hence it returns null value.

Creating exception instance with text message.

```
                throw new ArithmeticException("divide by zero ");
```

getMessage() returns value : divide by zero

Main purpose of throw key word is to throw user defined exceptions explicitly

Defining User defined exceptions / custom Exceptions:

user defined exceptions class should extends of **“Exception”** class.

```
package com.nrit.mnrao.test;
public class SampleException extends Exception{
        private String message;
        public SampleException(){
                message=null;
```

```
	}
	public SampleException(String message){
		this.message=message;
	}
	@Override
	public String getMessage() {
		// TODO Auto-generated method stub
		return message;
	}
	@Override
	public String toString() {
		// TODO Auto-generated method stub
		return message;
	}  }
public class Test{
	public static void main(String[] args) {
		System.out.println ("main start");
		try {
			System.out.println ("try start");
		int a = 10;
			int n = args.length;
			if (n == 0) {
				throw new SampleException("Sample user defined exception");
			}
			int b = a / n;
			System.out.println (" b = " + b);
			System.out.println ("First parameter " + args[0]);
			System.out.println ("second parameter " + args[1]);
			System.out.println ("Third parameter " + args[2]);
			System.out.println ("try end");
		}
		catch (SampleException   e) {
			System.out.println ("Arithmetic   Exception");
		  System.out.println ( e.getMessage() );//makes call to method of SampleException class
		  System.out.println (  e );//it makes a call to toString() method  of SampleException calss;
		}
		System.out.println ("end of main");
	}  }
```

Rules to define Custom Exceptions

1) Custom exception class should be a child of **Exception** class
2) It should have one private message variable, to store customized message
3) It should have default and parameterised constructors to initialize message variable
4) Define, overriding toString() , to override Throwable class toString()
5) Define, overriding getMessage() , to override Throwable class getMessage ()
6) Throwable instance must be child of Exception class.

Purpose of user defined exceptions, is to generate customized message (User own messages)

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1) Exception
   Checked Exception
   Unchecked Exception
2) Error

Difference between checked and unchecked exceptions

1) **Checked Exception**

Checked exceptions are checked at compile-time.

The classes that extends of Throwable / Exception classes except RuntimeException and Error are known as checked exceptions e.g.

IOException, SQLException, User defind Exception classes.
All Exception classes, from other than java.lang package are checked exceptions.

2) Unchecked Exception

Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
The classes, that extends RuntimeException e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
All the Exception classes from java.lang package are Unchecked Exceptions

Q. User defined Exceptions, should be difined under which exception

checked / unchecked ?

Ans:

Checked ,
reason it is not from java.lang package
it is having it's own package (user defined package)

3) Error

Error is irrecoverable
e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.
Checked exceptions should handled at compile time. It can be handled by using try –catch or usng throws clause.
try –catch is used to get exception message, where as throws clause is to suppress Exception at compile time.

Java throws keyword (throws clause)

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
return_type  method_name()  throws  exception_class_name
{
    //method code
}
```

checked exception, should be handled either using **try-catch** or using **throws** declaration

Java throws example

```
public class Test {
    public static void main(String[] args) throws SampleException {
        System.out.println("Main Start");
        System.out.println("tray start");
        int a = 10;
        int b;
        int n = args.length;
        if (n == 0) {
            throw new SampleException("Sample error");
        }
        b = a / n;
        System.out.println("b=" + b);
        System.out.println("main end");
    }  }
```

Another example

```
public class Test {
    public static void main(String[] args) throws SampleException {
        System.out.println("Main Start");
        display(args);
        System.out.println("main end");
    }
    public static void display( String[] args) throws SampleException
    {
        System.out.println("tray start");
        int a = 10;
        int b;
```

```
                int n = args.length;
                if (n == 0) {
                        throw new SampleException("Sample error");
                }
                b = a / n;
                System.out.println("b=" + b);
        }  }
```

If called method is having throws clause, then calling methods also must have throws clause or it should hadeld by using try-catch.

Throws clause with multiple Exceptions

```
public static void main(String[]args) throws IOException, SampleExeption, SQLException {
                System.out.println("Main Start");
                display(args);
                System.out.println("main end");
        }
```

Throws clause handling multiple Exceptions (with default Exception)

```
public static void main(String[]args) throws Exception {
                System.out.println("Main Start");
                display(args);
                System.out.println("main end");
        }
```

If called method has throws clause then, calling method also must have throws clause.

```
public class Test {
        public static void main(String[] args) throws SampleException {
                System.out.println("Main Start");
                display(args);
                System.out.println("main end");
        }
        public static void display( String[] args) throws SampleException {
                show(args);
        }
        public static void show(String[] args)throws   SampleException{
                System.out.println("tray start");
                int a = 10;
                int b;
                int n = args.length;
                if (n == 0) {
                        throw  new SampleException("Sample error");
                }
                b = a / n;
                System.out.println("b=" + b);
        }  }
```

Rising exception in the Called method and hadling local to that method.

```
public class Test {
        public static void main(String[] args)  {
                System.out.println("Main Start");
                        display(args);
                System.out.println("main end");
        }
        public static void display( String[] args)        {
                try{
                        System.out.println("tray start");
                        int a = 10;
                        int b;
                        int n = args.length;
                        b = a / n;
                        System.out.println("b=" + b);
                }
```

```
		catch(ArithmeticException e) {
			e.printStackTrace();
		} } }
```

Rising exception in the Called method and hadling inside the calling method.

```
public class Test {
	public static void main(String[] args) {
		System.out.println("Main Start");
		try{
			display(args);
		}
		catch(ArithmeticException e){
			e.printStackTrace();
		}
		System.out.println("main end");
	}
	public static void display( String[] args) {
		try	{
			System.out.println("tray start");
			int a = 10;
			int b;
			int n = args.length;
			b = a / n;
			System.out.println("b=" + b);
		}
		catch(NullPointerException e)
		{
			e.printStackTrace();
		} } }
```

Method re-throwing an exception :

```
public class Test {
	public static void main(String[] args) {
		System.out.println("Main Start");
		try{
			display(args);
		}
		catch(ArithmeticException e) {
			e.printStackTrace();
		}
		System.out.println("main end");
	}
	public static void display( String[] args) {
		try{
			System.out.println("tray start");
			int a = 10;
			int b;
			int n = args.length;
			b = a / n;
			System.out.println("b=" + b);
		}
		catch(ArithmeticException e){
			System.out.println("divide by zero error ");
			throw e;// re throwing exception
		} } }
```

Program to skip catch block .

```
public class Test {
	public static void main(String[] args) {
		System.out.println("Main Start");
		try{
			System.out.println("tray start");
```

```
                int a = 10;
                int b;
                int n = args.length;
                b = a / n;
                System.out.println("b=" + b);
        }
        catch(ArithmeticException e) {
                System.exit(0);
                System.out.println("divide by zero error ");
        }
        System.out.println("main end");
} }
```

Difference between throw and throws

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

throw	throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Throw key word is throw Checked exceptions.	Checked exception can be suppressed with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
We cannot throw multiple exceptions.	We can declare multiple exceptions e.g. public void method() throws IOException,SQLException.

Difference between final, finally and finalize

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

final	finally	Finalize
Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
Final is a keyword.	Finally is a block.	Finalize is a method.

1.can you differentiate between final, finally and finalize

Ans: **final** is an access modifier, it can be used with class, it's data members, methods and also with local variable.
Final data member is constant. Final local variables also constant. Final method can not be overridden in child class.
Final class prevents from inheritance.
Finally: it block for exception handling, it executes fot any conditions such, normally terminate or abnormally terminated.
Finalize: it is method from Object. It will be called automatically, when object is cleared by the Garbage collector.