

Introduction to Software

Software is collections of programs such as C , C++ , Java, Python, HTML 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

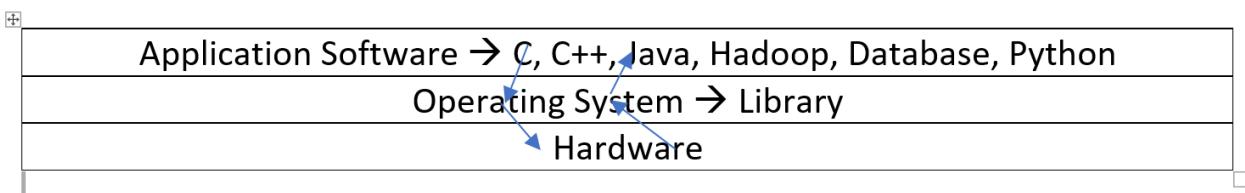
Booting is a process of loading OS from Hard Disk to RAM memory.

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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- 4) Every program has to link with **OS library** before going to run.



Application Software	
Front End Software	Back End Software
<p>These are to interact with end users, it collects data from end users and stores into backend Eg: Languages, Packages ERPs</p>	<p>These are used to store and maintain data Eg: Databases and files , HDFS (Bigdata storage)</p>

Front End Software	
Languages	Packages
<p>Programmer has to write coding. Complexity in development of application Eg: C, C++, JAVA, HTML, .Net</p>	<p>It is a readymade one, ready to use. Packages don't have programming MS-Office, ERP packages, Tally Accounting package</p>

Languages	
Low Level Languages	High Level Languages
<p>These are machine (processor) understandable languages Eg: Machine Language (IGL) and Assembly Language (II GL) Assembly language in the form of mnemonic codes Assembler converts assembly language into machine language</p>	<p>These are user understandable languages These look like English language Easy to understand. III GL and IV GL are high level languages</p>

Software Languages

1) Computer Understandable Language

2) User Understandable Language

Computer Understandable Language

that depends on Generation of Computers.

There are five generations

I Gen --> Vacuum Tubes based

II Gen --> Transistor based

III Gen --> IC based

IV Gen --> Processor based (present)

V Gen --> AI based (future)

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 English

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 a user understandable 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 mnemonic codes Assembler converts 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++ 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 (interprter)** → Binary Code → processor → Output

Source Code → Byte Code → Binary Code → Processor → Output

Source Code → Byte Code → **java compiler**

Byte Code → Binary Code → **java interpreter**

Byte code was introduced, to make java program as independent of platform

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 OS

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

4) It provides security for local resources.

JVM has Security Manager 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 converts 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:

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

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

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)

15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is released in March month and an odd version released in September month.

Java Installation :

Installing Java

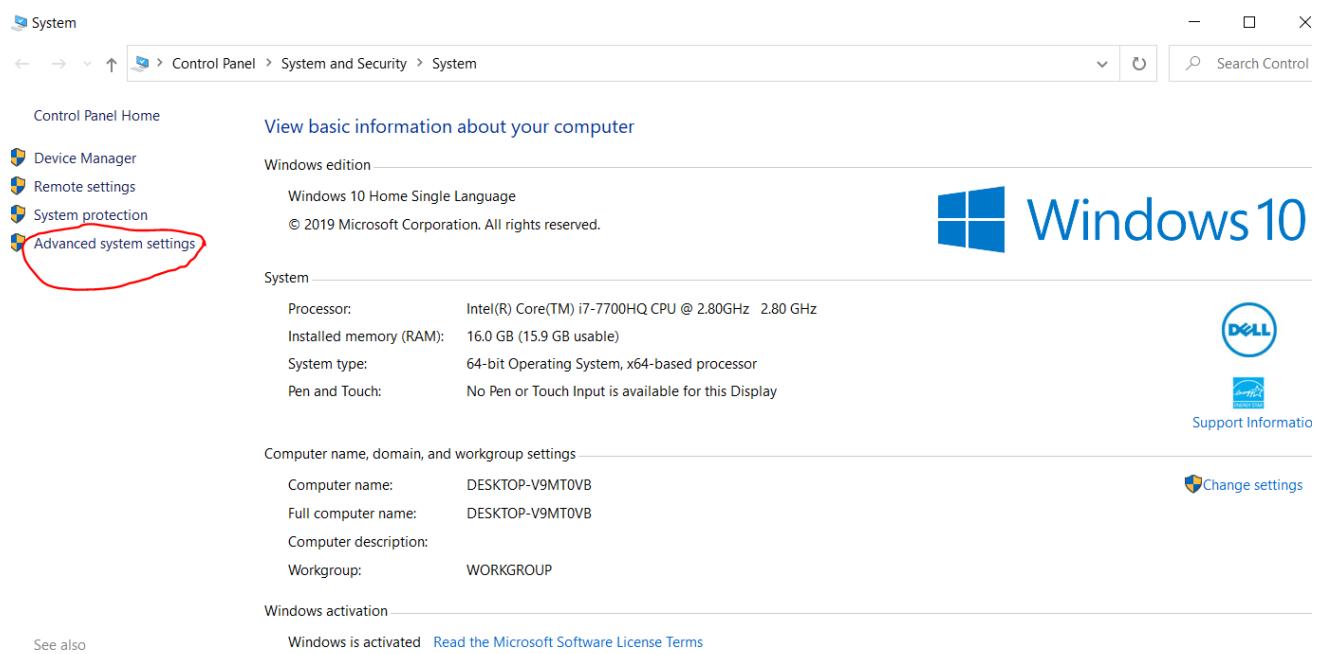
Copy JDK1.8 into C-Drive and double click → yes → next → next → close

Setting Environment Variable for Java

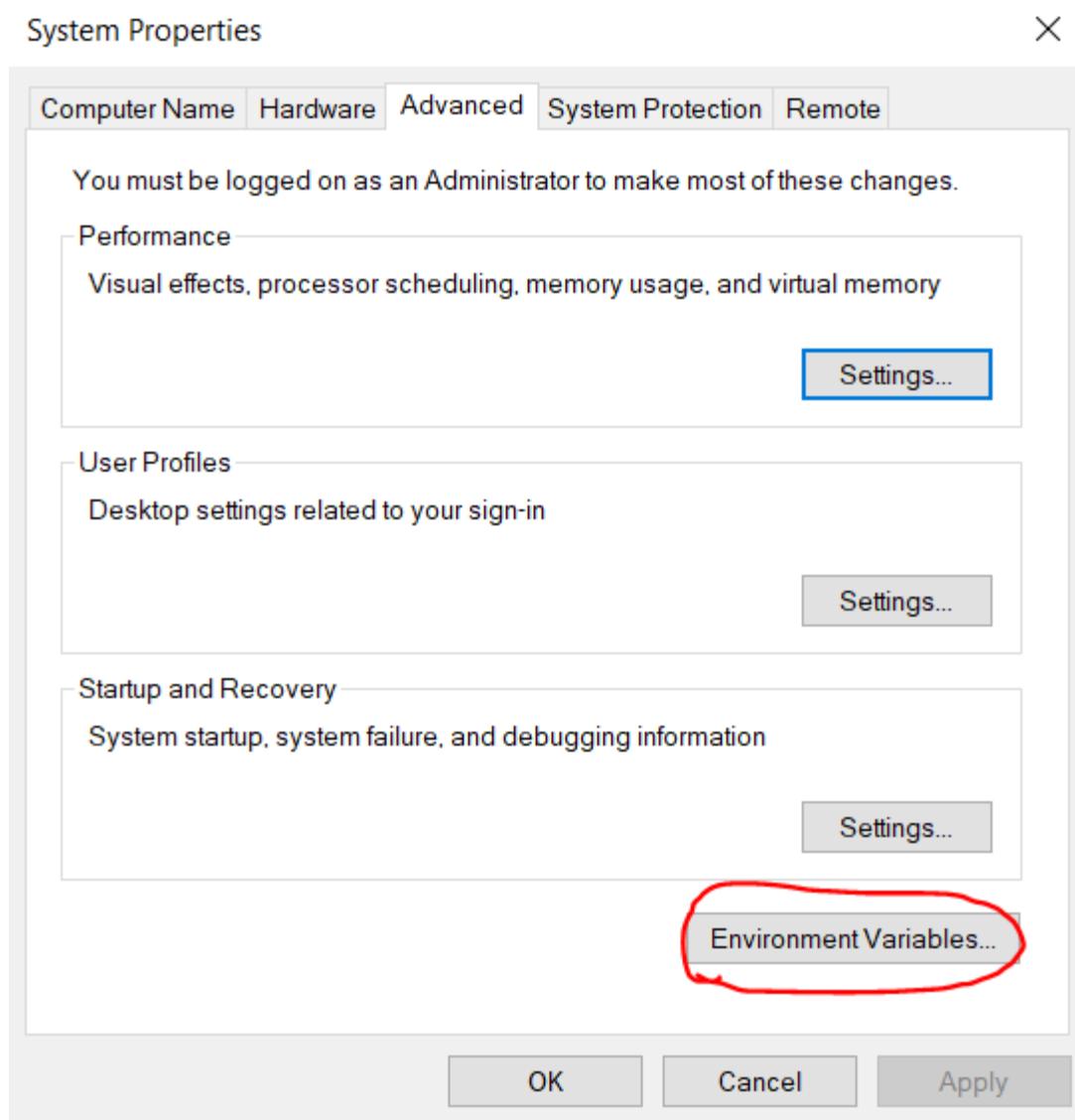
Win-10

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

Select Advanced System settings



Click on Environment variables as below



Goto System variables

Environment Variables

X

User variables for mnrao

Variable	Value
OneDrive	C:\Users\mnrao\OneDrive
OneDriveConsumer	C:\Users\mnrao\OneDrive
Path	C:\Users\mnrao\AppData\Local\Microsoft\WindowsApps;C:\P...
PyCharm Community Editi...	C:\Program Files\JetBrains\PyCharm Community Edition 2020....
TEMP	C:\Users\mnrao\AppData\Local\Temp
TMP	C:\Users\mnrao\AppData\Local\Temp

New...

Edit...

Delete

System variables

Variable	Value
CATALINA_HOME	C:\apache-tomcat-9.0.38
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;D:...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

New...

Edit...

Delete

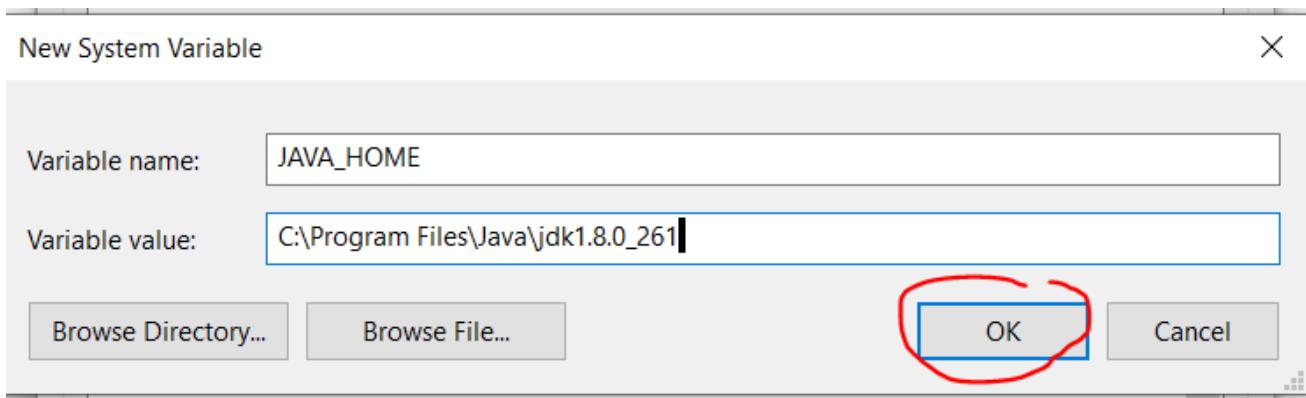
OK

Cancel

Variable Name : JAVA_HOME

Variable value : C:\Program Files\Java\jdk1.8.0_121

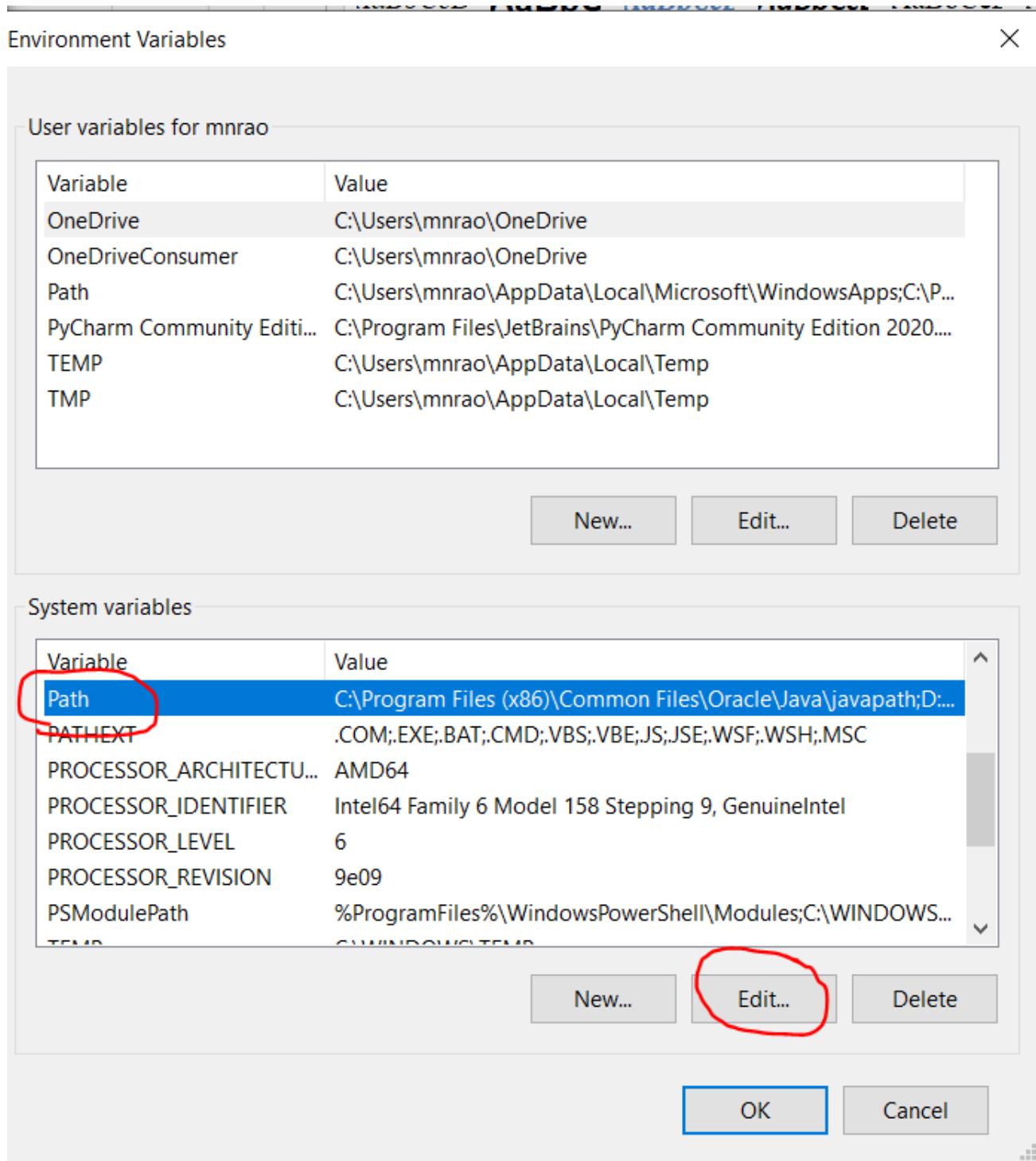
Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123



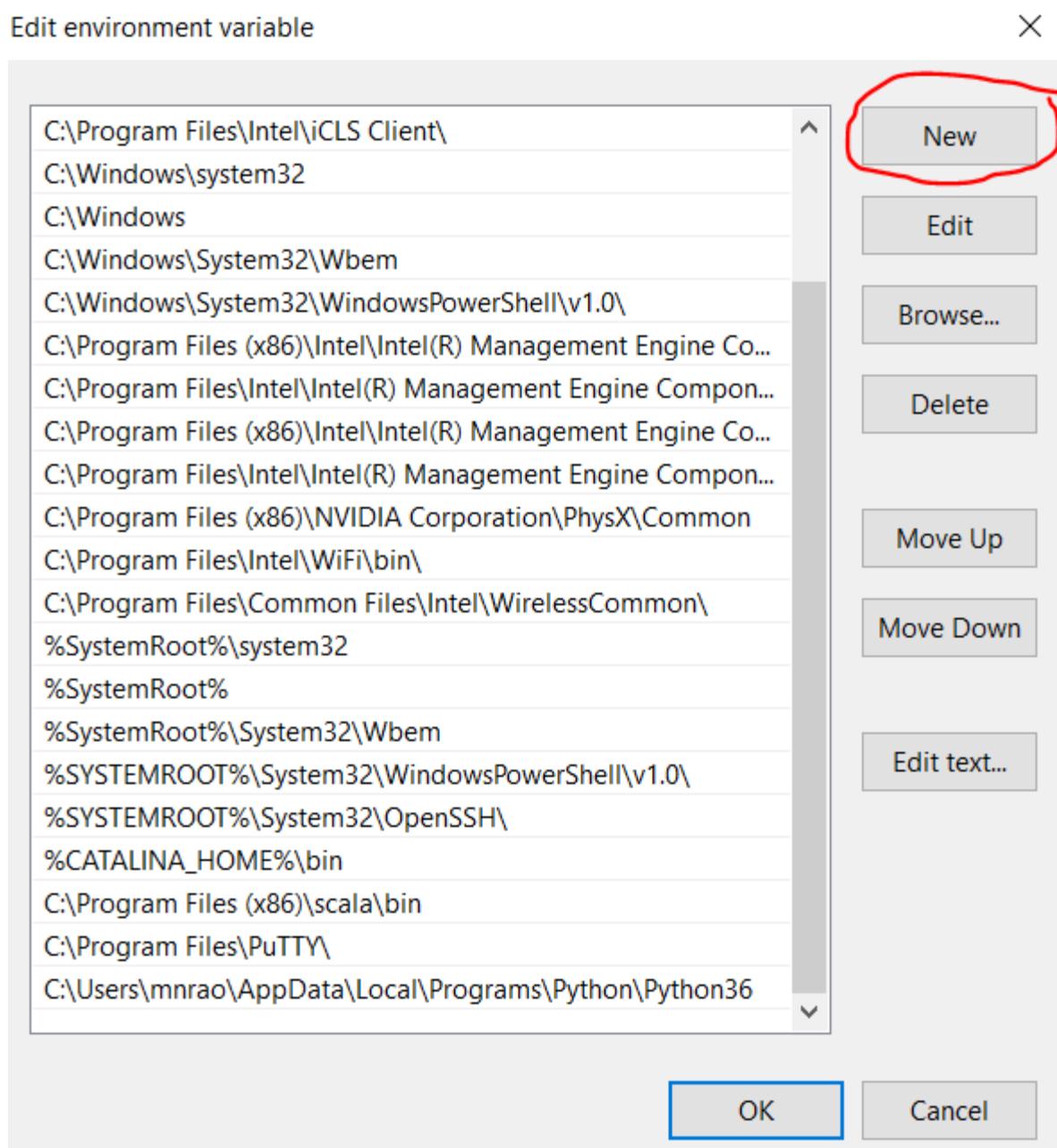
Setting path :

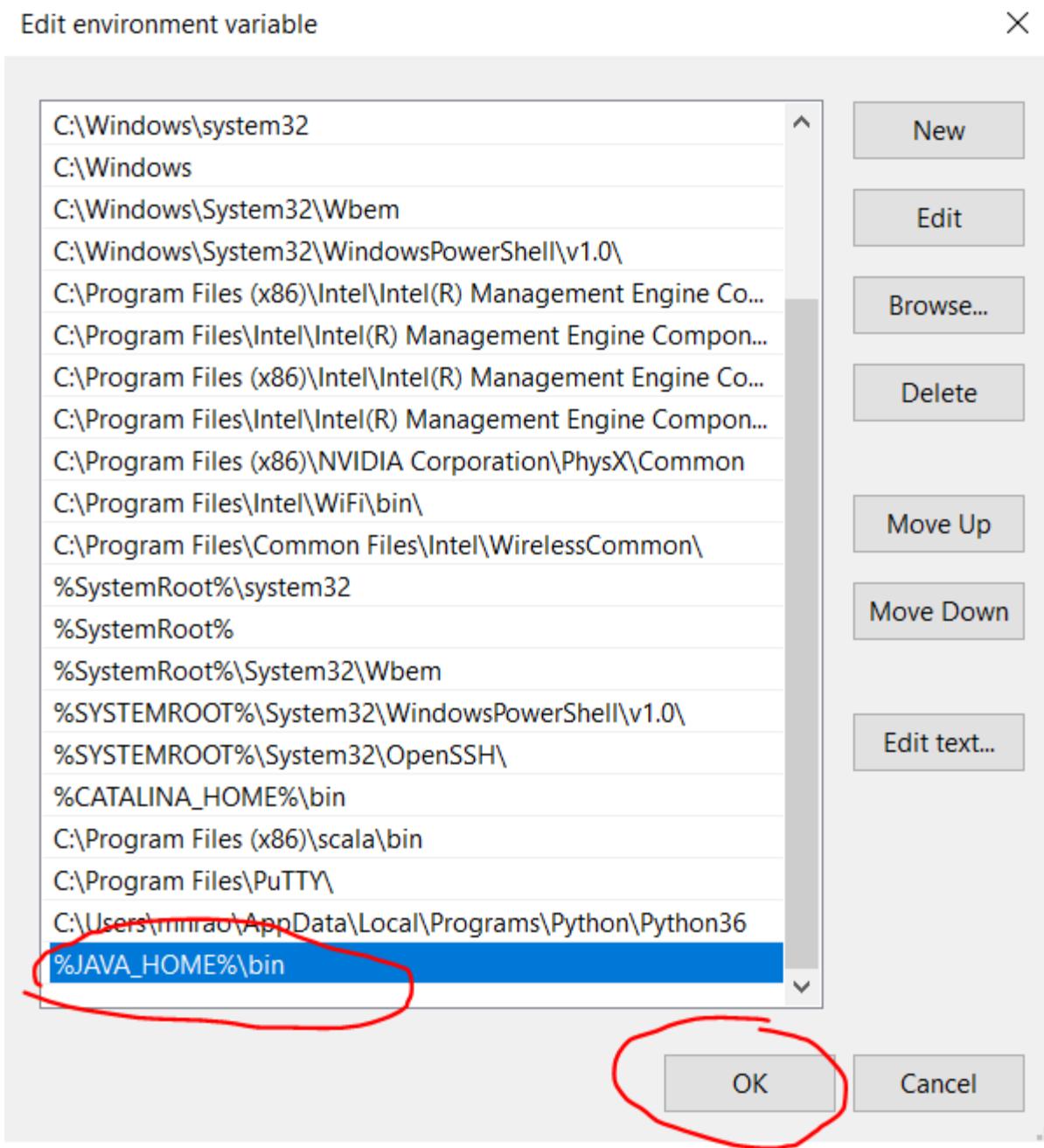
Goto system variables

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



Select New





Click on OK → OK → OK

Setting Java Path in Linux OS:

```
$gedit ~/.bashrc
```

go 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
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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)\am Files\Intel\iCLS Client\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\Win gram Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management rogram Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management \Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Comm on\;C:\WINDOWS\system32;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\PY\;C:\Program Files\Java\jdk1.8.0_121\bin;C:\Users\mnrao\AppData\Local\Microsoft\WindowsApps;C:\Program Fi nity Edition 2020.1\bin\;;C:\Users\mnrao\AppData\Local\Programs\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 and select 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

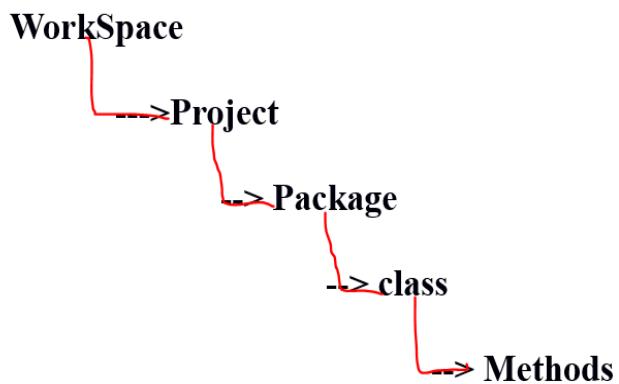
IDE: Integrated Development Environment.

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

Project Hierarchy :



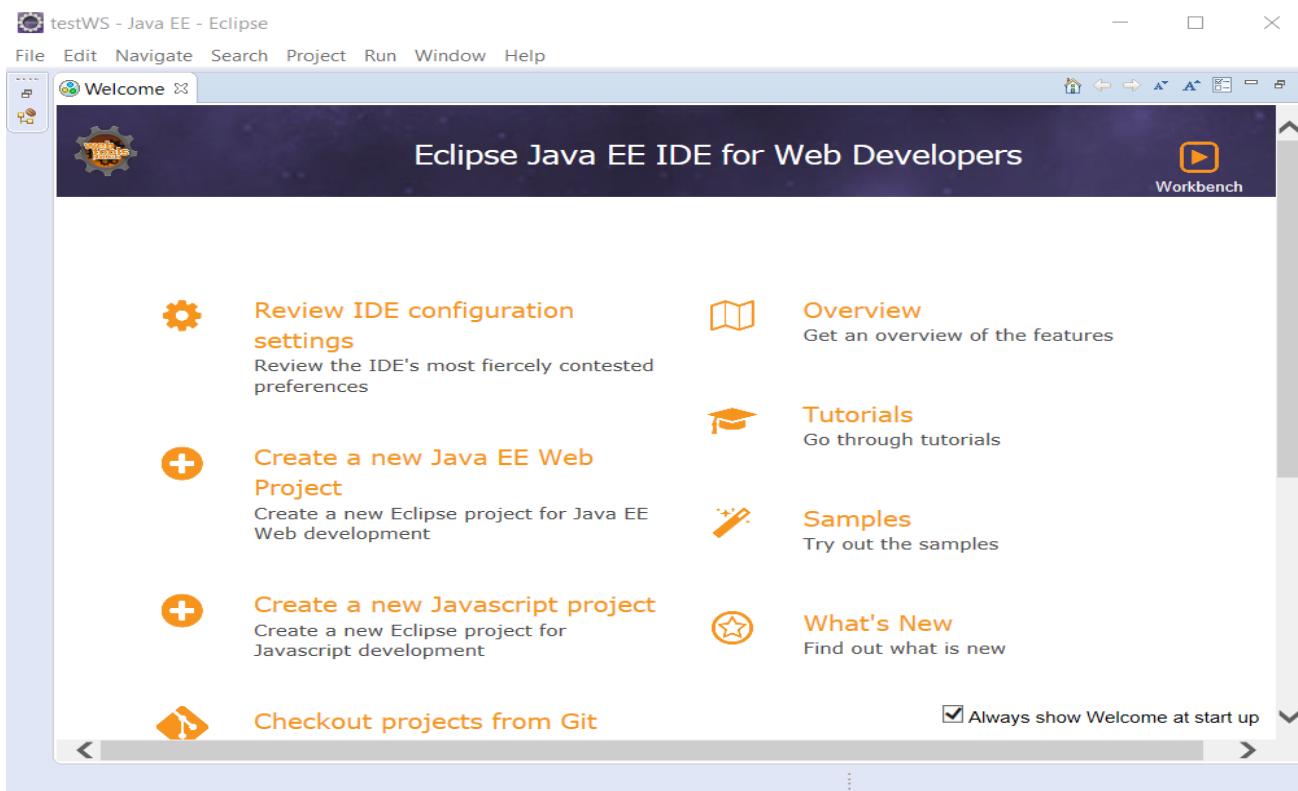
Step1:

Create Workspace

Goto Desk top and double click on Eclipse Icon, then Launch eclipse → Select Work Space location (browse) and provide the name , D:\test\SampleWS

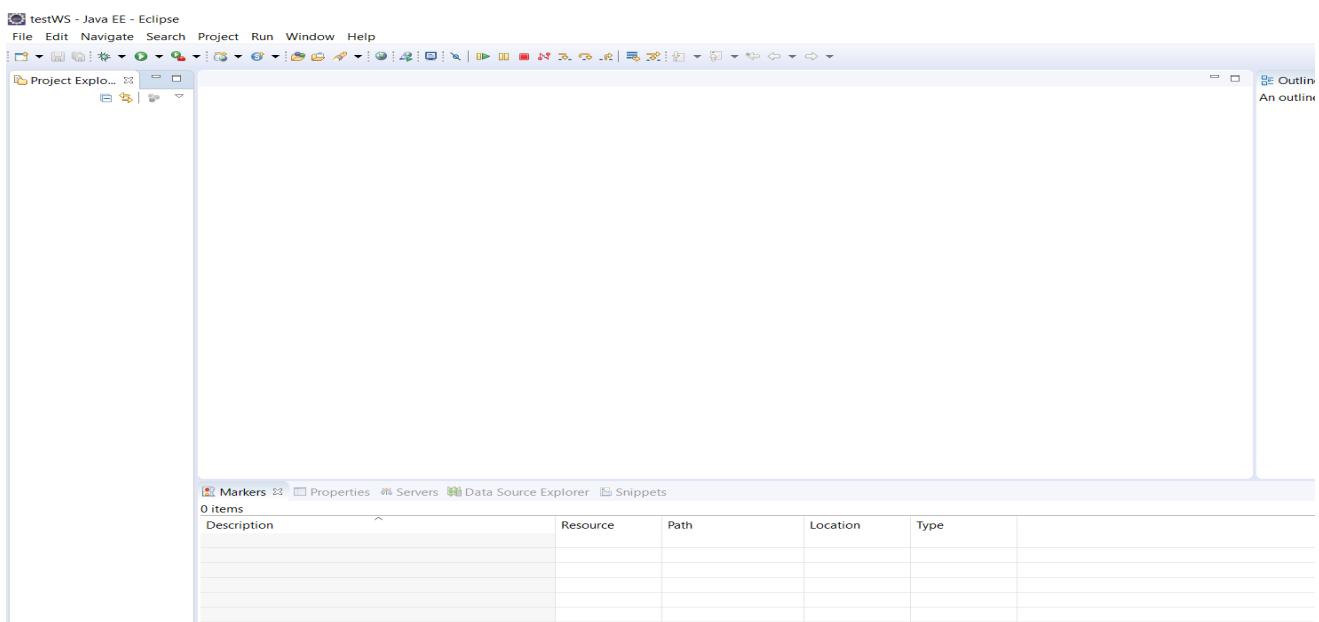
SampleWS → is a workspace name → Ok

Close below **welcome** screen.



Step2:

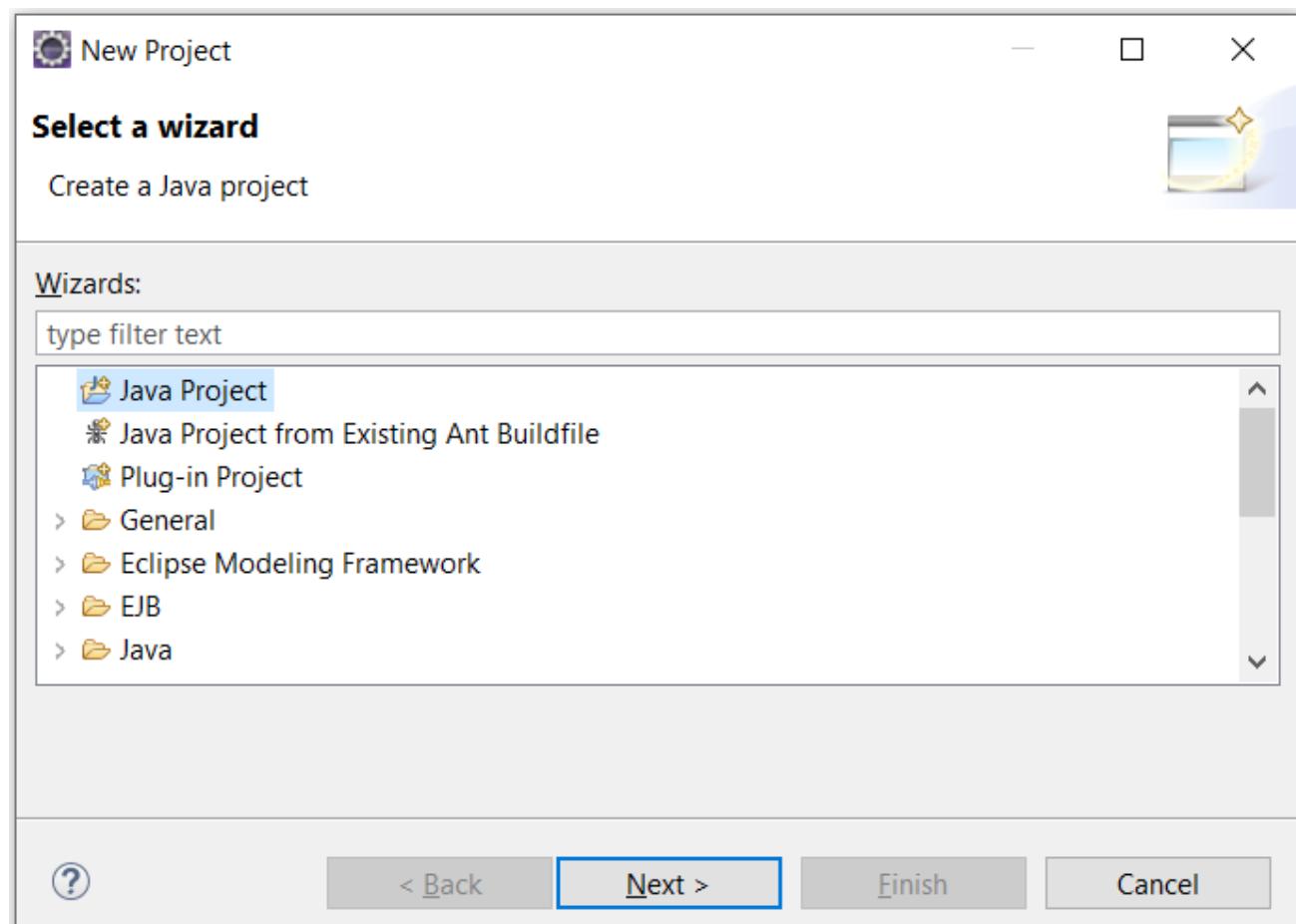
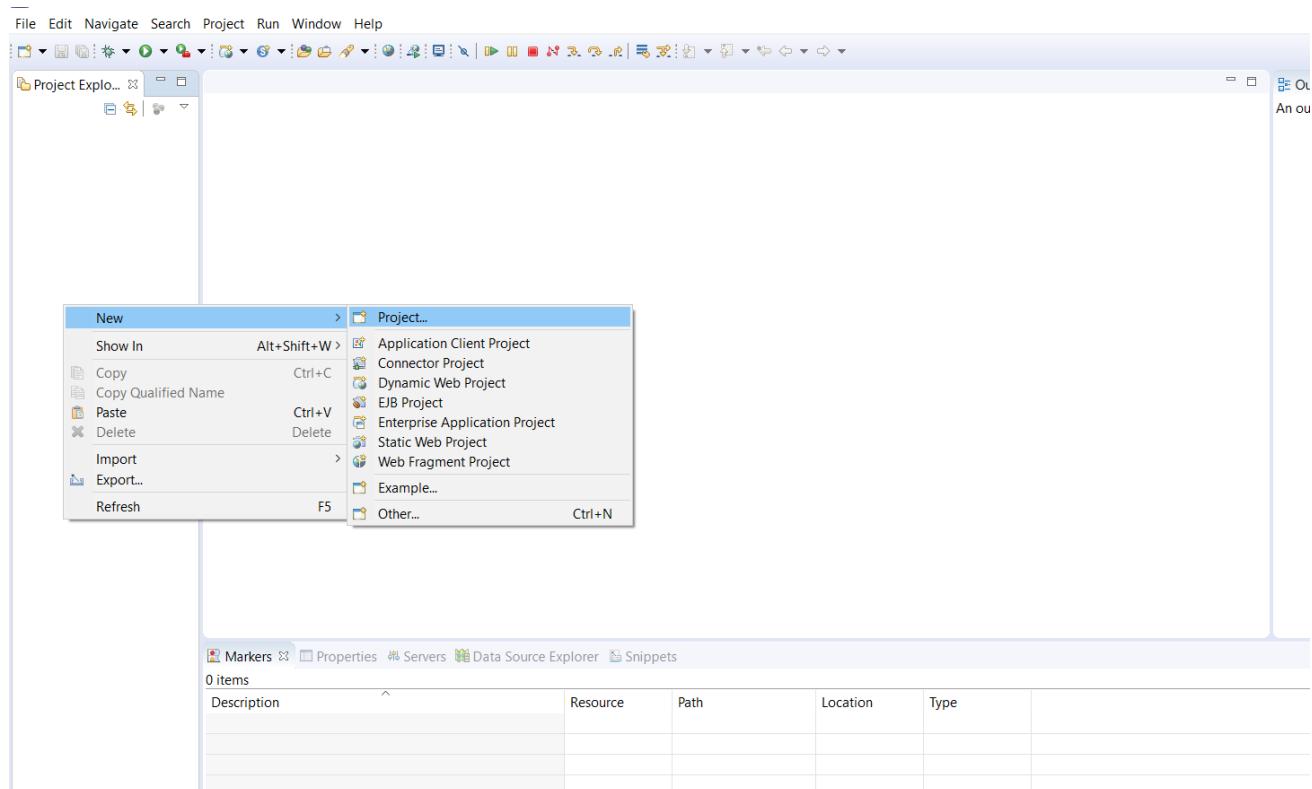
Create Project

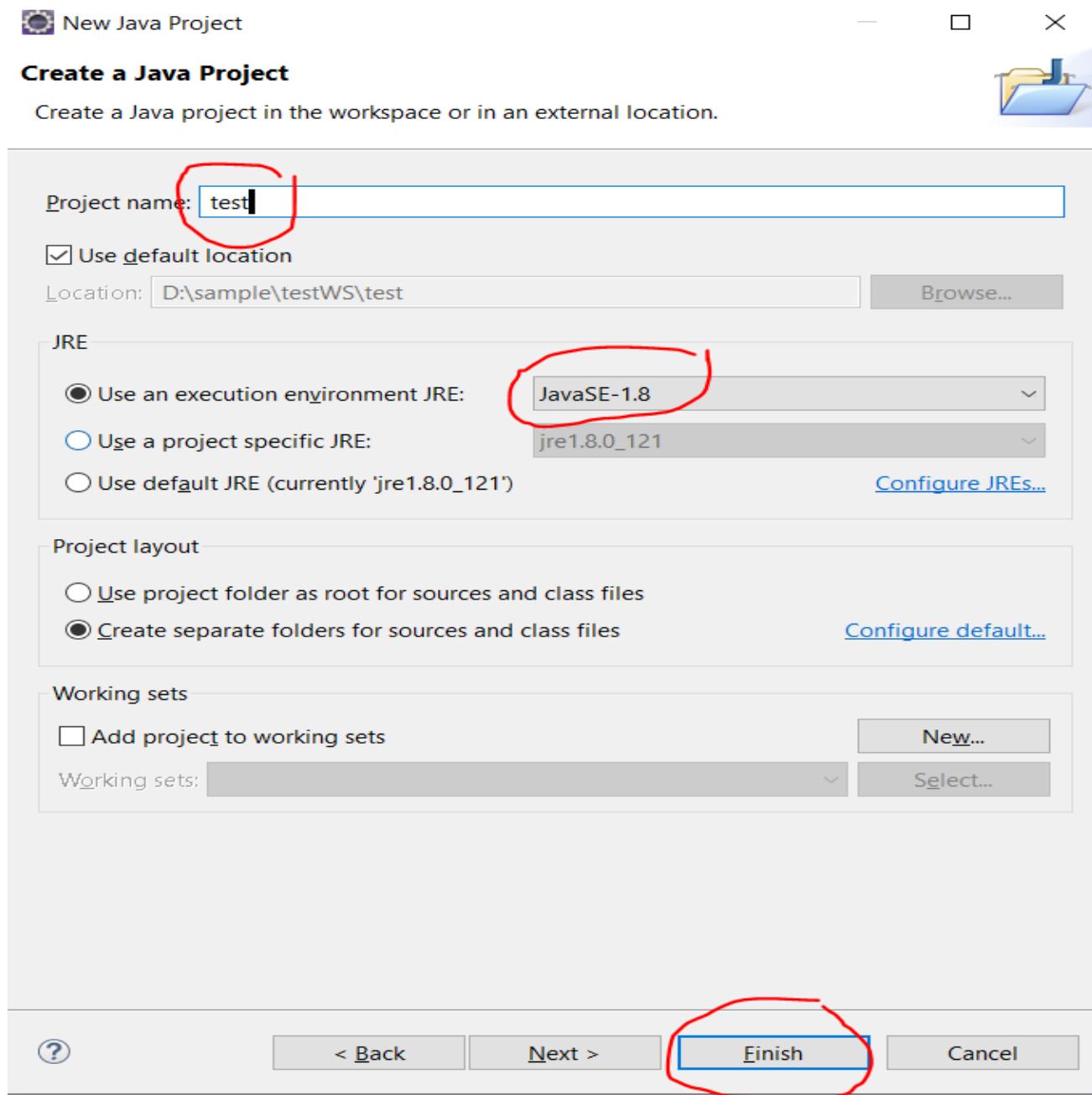


Rt.Click on Project Explorer area → new → project → java project → next
provide project name (test)
execution environment : JavaSE1.8

Finish

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

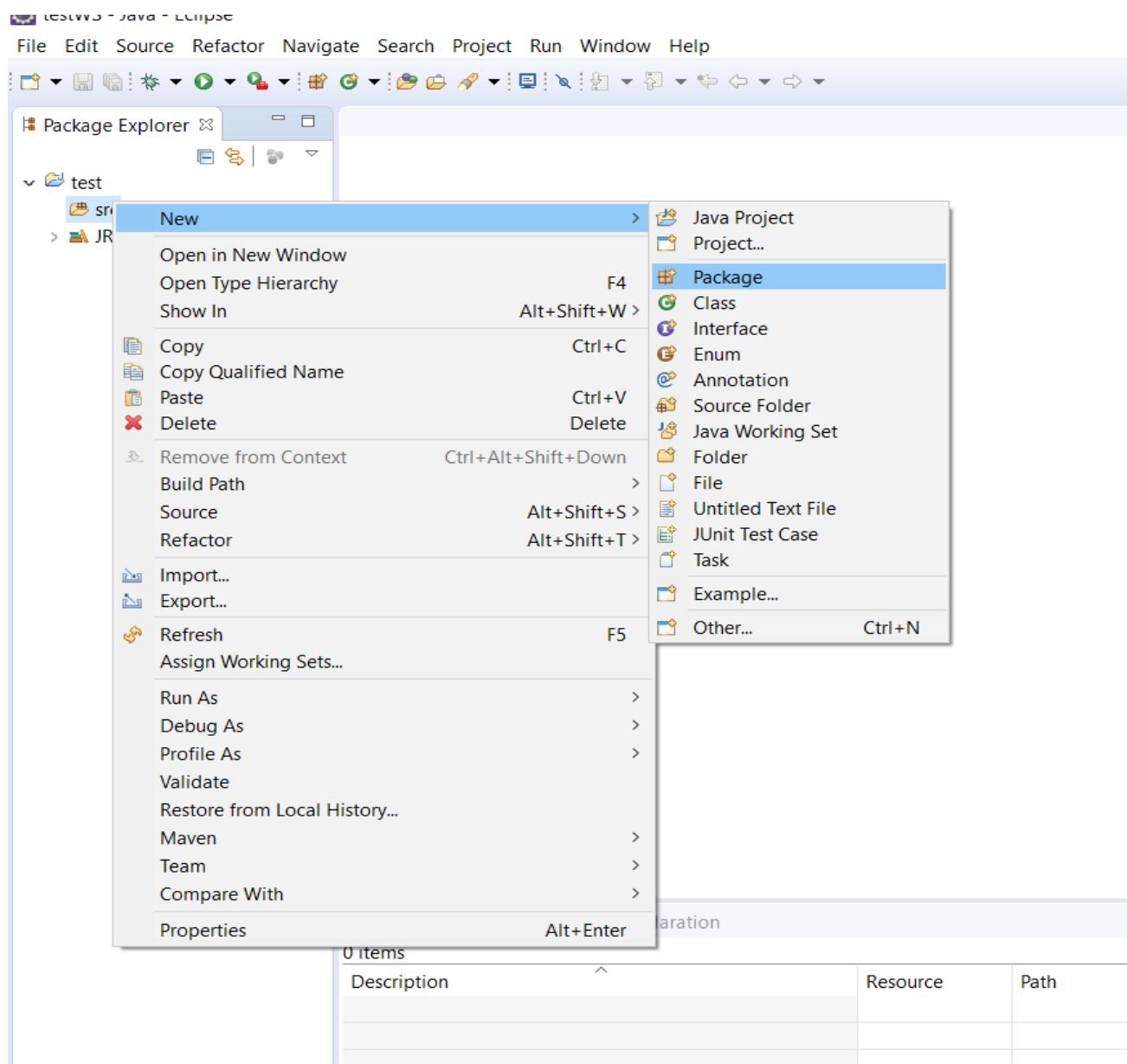




Step3:

Create a package

Expand project → rt.click on **src** → new → package →



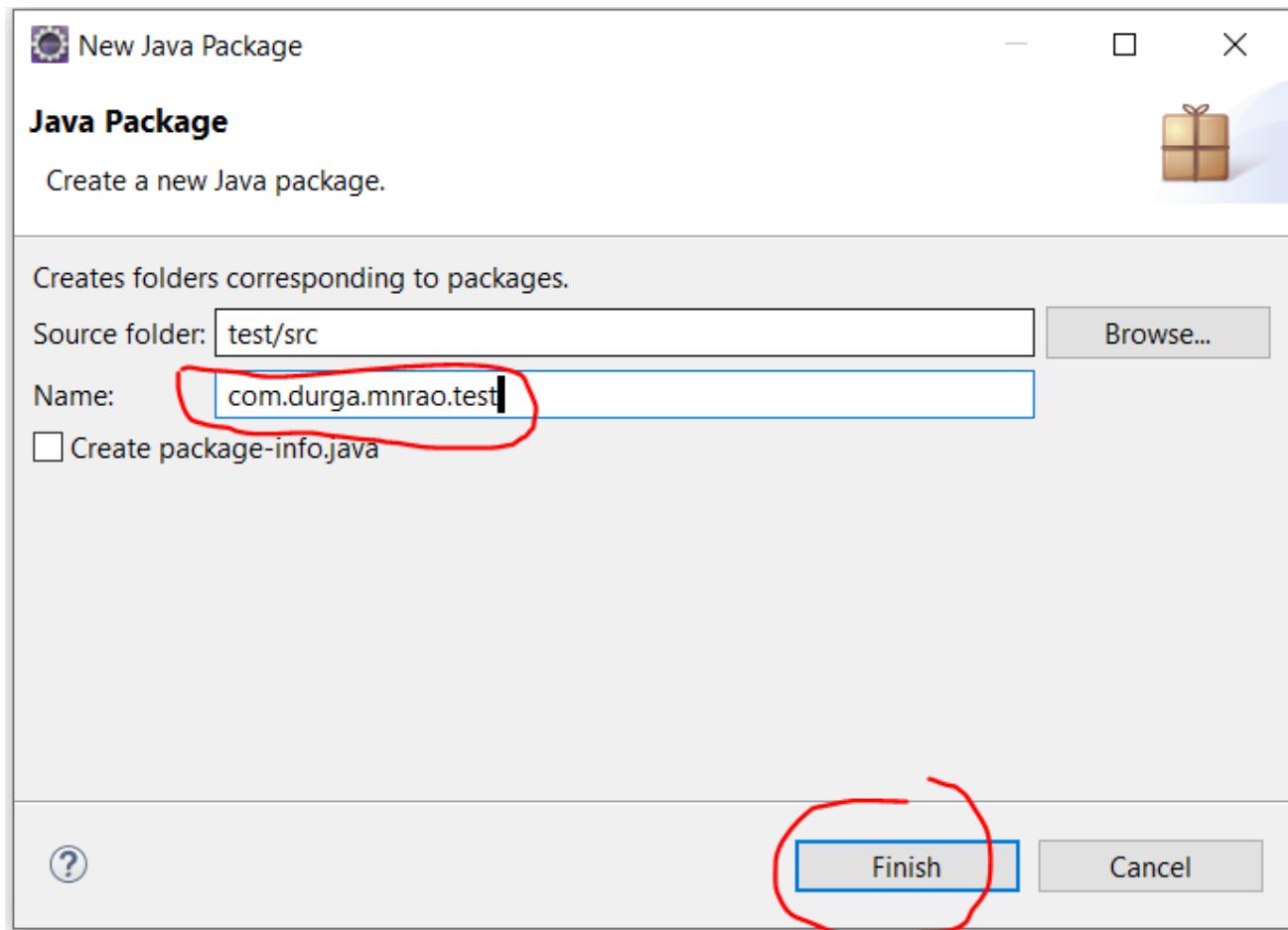
Package name (lower case) : <com/org>.<client_name>.<project_name>.<module_name>

com → Company

org → Organization

Eg:

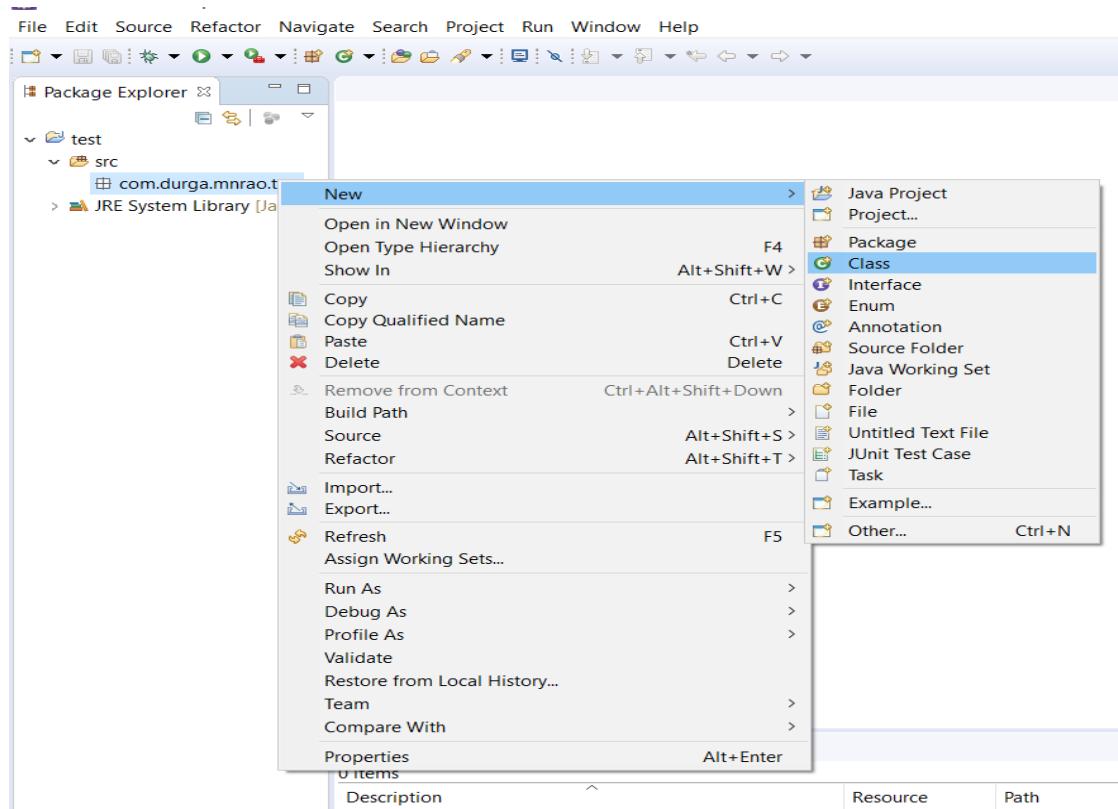
com.nr.it.track.admin



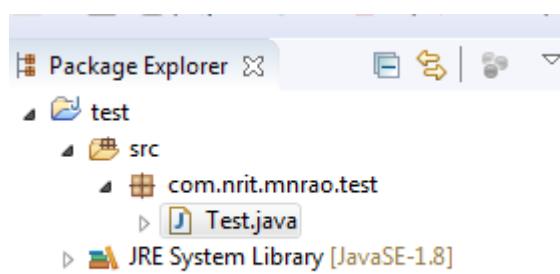
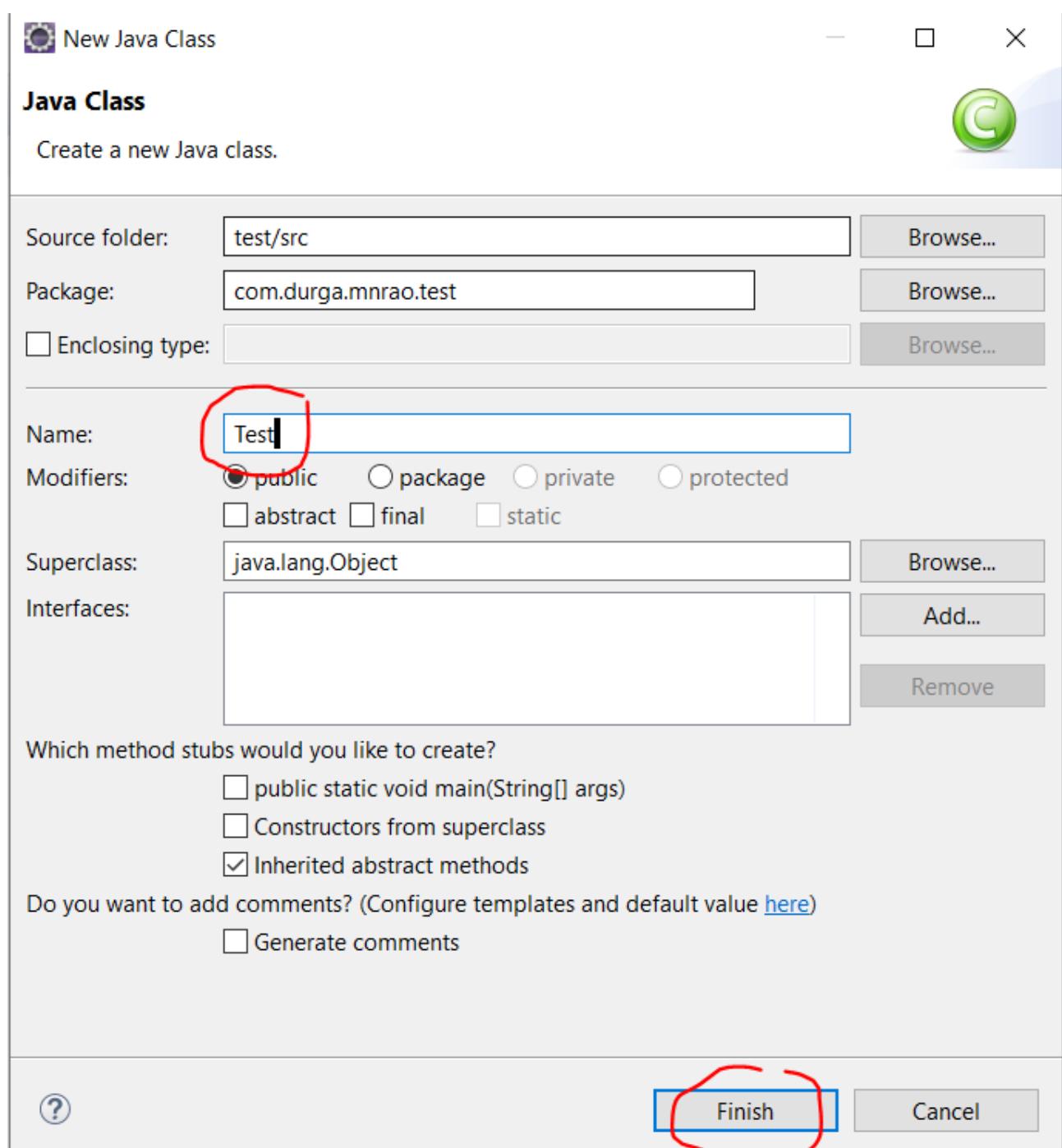
Step4:

Create a class

Rt.click on package → new → class



Provide the class name : Test (class name should start with upper case alphabet.



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

public class Test {

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

How to run java program

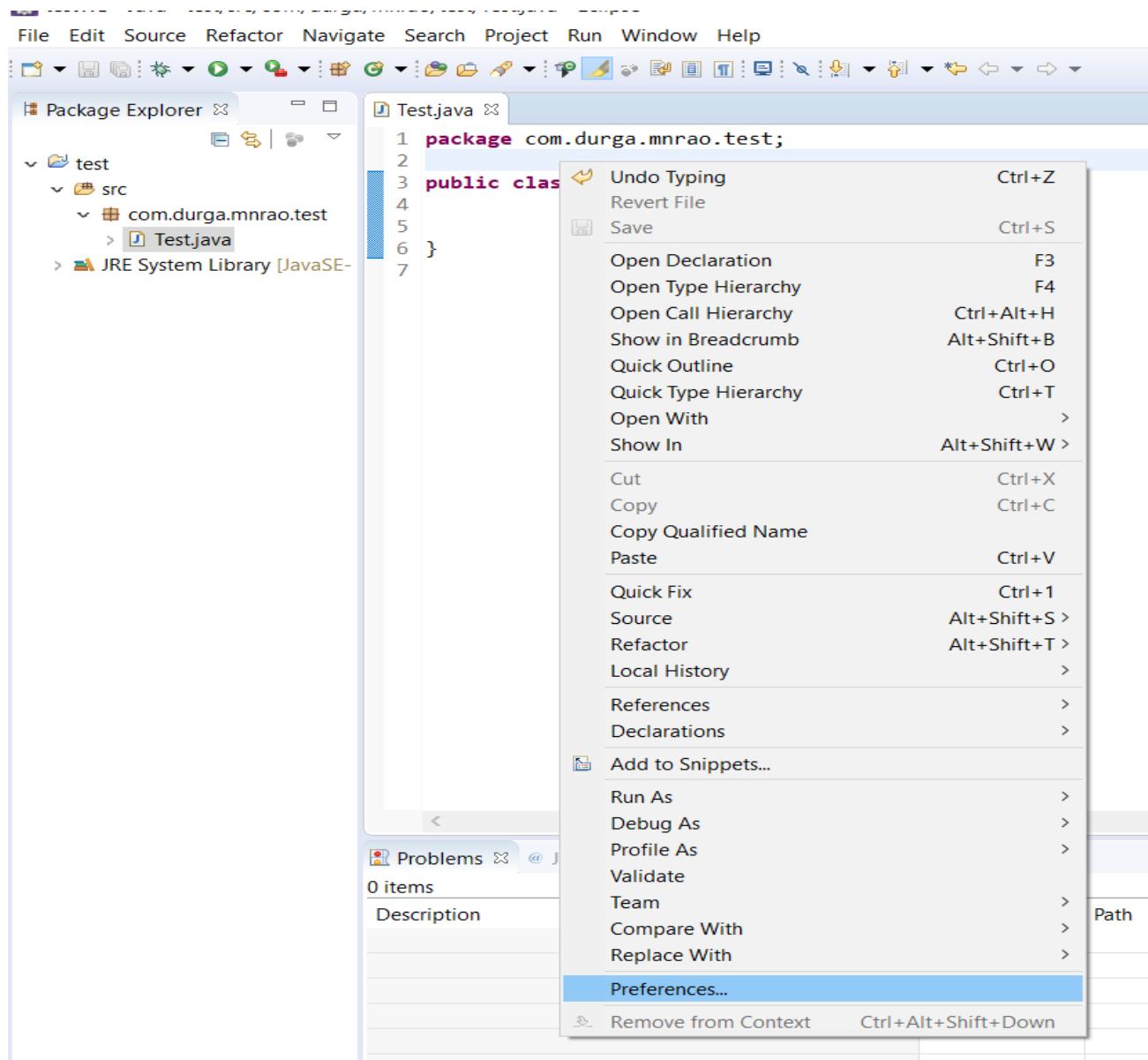
Rt.click on main() program file → runAS → java Applicaton

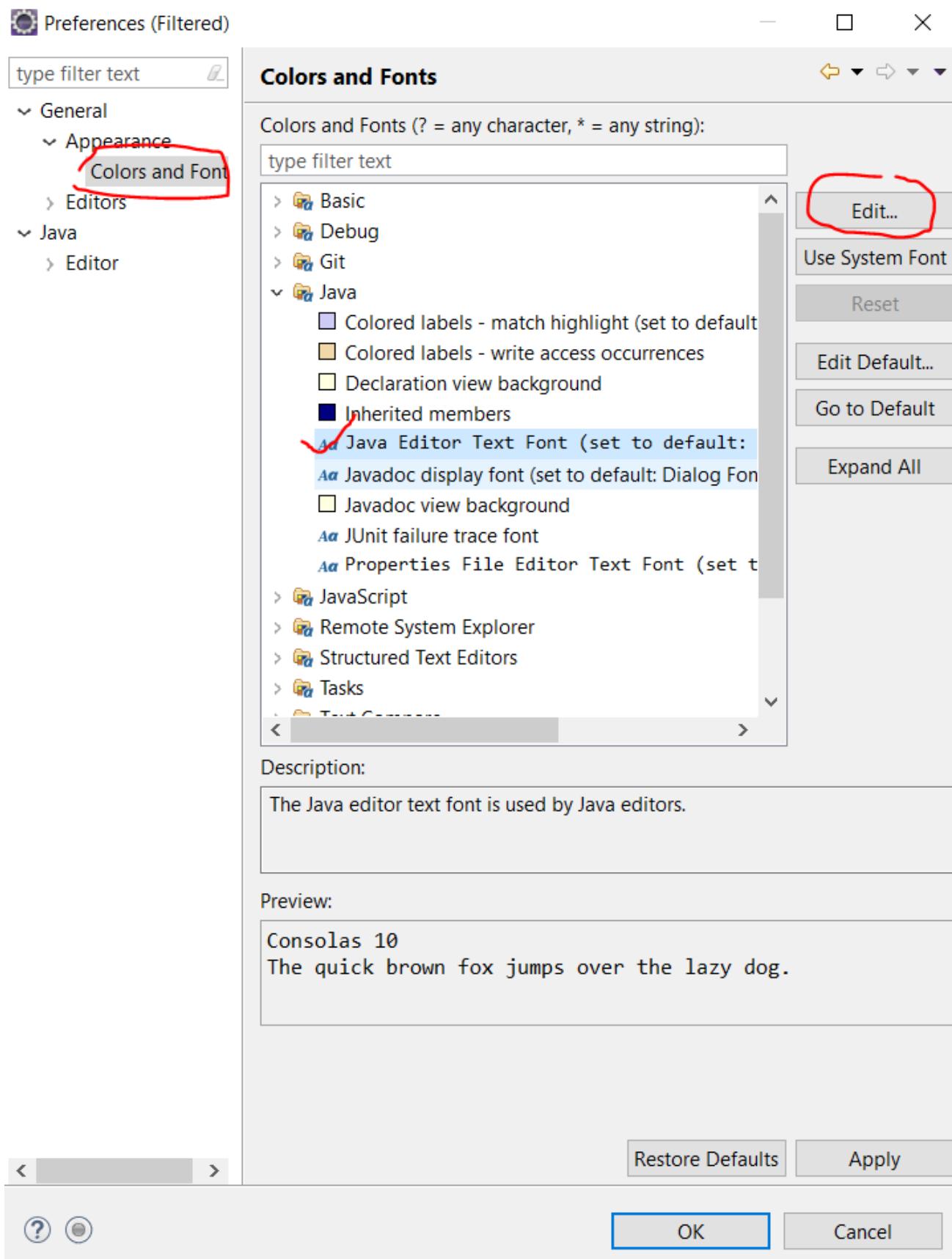
At Console O/p:

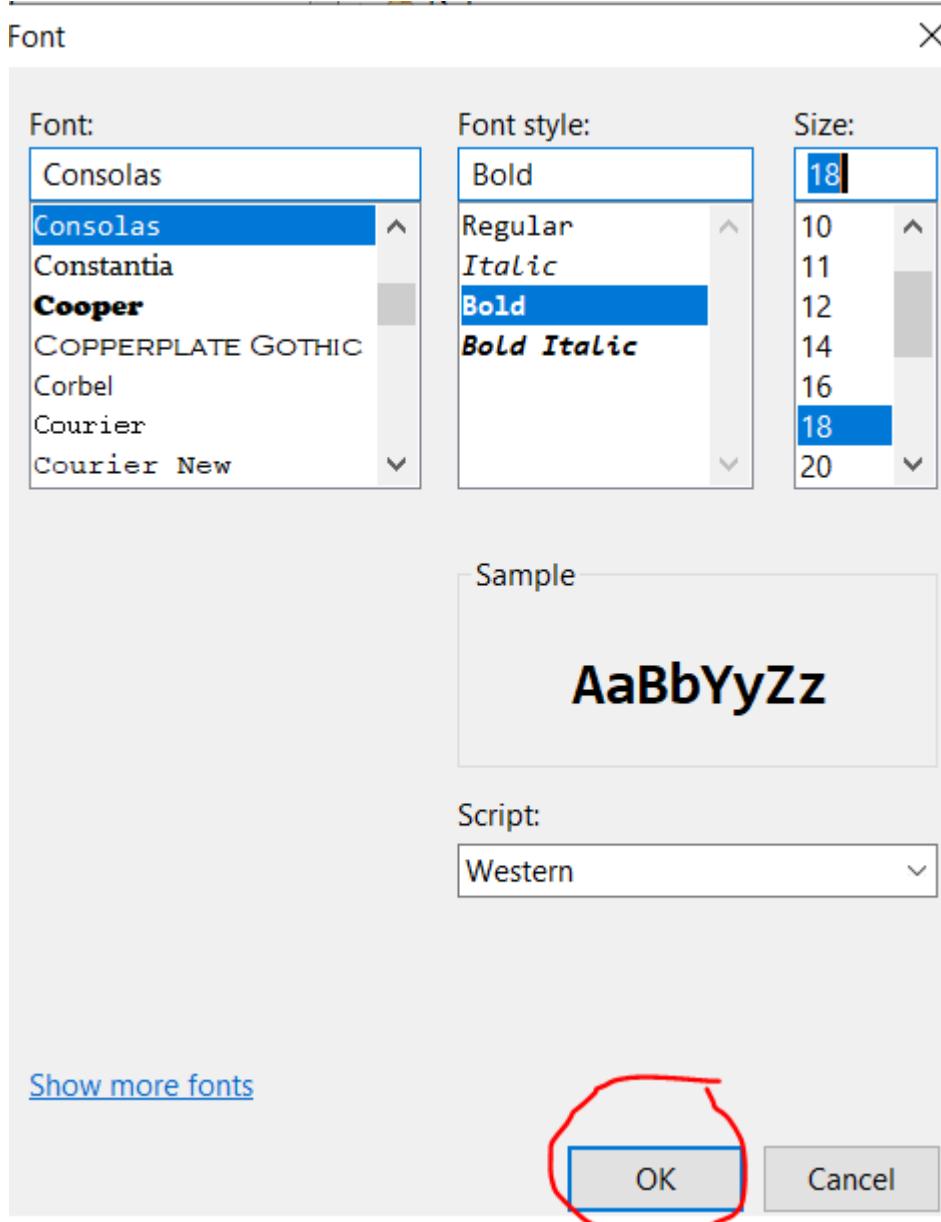
Hello World

Changing font size :

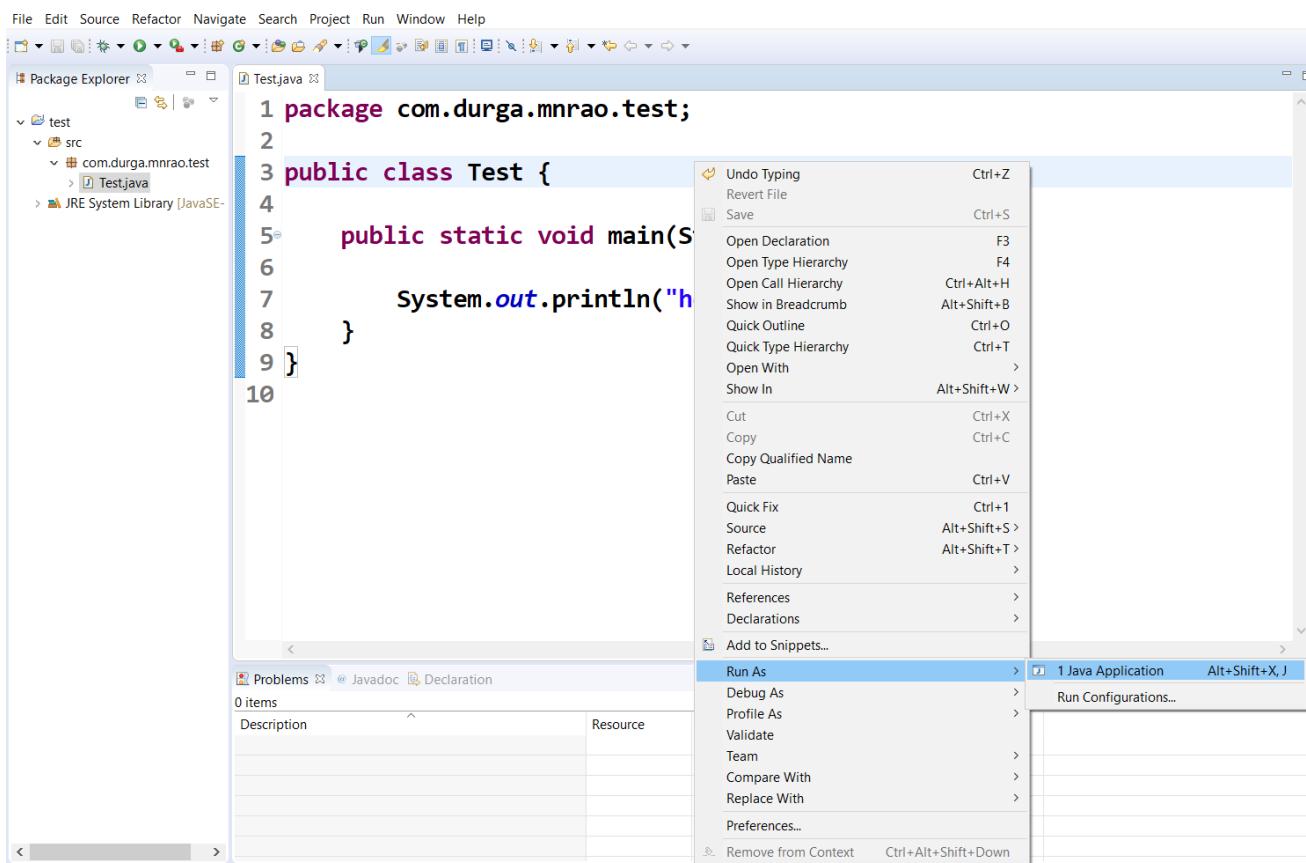
Right click on main program file and select preference as below







Running Project :



How to write sample program in java using Notepad

Hello World program:

D:\>test>notepad Test.java → windows

//program to display Hello World

```
/*
Developed On      :
Developed By      :
Modified On       :
Modified By       :
Reason to Modify :
Description       :
```

Rights reserved @ volkswagen

*/

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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,

eg:

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**.

All Java packages are from java library .

Q.what is java library ?

ans:

it is a collection of packages

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 all the classes from the packages (including 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, which is 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
WeeklySalesReport
MyFirstExample
WordCountExample
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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
    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(), 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?

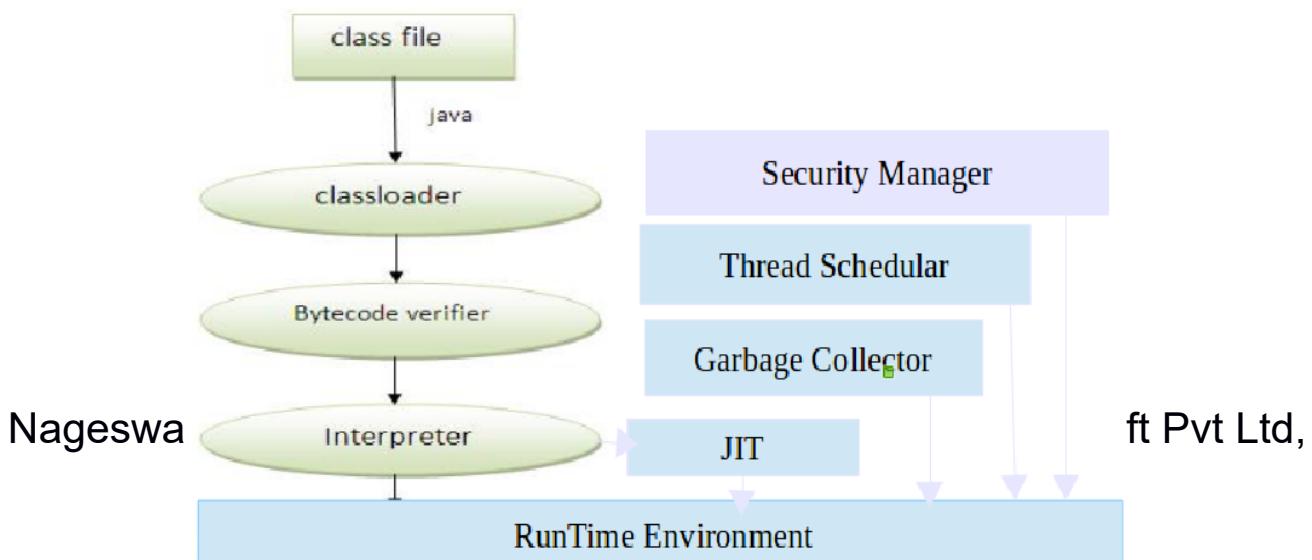
At runtime, following steps are performed:

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 converts 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 :

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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 Scheduler:

Thread scheduler 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.

Interview Questions:

1) What is operating system ?

Ans:

It is a platform, where we can execute application programs and
It is an interface between Hardware and Application software.

2) What is an exe file ?

Ans:

exe file is a prepared file for specific OS. It contains binary code.
exe file contains, program code, application software libraries, OS libraries and hardware
exe file is a OS dependent.

3) can we create exe file in java ?

Ans :

No we can't create exe file. In java every thing is .class file, it is not an exe file

4) What is a .class file ?

Ans;

.class file is byte code file, which contains only java program code and Java libraries.
It don't have any OS library and it is independent of platform

5) how java program file is an independent of platform ?

Ans;

it is independent of Platform since it does not contain any OS libraries and Hardware information.

Since Java .class file dynamically linking OS library , it is an independent

6) What is difference between exe file and .class file ?

Ans:

exe file contains binary code and .class file contains byte code
exe file is prepared file for specific OS and .class is not prepared file.
After moving .class file to OS, then preparation starts as per current OS.
Exe file contains OS library but .class file don't have any OS Library

7) what is the difference between JDK, JRE and JVM

Ans:

JVM (Java Virtual Machine)

It is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed.

JVM, JRE and JDK are platform dependent because configuration of each OS differs.
But, Java is platform independent.

JDK (Java Development Kit):

It contains JRE + Development Tools (javac, java, javap, javadoc, jar, appletviewer, rmic, jps and etc)

JRE (Java Runtime Environment)

It is used to provide runtime environment.

It is the implementation of JVM.

It contains set of libraries (jar files) and other files that JVM uses at runtime.

JRE = JVM + Java Packages Classes (like util, math, lang, awt, swing etc)+runtime libraries (jars).

JDK	
JRE	Development Tools javac, java, javap, javadoc, jar, appletviewer, rmic, jps and etc)

JVM	Java Packages Classes	
-----	--------------------------	--

The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

8) What is the purpose of JIT Compiler ?

Ans:

Just-In-Time (JIT) compiler is a feature of the run-time interpreter.
It prepares the binary code (generated by Interpreter) as per the current OS Specifications.

9) Can we compile java program without main() ?

Ans: yes.

10) Can we execute java program without main() ?

Ans: No

11) what does mean, Java is portable ?

Ans:

java application build can be moved from one OS to another OS with out any changes in the application build .

12) - Can we have multiple classes in same java file?

Ans : Yes

13) Can we have two public classes in one java file?

Ans:

No

A class must be public which name is same as File name.

14) What is correct syntax for main method of a java class?

- A - public static int main(String[] args)
- B - public int main(String[] args)
- C - public static void main(String[] args)
- D - None of the above.

Ans: C

15) Is an empty .java file a valid source file?

Ans:

No, not valid valid file but it can be compiled.

A Java-source file with zero class-declarations will not result in any .class files

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

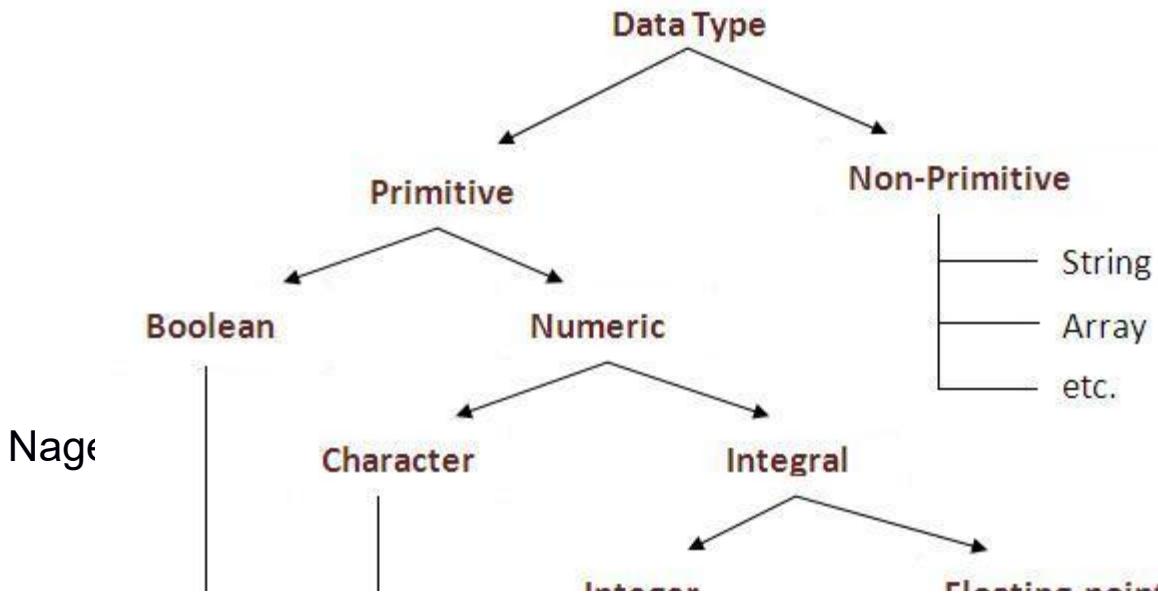
System --> compiler

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	'\u0000'	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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

1 PB → (10^{15}) = 1024 PB

1 EB → (10^{18}) = 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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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

Last bit is for sign

0 → +ve

1 → -ve

$$0 * 2^6 + 0 * 2^5 + 0 + 0 + 1 * 2^3 + 0 * 2^2 + 0 + 1 * 2^1 + 0 * 2^0 = 10$$

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

Last bit is for sign

0 → +ve

1 → -ve

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

1 byte max value = 127

2 bytes of memory

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}

first bit

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

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

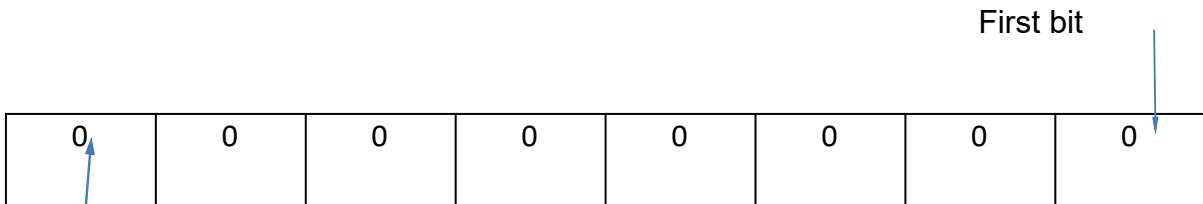
2) Unicode --> Universally Accepted Code

size of Unicode char = 2 bytes

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

ASCII char = 1 byte

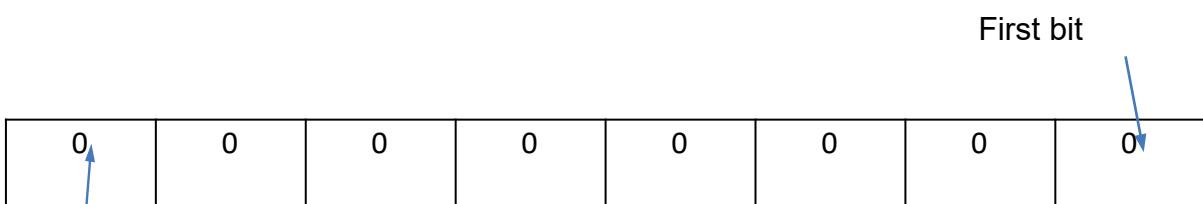
unsigned 1 byte , what is the min value.



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



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			
1	1	1	1	1	1	1	1			

Last bit is also for data

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

$$\begin{aligned}1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 \\128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255\end{aligned}$$

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 Unicode 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 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 languages 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

i18n is char mapping between English and other language.

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);
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 ?

- 1) it is a name given to some location in the program memory
- 2) its value changes at runtime.
- 3) it is a temporary memory , it clears automatically once program stopped
- 4) it is an identifier

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 indentiers

Varaibale 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 overflow 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

Stack Memory space :

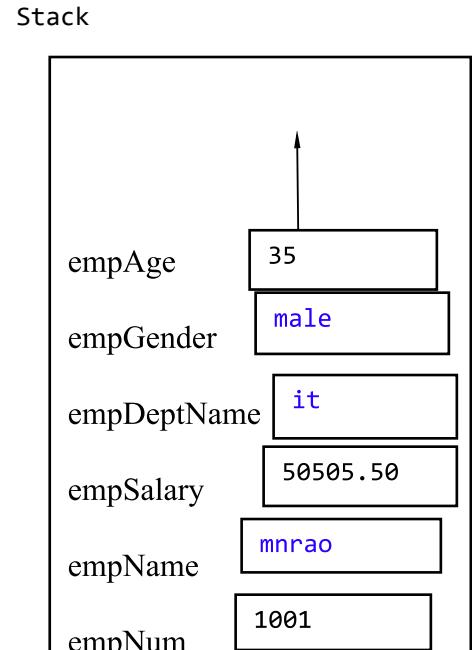
To store all local variables.

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 satck over flow exception.

```
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);  
}  
}
```

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";
```

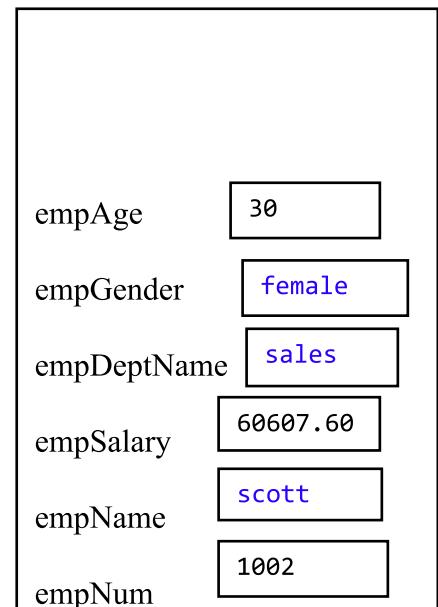
**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 variables :

```
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";  
    }  
}
```

Stack



```
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);  
}  
}
```

Every time of chaning value, 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 .

+ → if both operands are integers , then it is addition

$$a + b$$

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

→ atleast one operand is a string, then it is a concatenation

concatenation --> joining together (placing side by side)

```
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;
```

a

b

temp

```
        int b = 20;
```

20

20

10

```
        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:

InputStreamReader

Console

Scanner

DataInputStream etc.
BufferedReader

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 enter 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();
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 variable 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

Eg:

```
package com.durga.mnrao.xyz;

import java.util.Scanner;

public class Test {

    public static void main(String[] args)  {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter emp details ");

        System.out.println("Enter emp number ");
        int empNum = sc.nextInt();

        System.out.println("enter emp name ");
        String empName = sc.next();

        System.out.println("Enter emp salary ");
        double empSalary = sc.nextDouble();

        System.out.println("enter emp dept name " );
        String empDeptName = sc.next();

        System.out.println("enter emp Gender ");
        String empGender = sc.next();

        System.out.println("Enter emp age : ");
        int empAge = sc.nextInt();

        System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);

    }
}
```

Enter emp details

Enter emp number

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

101

```
enter emp name  
vijaya  
Enter emp salary  
54376.40  
enter emp dept name  
IT  
enter emp Gender  
female  
Enter emp age :  
22  
101 vijaya 54376.4 IT female 22
```

Type Casting:

It is a process of converting one data type to another data type

These are of two types

- 1) primitive data types casting (basic to basic)
- 2) Non-Primitive casting (user defined classes)

primitive data types casting

these are of two types

- 1) Implicit Casting and 2) Explicit casting

1) Implicit Casting

Implicit means internally (automatically)

Implicit casting is available for the below

- 1) Small size to bigsize
- 2) Integer to real number

Small size to bigsize :

eg1:

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

2) Explicit casting:

It is reverse of implicit 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

eg1:

```
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; // valid
```

(or)

float f = 10.5f; // valid

default data type.

Real numbers are by default treated as **double** type.

Intger numbers are by default treated as **int** 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 least 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 categoires

1) Unary Operators → Single Operand

2) Binary Operators → Two Operands

3) Ternary Operators → Three Operands

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Binary Operators:

1) Arithmetic / Mathematical Operators

2) Relational Operators

3) Logical Operators

4) Assignment Operator

5) short hand assignment Operator

6) Bitwise operators

Arithmetic / Mathematical Operators

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

Eg:

$10 \% 3 = 1$

$3 \% 10 = 3$

$9 \% 10 = 9$

$50 \% 51 = 50$

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.

These operators are to compare the values and returns either true or false (Boolean values)

Eg:

```
int a = 10;
```

```
int b = 5;
```

```
boolean res = a>b;
```

here → res value is true

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);
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 (++) :

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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

Control Statements

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

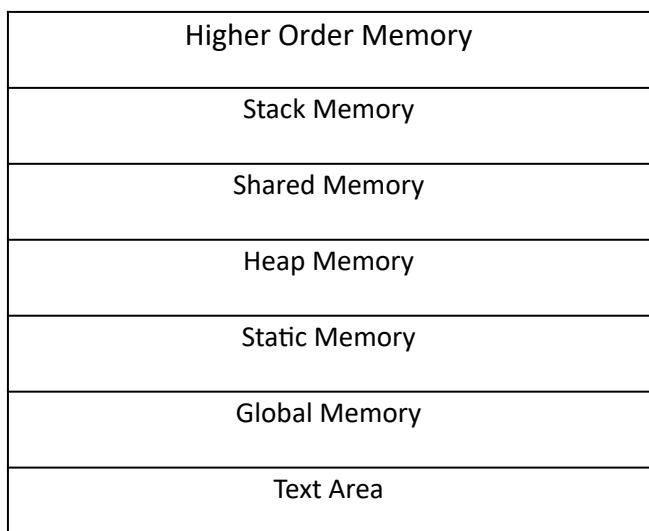
if – statement:

these are of four types

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

1) simple-if:

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



Method Area

2) if-else

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

}
else
{
    =====
    =====
    false part;
    =====
    =====

}
```

b) Nested – if :

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

4) 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");
        }
    }
}
```

Another logic for the above program :

```
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();

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

    }
}
```

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();
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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	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 :

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

Cond1	Cond2	Cond3	Cond4	Cond5	Result
F	F	F	F	F	F -->F
F	F	F	F	T	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 ");

```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 , atleast 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 {
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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");
        }

    }

}
```

Another way of solving (but not recommended)

```
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 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();

        if( m1>=40 && m2>=40 && m3>=40 )
        {
            System.out.println("student passed ");
        }
        else if( m2>=40 && m3>=40 && m4>=40 )
        {
            System.out.println("student passed ");
        }
        else if( m3>=40 && m4>=40 && m1>=40 )
        {
            System.out.println("student passed ");
        }
    }
}
```

```
        else if( m4>=40 && m1>=40 && m2>=40 )
        {
            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
```

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 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 presence 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.**

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

Case Constant can be a **integer** or **char** or **String** but should not be a **real number** or **Boolean type**

Eg:

10, 20 , 30 , 40 , 50 → valid

‘a’ , ‘b’ , ‘c’ , ‘d’ → valid

“ONE” , “TWO”, “THREE” → valid

10.5, 20.1, 1.1 → invalid

true / false → 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.nrity.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.nrity.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)
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
    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)  
    {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 n.

```
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 = 1;

        int evenSum = 0;

        int oddSum = 0;

        while ( i <= n )
        {

            if( i%2 == 0)
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
{  
    evenSum = evenSum + i;  
}  
else  
{  
    oddSum = oddSum + i;  
}  
  
i++;  
}  
  
System.out.println("even sum = "+evenSum);  
  
System.out.println("odd sum = "+oddSum);  
  
}  
}
```

using for loop:

```
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 evenSum = 0;  
  
        int oddSum = 0;  
  
        for( int i = 1; i <= n; i++ )  
        {  
  
            if( i%2 == 0)  
            {  
                evenSum = evenSum + i;  
            }  
else  
            {  
                oddSum = oddSum + i;  
            }  
        }  
    }  
}
```

```
System.out.println("even sum = "+evenSum);

System.out.println("odd sum = "+oddSum);

}

}
```

Program to print following series

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

```
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();

        for(int i = 1, j = 2; i<=n && j<=n ; i = i + 2, j = j + 3)
        {
            System.out.print(i+" "+j+" ");
        }

    }
}
```

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;
}
}
```

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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

```
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

```
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 is a keyword , to terminate the loop.

```
while<cond>
{
=====
=====
    if<cond>
    {
        break;
    }
=====
=====
}
=====
=====;
```

System.exit(0) → it is a method from System class, to terminate complete program.

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");
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

}

}

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");
        }
    }
}
```

Checking for Prime number :

```
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(); // 7

        boolean flag = true;

        for(int i = 2 ; i<n ; i++)
        {

            if( n % i == 0 )
            {
                flag = false;

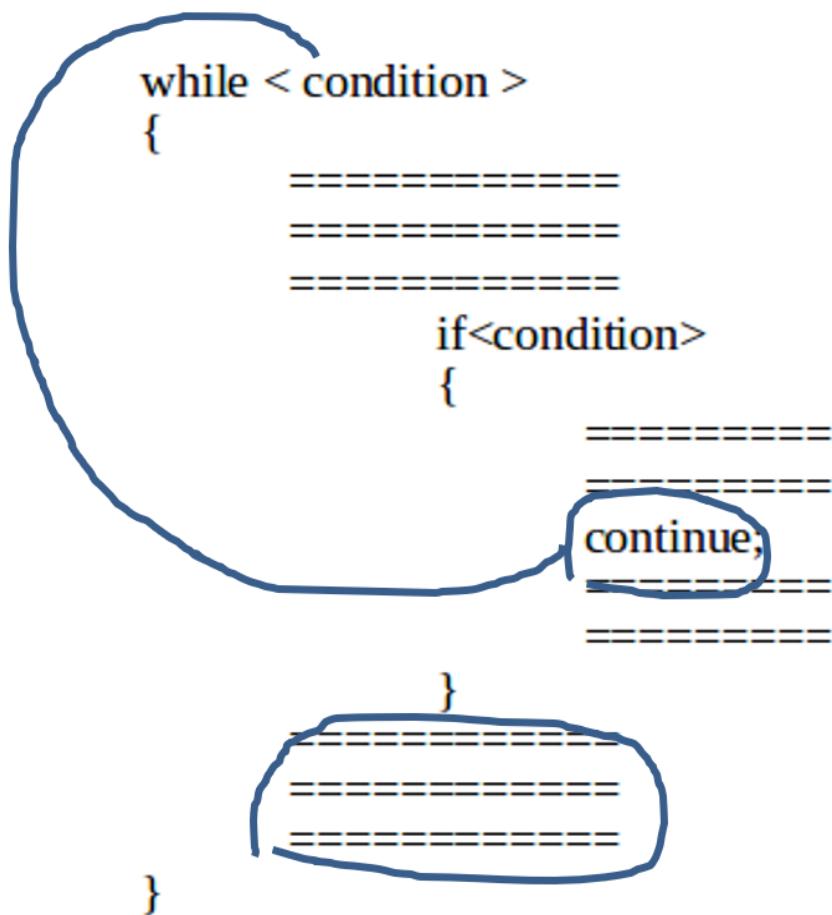
                System.out.println("Not a prime number");

                break;
            }
        }

        if( flag )
        {
            System.out.println("it is a prime 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.

```
for(initialization ; <cond> ; incr / decr )  
{  
=====;  
=====;  
if<cond>  
{  
    continue;  
}  
=====;  
=====;  
}
```

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

Nested loops :

```
while<cond>
{
=====
=====
    while<cond>
    {
=====
=====
=====
=====
    }
=====
=====
}
```

```
for( initialization ; condition ; incr/decr )  
{  
=====;  
=====;  
    for( initialization ; condition ; incr /decr )  
    {  
=====;  
=====;  
=====;  
=====;  
    }  
=====;  
=====;  
}
```

program to check given number is strong number or not

$$145 \rightarrow 1! + 4! + 5!$$

$$1 + 24 + 120 = 145$$

```
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 sum = 0;  
  
        while(n>0)  
        {  
            int r = n%10;  
  
            int fact ;  
  
            for(fact = 1; r>=1; r--)  
            {  
                fact = fact * r;  
            }  
        }
```

```
        sum = sum + fact;

        n = n / 10;

    }

    System.out.println("sum = "+sum);

    if( sum == temp)
    {
        System.out.println("Strong number");
    }
    else
    {
        System.out.println("Not a strong number");
    }

}
```

program to display strong numbers from 1 to n

```
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();

        for(int i = 1; i<=n; i++)
        {

            int temp = i;

            int sum = 0;

            while(temp>0)
            {
                int r = temp%10;

                int fact ;

                for(fact = 1; r>=1; r--)
                {
```

```
        fact = fact * r;
    }

    sum = sum + fact;

    temp = temp / 10;

}

if( sum == i)
{
    System.out.println(i+" ");
}

}
}
```

program to check given number is armstrong number or not

153 --> 1^3 + 5^3 + 3^3

1 + 125 + 27 = 153

```
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 temp = n;

        int sum = 0;

        while(n>0)
        {
            int r = n%10;

            sum = sum + r * r * r;

            n = n / 10;
        }
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
    }

    System.out.println("sum = "+sum);

    if( sum == temp)
    {
        System.out.println("Arm Strong number");
    }
    else
    {
        System.out.println("Not an Arm strong number");
    }

}

}
```

program to display armstrong number from 1 to n

```
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();

        for(int i =1; i<=n ; i++)
        {

            int temp = i;

            int sum = 0;

            while(temp>0)
            {
                int r = temp%10;

                sum = sum + r * r * r;

                temp = temp / 10;
            }

            if( sum == i)
```

```
{  
    System.out.println(i+ " ");  
}  
  
}  
}  
}
```

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 .

```
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();

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 display twin primes from 1 to n

difference between two consecutive prime numbers should be 2.

1 2 3 5 7 11 13 17 19 23 29 31....

```
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 p1=0;

        int p2=0;

        for(int i=1; i<=n; i++)
        {
            boolean flag = true;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        for(int j = 2; j<i ; j++)
        {
            if( i%j == 0)
            {
                flag = false;

                break;
            }
        }

        if(flag)
        {
            p2 = i;
        }

        if( (p2-p1) == 2 )
        {
            System.out.println(p1+"\t"+p2);
        }

        p1 = p2;

    }

}
```

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++)
    }}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 triangle

```
*
```

```
* *
```

```
* * *
```

```
* * * *
```

```
* * * * *
```

```
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+" ");  
            }  
        }  
    }  
}
```

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

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

```
}
```

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 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+" ");  
            }  
        }  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
    }

}

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	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
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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  
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)
                {

```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

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.

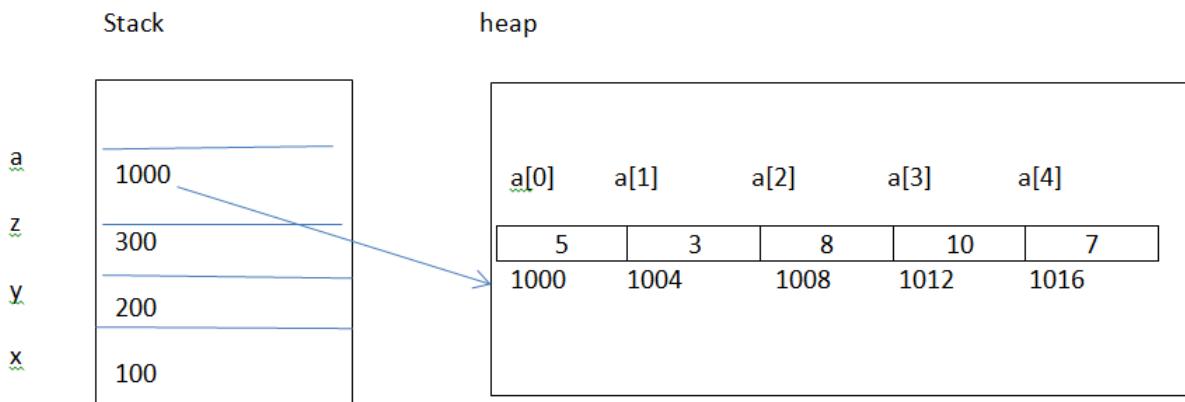
Accessing array elements:

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 ?

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

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 and initialization

`int []a=new int[6]{1,2,3,4,5,6}; // invalid, size should not be specified`

`int a[10]={1,2,3,4,5.....};// is not valid in java, size should be specified`

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

`a.length → gives no of elements in array`

length :

it is a pre-defined variable provided by JVM . it is **not** from any pre-defined package (such as `java.lang`)

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Memory Management:

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 overflow 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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

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; // declaration
a=new int[5]; // memory allocation
a[0]=10; // assigning values
a[1]=15;
a[2]=21;
a[3]=25;
a[4]=30;**

2) int a[]={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]; // valid  
b=new int[9]; // Invalid, it is not an array  
c=new int[8]; // Invalid, it is not an array
```

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 better 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);

}
}
```

Foreach loop:

Syntax:

```
for( data_type variable : array)
{
    =====;
    =====;
    =====;
}
```

Accessing array elements using for-each loop:

```
public class Test
{
    public static void main(String[] args)
    {
        int a[]={10,8,13,18,15,6};
```

```
for (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.

Since String class is from java.lang, it is not necessary to import

```
public class Test {

    public static void main(String [] args)
    {
        String [] names={"hadoop","java","html","oracle","hive","sqoop","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","sqoop","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.nrity.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 occurrences "+counter);

    }
}
```

Binary Search:

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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 };

        Scanner sc = new Scanner(System.in);

        System.out.println("enter the number you want to search ");

        int num = sc.nextInt();

        int l = 0;

        int h = a.length-1;

        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");
        }
    }
}
```

1) Linear Search:

--> searching from beg to end

--> numbers are duplicate and randomly

eg:

name , age, dob, gender, dept

except ID field

2) Binary Search

--> Search in the half

--> numbers are unique and Ascending order

eg:

searching based on id field

How to sort an Array in Java:

Arrays is a class from `java.util` package , which provide **sort()** to sort the elements.

ArraySort.java

```
package com.nrity.mnrao.test;

import java.util.Arrays;

public class ArraySort {

    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]);
        }
    }
}
```

```
}
```

```
}
```

Strings sorting :

```
package com.nrity.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];
```

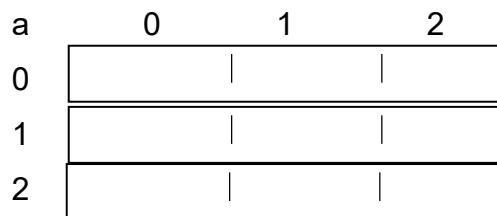
eg:

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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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 initialization :

```
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

program to check no of rows and cols in two-Dimensional array

```
public class Test {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public static void main(String[] args) {  
  
    int [][] a = { {1,2,3}, {4,5,6,7,8}, {9,10},{11},{12,13,14,15} };  
  
    System.out.println("no of rows = "+a.length);  
    System.out.println("no of cols in 1st row = "+a[0].length);  
    System.out.println("no of cols in 2nd row = "+a[1].length);  
    System.out.println("no of cols in 3rd row = "+a[2].length);  
    System.out.println("no of cols in 4th row = "+a[3].length);  
    System.out.println("no of cols in 5th row = "+a[4].length);  
  
}  
}  
}
```

eg:

```
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] + " ");  
            }  
        }  
    }  
}
```

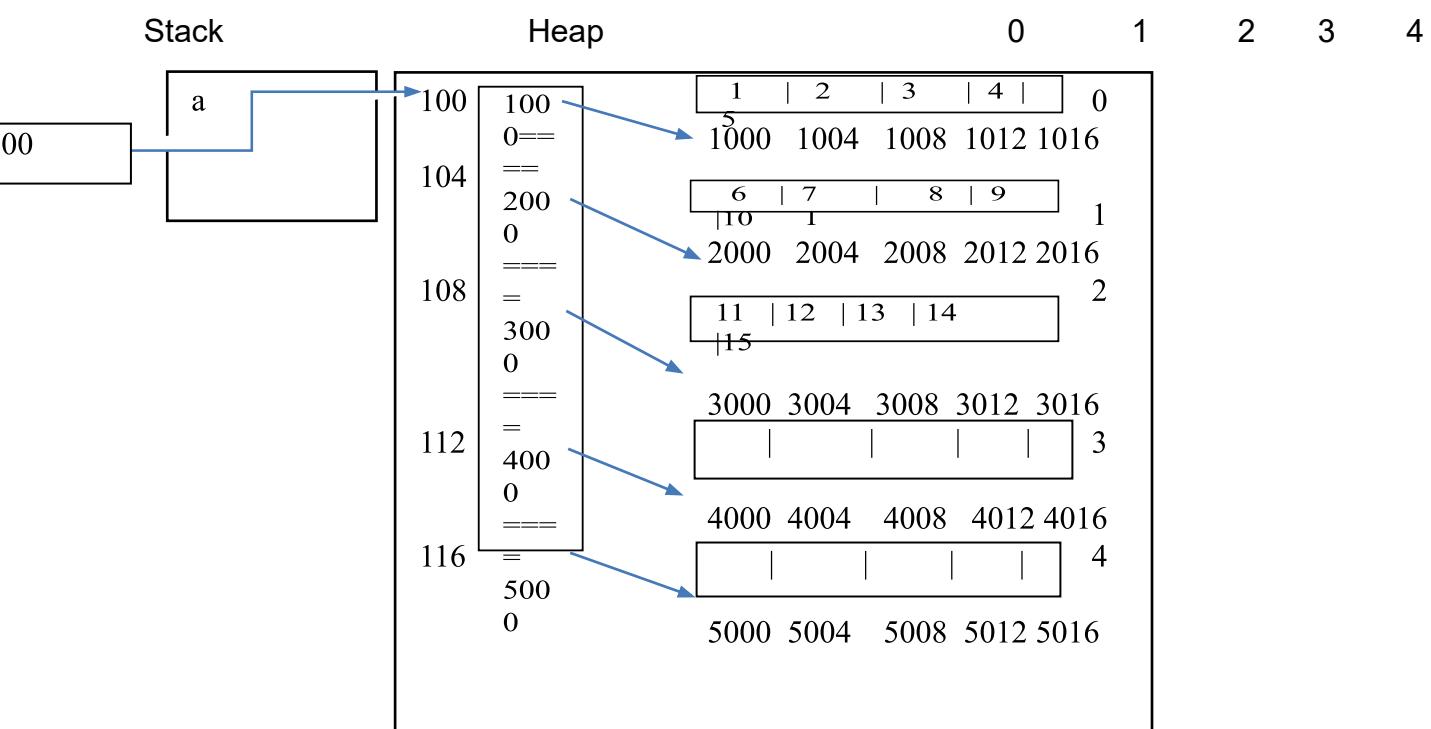
Memory Map of Two – Dimensional Arrays

Dynamic Memory Management for Two Dimensional Arrays :

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

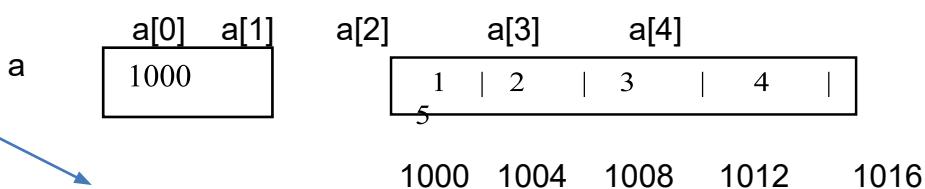
Eg:

```
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 \rightarrow 1000.length \rightarrow (base address). length

Two Dimensional arrays

```
a[0]  $\rightarrow$  1000[0]  $\rightarrow$  *( 1000 + 0 * 4 )  $\rightarrow$  *(1000)  $\rightarrow$  1
a[1]  $\rightarrow$  1000[1]  $\rightarrow$  *( 1000 + 1 * 4 )  $\rightarrow$  *(1004)  $\rightarrow$  2
a[2]  $\rightarrow$  1000[2]  $\rightarrow$  *( 1000 + 2 * 4 )  $\rightarrow$  *(1008)  $\rightarrow$  3
a[3]  $\rightarrow$  1000[3]  $\rightarrow$  *( 1000 + 3 * 4 )  $\rightarrow$  *(1012)  $\rightarrow$  4
a[4]  $\rightarrow$  1000[4]  $\rightarrow$  *( 1000 + 4 * 4 )  $\rightarrow$  *(1016)  $\rightarrow$  5
```

base address [0] \rightarrow 1st element

base address [1] \rightarrow 2nd element

base address [2] \rightarrow 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, different 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 wastage 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]+ " ");
}

}

}

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

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

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] = i+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] + " ");
    }
}

}
```

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**

eg:

defining a method :

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

}

Calling a method:

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

```
public void display()// called method.
{
    =====;
    =====;
    =====;
    =====;
    =====;
}
```

How to choose name of the methods ?

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

same for all identifiers

2) Naming Conventions :

=====

these are coding standards

- 1) method name should start with lower case alphabet

2) method name contains multiple words , then

1st word start with lower case and next words start with upper case

eg:

```
setEmpDeptName();  
getMaxRateOfInterest();  
getMinRateOfInterest();  
actionPerformed()  
getDatabaseConnection();
```

There are four types of methods

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

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

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 ("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
display start
I am in show
display end
after display
before show
I am in show
after show
main end
```

eg:

example to explain about method stack

```
package com.nr.it.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");
    }
}
```

Memory Management:

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 overflow 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

Method Area / Method Stack :

It contains copy of the methods. For every time of calling a method, JVM creates copy of method and loads into method stack .

Method stack works based on LIFO (Last In First Out)

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");

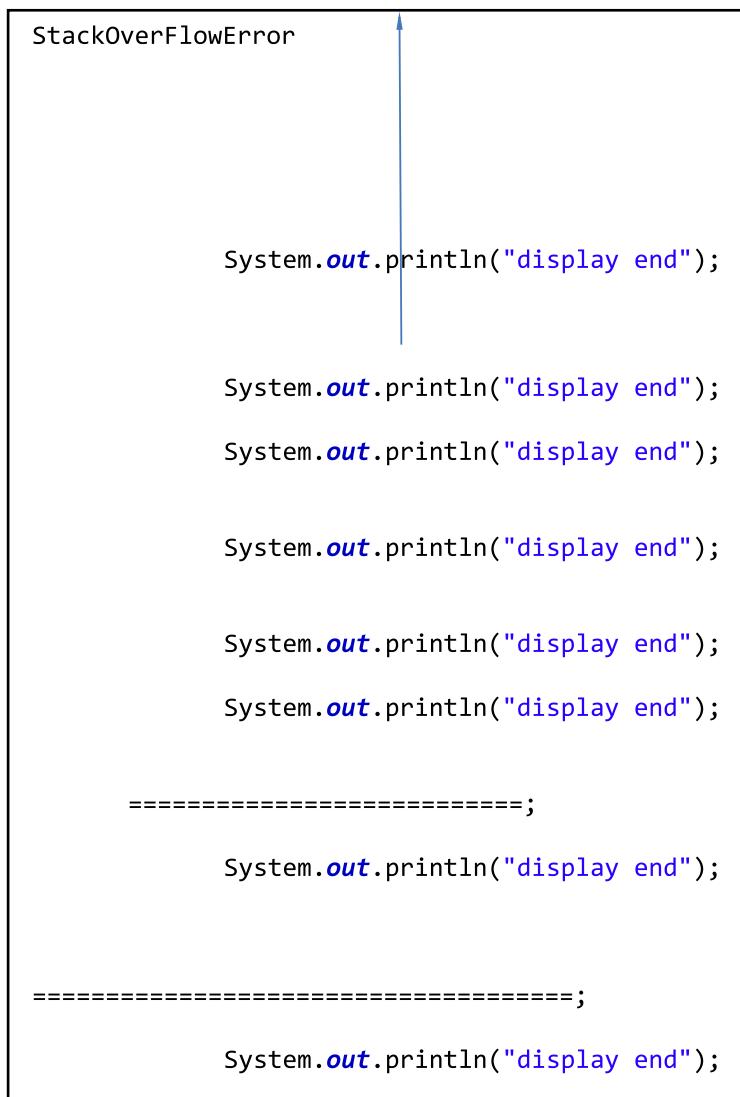
    display();

    System.out.println ("display end");
}
```

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

Once method stack filled with all pending statements, it raises stack StackOverflowError

Method Area (stack):



Method with parameters /arguments and no return value

public class Test

{

public static void main(String[] args)

{

int a=10; a
int b=20;

System.out.println ("Main start");

display(a , b); // here a and b are parameters

System.out.println ("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 10 y 20

System.out.println ("display");

System.out.println ("x "+x+" y "+y);

}//display close

public static void show(int x, int y)

{

x 50 y 60

System.out.println ("I am in show");

System.out.println ("args are "+x+" "+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 :

method arguments are local to method and local copies creates for arguments.

Once control come out of method, loss of all local copies (arguments)

Scope of arguments within the method only.

To call a method, no of parameters and type of parameters should match with method definition

Method with different type of arguments :

```
public class Test {  
  
    public static void main(String[] args)  
    {  
  
        System.out.println("main start");  
  
        display(10, 20.5f, 'A');  
  
        System.out.println("main end");  
    }  
    // different type of arguments  
    public static void display(int x, float y, char ch)  
    {  
        10  20.5  A  
        System.out.println("x = "+x +"\t y = "+ y +"\t ch = "+ch);  
    }  
}
```

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 ("subtraction "+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
subtraction -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 condition 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
```

Method with no args and return value:

```
package com.nrity.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 ;
}
=====;
=====;
}
```

break --> is a keyword to terminate loop

exit --> is a method from System class to terminate complete program

```
System.exit(0);
```

return --> is a key word to return value or to terminate method

continue --> is a keyword to skip rest of the loop

Interview Questions

1.Why Java has most popularity in software development:

Ans: Independent of Plat form and portability

2.can we execute exe file on any plat form ?

Ans : No , exe file is plat form dependent.

3.can we create exe file in Java

Ans : No, can not be created. In java .class is not an exe.

4.How can we achieve independent of plat form and portability in Java

Ans: JVM, is responsible to execute java .class file on any OS.

5.what is the difference between C-Language array and Java array.

Ans: in C-language, array is a variable, whereas in Java it is an Object.

8. What is the difference between continue and break statement?

Ans: break and continue are two important keywords used in Loops. When a break keyword is used in a loop, loop is broken instantly while when continue keyword is used, current iteration is broken and loop continues with next iteration.

9.What are Java Packages? What's the significance of packages?

Ans: In Java, package is a collection of classes and interfaces which are bundled together as they are

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

related to each other. Use of packages helps developers to modularize the code and group the code for proper re-use. Once code has been packaged in Packages, it can be imported in other classes and used.

10. which is default available package in Java application

Ans:

java.lang package is a default available package in the java applications, hence import not required.

11. can we execute java program with out main method.

Ans: No, we can't

12. Can we use String with switch case?

One of the Java 7 feature was improvement of switch case of allow Strings. So if you are using Java 7 or higher version, you can use String in switch-case statements.

13. What is difference between Heap and Stack Memory?

Major difference between Heap and Stack memory are as follows:

Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

Memory management in stack is done in LIFO manner whereas it's more complex in Heap memory because it's used globally.

14. Java Compiler is stored in JDK, JRE or JVM?

The task of java compiler is to convert java program into bytecode, we have javac executable for that. So it must be stored in JDK, we don't need it in JRE and JVM is just the specs.

15) What do you mean by platform independence of Java?

Platform independence means that you can run the same Java Program in any Operating System. For example, you can write java program in Windows and run it in Mac OS.

16) What is JVM and is it platform independent?

Java Virtual Machine (JVM) is the heart of java programming language. JVM is responsible for converting byte code into machine readable code. JVM is not platform independent, that's why you have different JVM for different operating systems. We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It's called virtual because it provides an interface that doesn't depend on the underlying OS.

17) What is the difference between JDK and JVM?

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Java Development Kit (JDK) is for development purpose and JVM is a part of it to execute the java programs.

JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence.

18) What is the difference between JVM and JRE?

Java Runtime Environment (JRE) is the implementation of JVM. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn't contain any development tools like java compiler, debugger etc. If you want to execute any java program, you should have JRE installed.

19) What is difference between path and classpath variables?

PATH is an environment variable used by operating system to locate the executables. That's why when we install Java or want any executable to be found by OS, we need to add the directory location in the PATH variable.

Classpath is specific to java and used by java executables to locate class files. We can provide the classpath location while running java application and it can be a directory, ZIP files, JAR files etc.

20) What is ternary operator in java?

Java ternary operator is the only conditional operator that takes three operands. It's a one liner replacement for if-then-else statement and used a lot in java programming. We can use ternary operator if-else conditions or even switch conditions using nested ternary operators.

that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

21) Is Empty .java file name a valid source file name?

Yes, save your java file by .java only, compile it by **javac .java** and run by **java yourclassname**. Let's take a simple example:

```
//save by .java only
class A{
    public static void main(String [] args){
        System.out.println ("Hello java");
    }
}
```

22) What is JIT compiler?

Just-In-Time(JIT) compiler:It is used to improve the performance. JIT compiles parts of the byte code

23) If I don't provide any arguments on the command line, then the String array of Main method will be empty or null?

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

It is empty. But not null.

24) What if I write static public void instead of public static void?

Program compiles and runs properly.

25) What is the default value of the local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

26) Why Java does not support pointers?

Pointer is a variable that refers to the memory address. They are not used in java because they are unsafe(unsecured) and complex to understand.

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- 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)

C-Language is first developed high level language

C - Language --> 1972 --> by Dennis Ritchie

History of OOP languages:

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

C++ supports all features of OOP's Concept

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Java supports only below features

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

Python Support below features

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

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:

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

It is way of initializing an Object with required values

Inheritance:

It is acquiring properties from parent to child. It is 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 int 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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 :

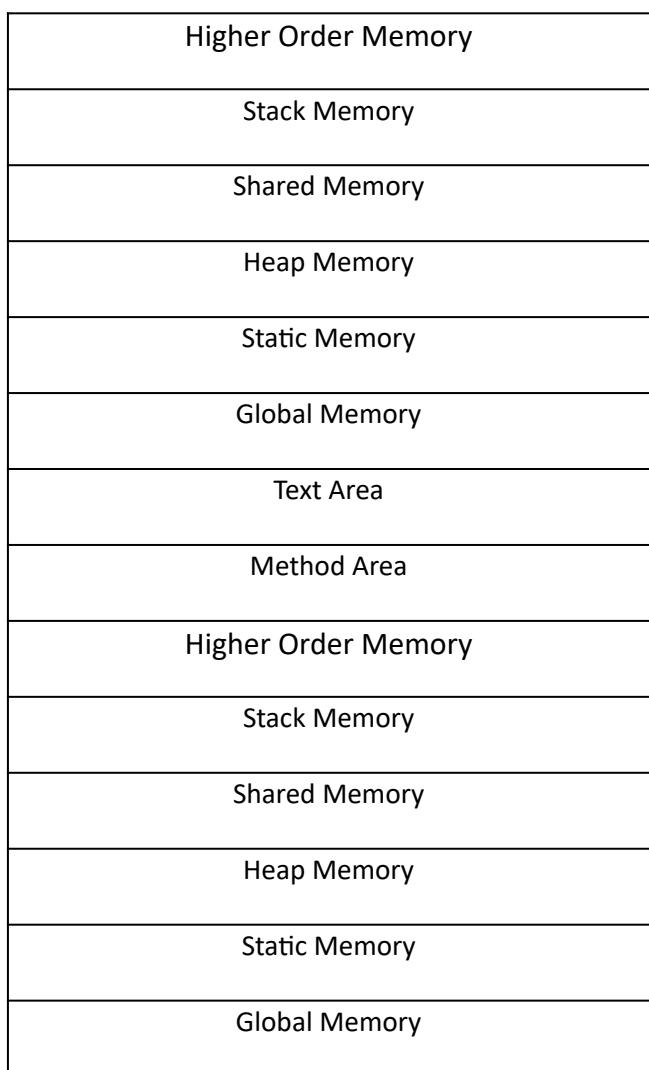
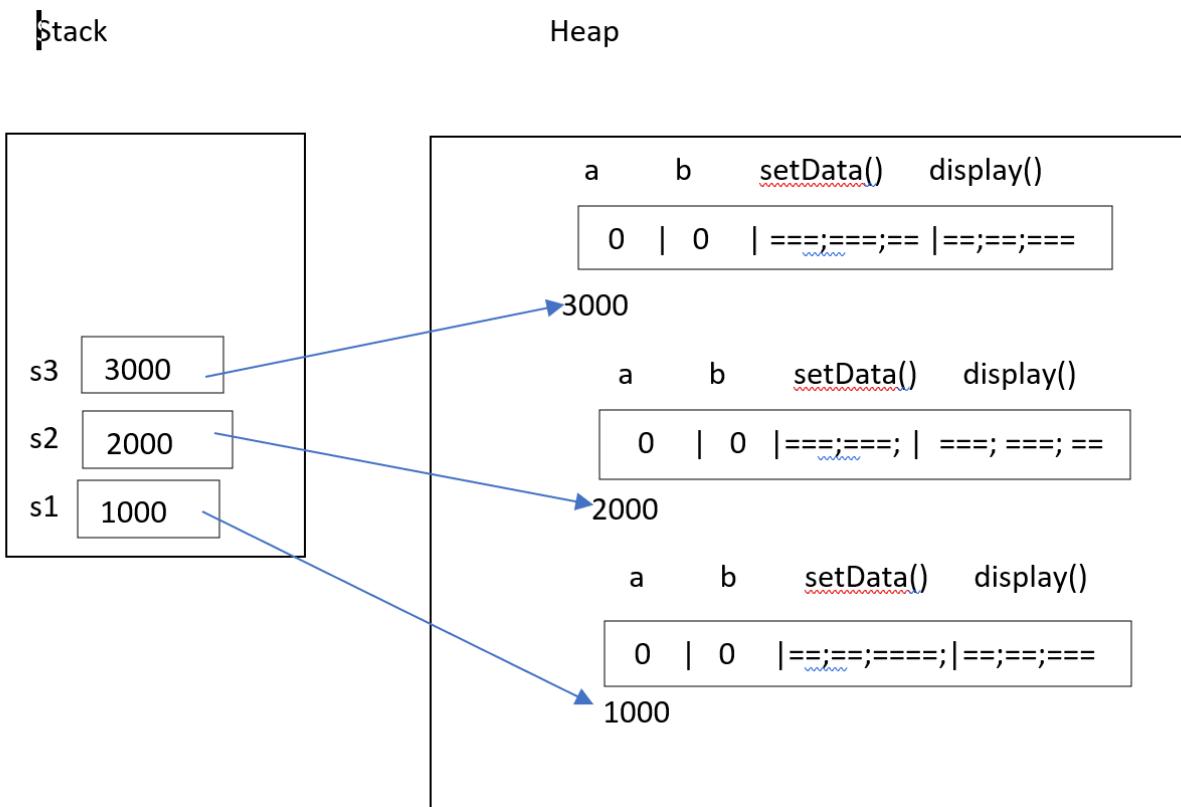
Sample s1 ; // Just it is reference of an Object, not an Object

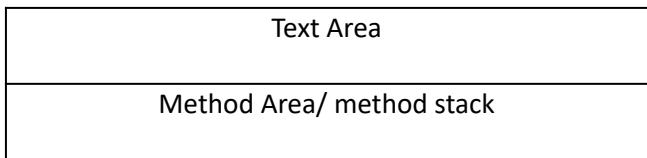
s1 = new Sample(); // it creates an object

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 :

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

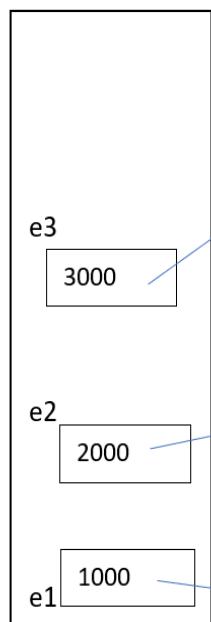
```
Employee e1 = new Employee();
```

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

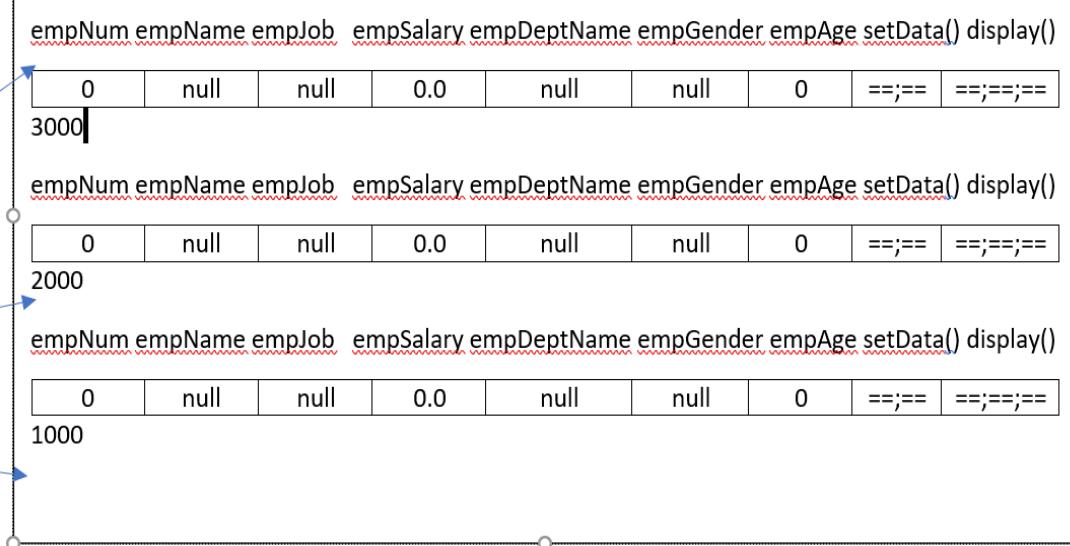
```
Employee e3 = new Employee();
```

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

Stack



Heap



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.nrity.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.nrity.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();  
    }  
}
```

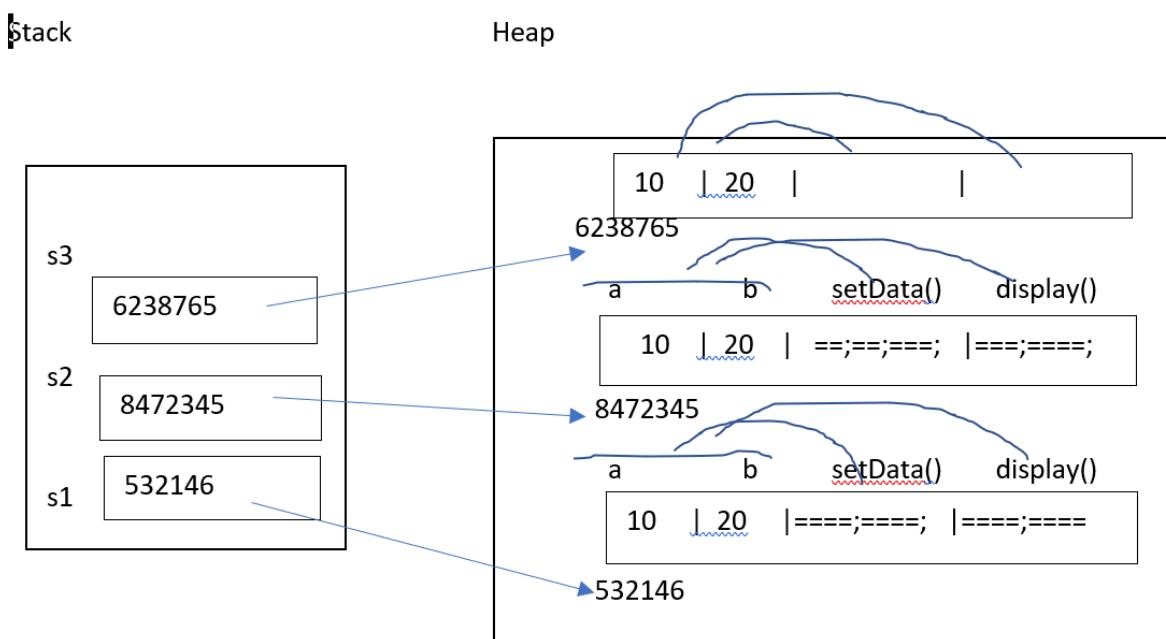
**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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



eg:

```
public class Sample  
{  
    private int a;  
    private float b;  
  
    public void setData()  
    {  
        a=10;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        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();  
    }  
}
```

O/p:

```
a=10  b=30.5  
a=10  b=30.5  
a=10  b=30.5
```

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

```

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+"\t"
b = "+b);
    }
}

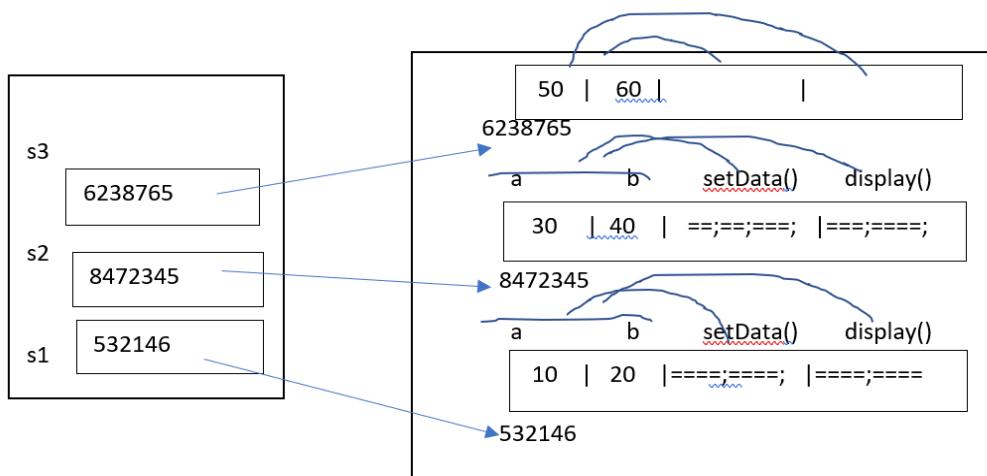
package com.nrit.mnrao.test;

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

```

Stack

Heap



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();  
    }  
}
```

}

Employee 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();

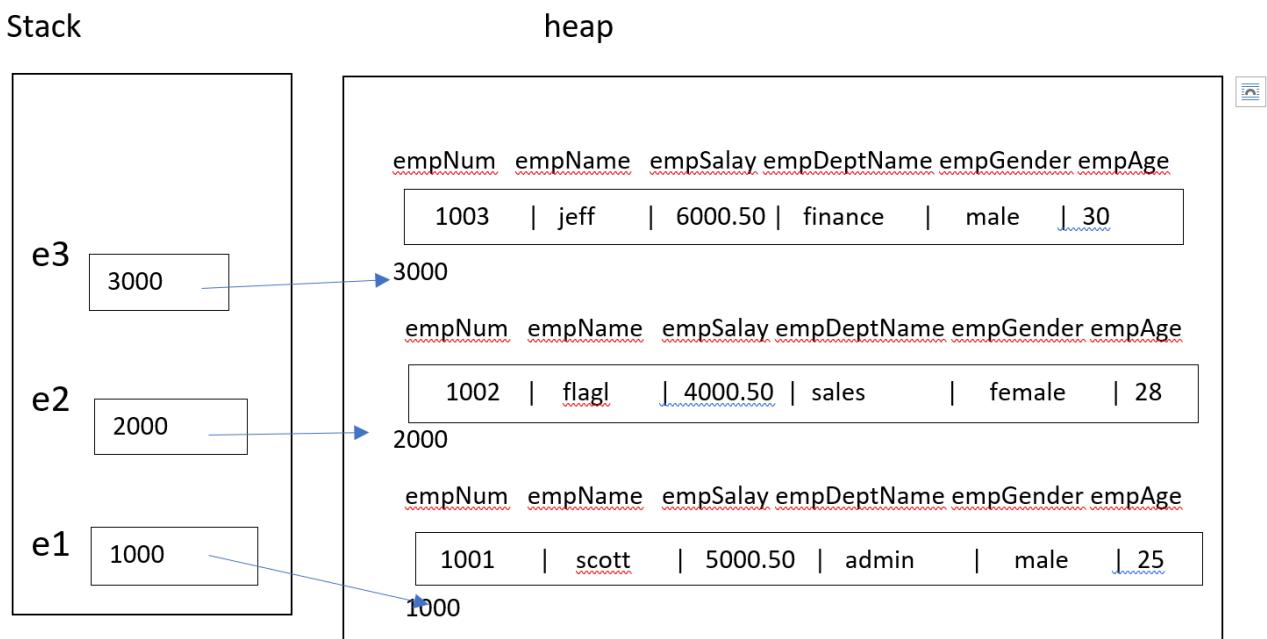
System.out.println ("Emp2 details are");

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();
```

Real 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" + empAge);
}

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"+em
pDeptName+"\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();

    }
}
```

Purpose of setter methods , is to store data into Object and also to change data of an object.

Reading data form key board:

```
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"+em
pDeptName+"\t"+empGender+"\t"+empAge);
}
}

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();

}

}
```

purpose of setter methods, is store data into object
and also to modify the data in the object.

Array of Objects (Multiple Objects) :

Program to read and display five employees records

```
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"+em
pDeptName+"\t"+empGender+"\t"+empAge);
}
}
```

```
package com.durga.mnrao.xyz;

import java.util.Scanner;

public class Test {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("How many employees : ");

        int n = sc.nextInt();

        Employee [] emp = new Employee[n]; // creating array of references

        // creating array of objects

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

            emp[i] = new Employee();
        }

        // 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].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);

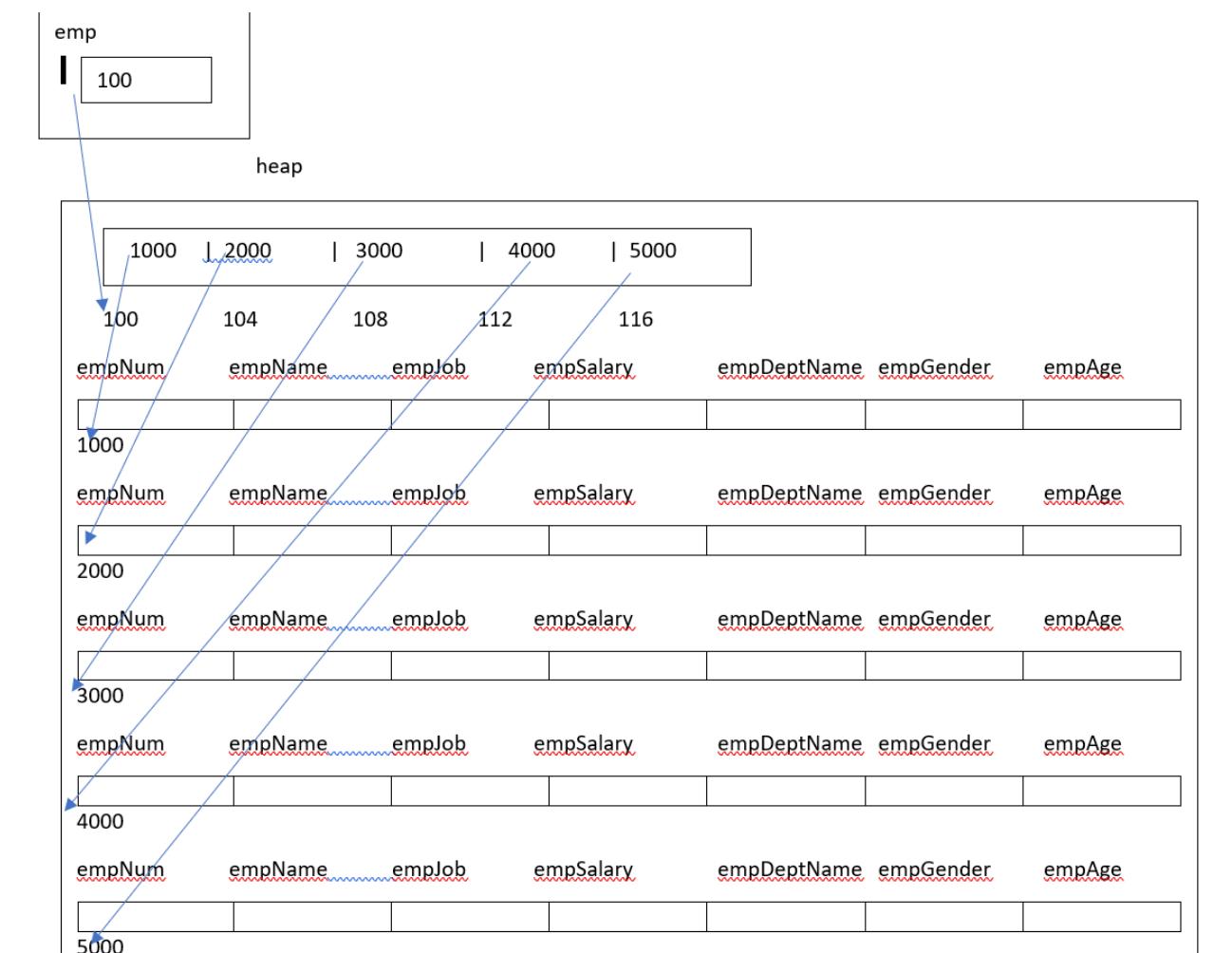
    }

// below loop to display all emp records

for(int i=0; i<emp.length; i++)
{
    emp[i].display();
}

}
```

Memory map:



1. Can we overload main method?

Yes, we can have multiple methods with name “main” in a single class. However if we run the class, java runtime environment will look for main method with syntax as public static void main(String [] args).

2. What is static keyword?

static keyword can be used with class level variables to make it global i.e all the objects will share the same variable.

static keyword can be used with methods also. A static method can access only static variables of class and invoke only static methods of the class.

Read more in detail at java static keyword.

3. Can we declare a class as static?

We can't declare a top-level class as static however an inner class can be declared as static. If inner class is declared as static, it's called static nested class.

Static nested class is same as any other top-level class and is nested for only packaging convenience.

4. What is static block?

Java static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader. It is used to initialize static variables of the class. Mostly it's used to create static resources when class is loaded.

5. What is the importance of main method in Java?

main() method is the entry point of any standalone java application. The syntax of main method is public static void main(String [] args).

main method is public and static so that java can access it without initializing the class. The input parameter is an array of String through which we can pass runtime arguments to the java program.

6) Can we declare a class as static?

We can't declare a top-level class as static however an inner class can be declared as static. If inner class is declared as static, it's called static nested class.

Static nested class is same as any other top-level class and is nested for only packaging convenience.

7) What is static import?

If we have to use any static variable or method from other class, usually we import the class and then use the method/variable with class name.

```
import java.lang.Math;
```

```
//inside class  
double test = Math.PI * 5;
```

We can do the same thing by importing the static method or variable only and then use it in the class as if it belongs to it.

```
import static java.lang.Math.PI;
```

```
//no need to refer class now  
double test = PI * 5;
```

Use of static import can cause confusion, so it's better to avoid it. Overuse of static import can make your program unreadable and unmaintainable.

8) What is difference between object oriented programming language and object based programming language?

Object based programming languages follow all the features of OOPs except Inheritance. Examples

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

of object based programming languages are JavaScript, VBScript etc.

9) What if the static modifier is removed from the signature of the main method?

Program compiles. But at runtime throws an error "NoSuchMethodError".

10) What is difference between static (class) method and instance method?

static or class method

1)A method i.e. declared as static is known as static method.

2)Object is not required to call static method.

3)Non-static (instance) members cannot be accessed in static context (static method, static block and static nested class) static and non-static variables both can be accessed in instance methods.

4)For example: public static int cube(int n){ return n*n*n; } For example: public void msg(){...}.

instance method

A method i.e. not declared as static is known as instance method.

Object is required to call instance methods.

11.What will happen if static modifier is removed from the signature of the main method?

12. How we can execute any code even before main method?

Ans: If we want to execute any statements before even creation of objects at load time of class, we can use a static block of code in the class. Any statements inside this static block of code will get executed once at the time of loading the class even before creation of objects in the main method.

this keyword in java

What is output of Following program, ?

```
public class Sample
```

```
{
```

```
    private int x;
```

```
    public void setData(int x)
    {
        x = x; //overwriting
    }
```

```
    public void display()
    {
        System.out.println (x);
    }
}
```

```
public class Test {
```

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

```
        Sample s1 = new Sample();
```

s1

```
        s1.setData(10);
```

1000

```
        s1.display();
```

```
}
```

```
}
```

O/p:

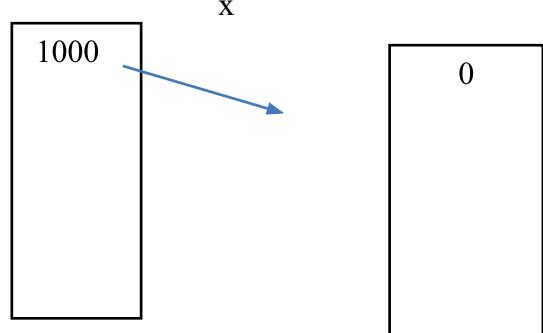
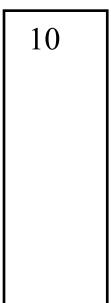
0

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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**



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

Here is given usages 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 varaiable 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.

eg:

below example , instance variable name and method argument name are same

public class Sample {

private int x; // instance variable name
 ^
 # method argument name

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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.

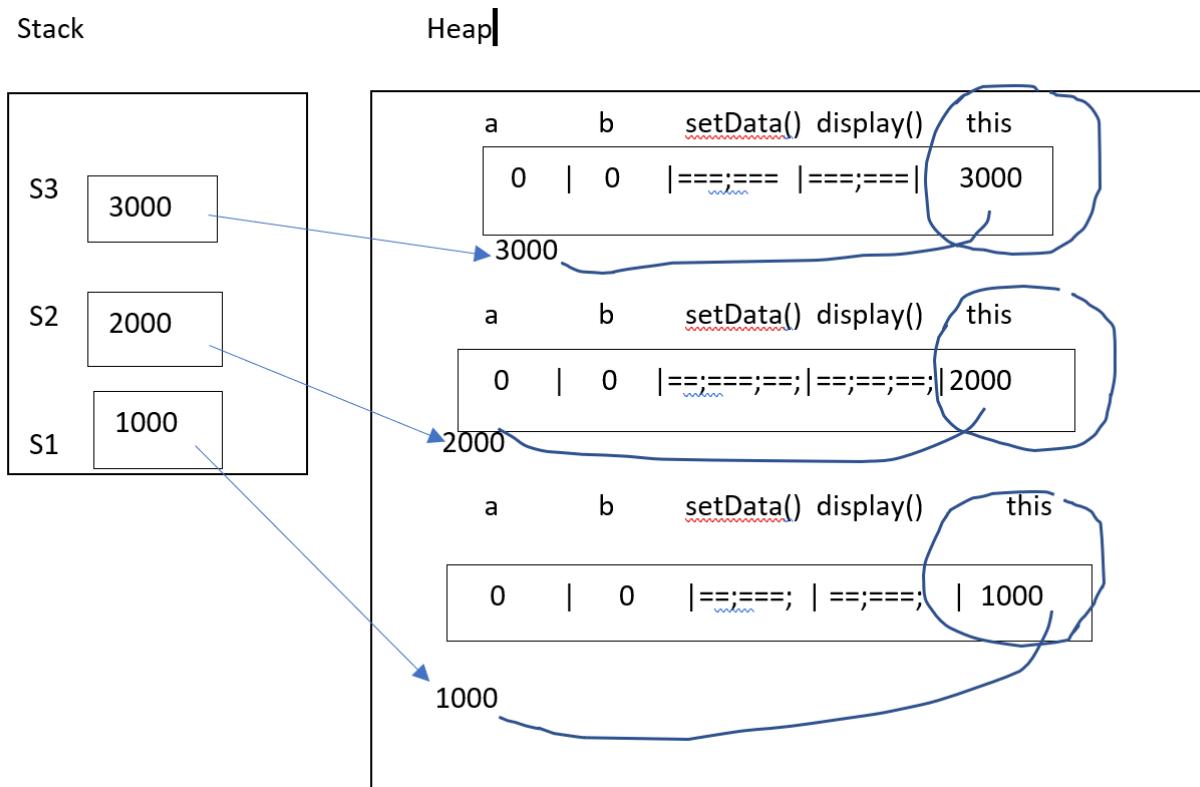
```
public class Sample
{
    private int a;

    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"+em
pDeptName+"\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"+em
pDeptName+"\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

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;

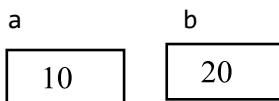
        int b = 20;

        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) {
        a           b
        int a=10;   int b=20;
        int b=20;   10          20
        swap method);                                     System.out.println("before call to
10 20                                         System.out.println(a+"\t"+b); →

        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;
    }
}

```

a	b
10	20

a	b
20	10

a	b
10	20

```
x=y;           x           y           temp
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;
    }

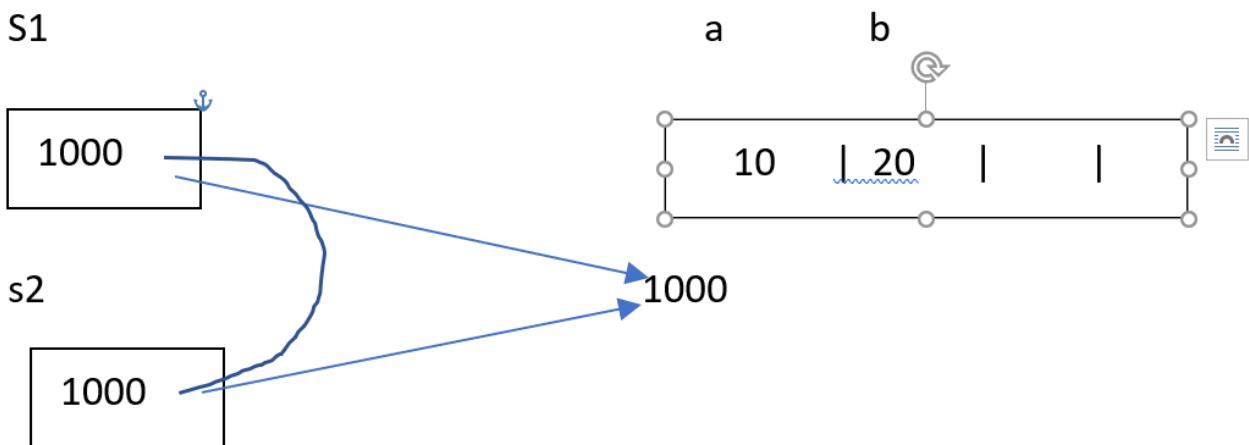
    public void display()
    {
        System.out.println("a= "+a+"\t b= "+b);
    }

}
```

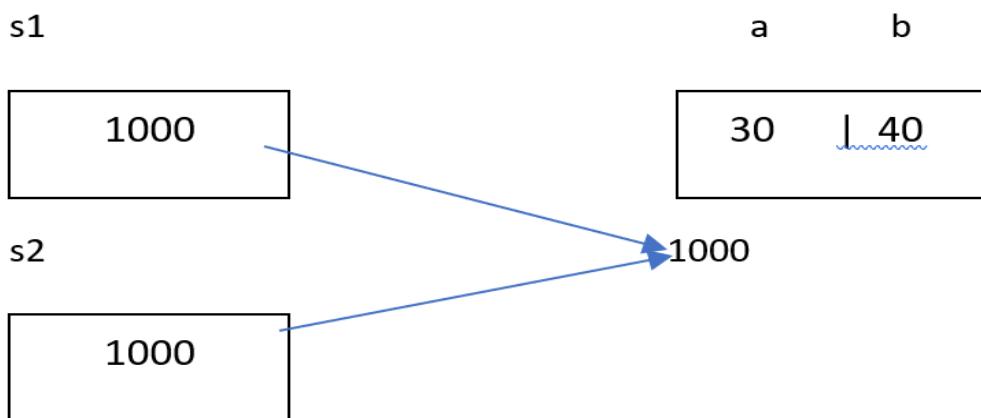
```
public class Test {  
  
    public static void main(String [] args)  
    {  
  
        Sample s1 = new Sample();  
  
        s1.setData(10, 20);  
  
        Sample s2 = s1;  
  
        System.out.println("initail values ");  
  
        s1.display();  
  
        s2.display();  
  
       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

```
initail 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.nrity.mnrao.test;

public class Sample {

    private int a;

    private int b;

    public void setData(int x, int y)
    {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```

a=x;

b=y;
}

public void display(){

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

}

```

```

public class Test
{

```

```

    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();

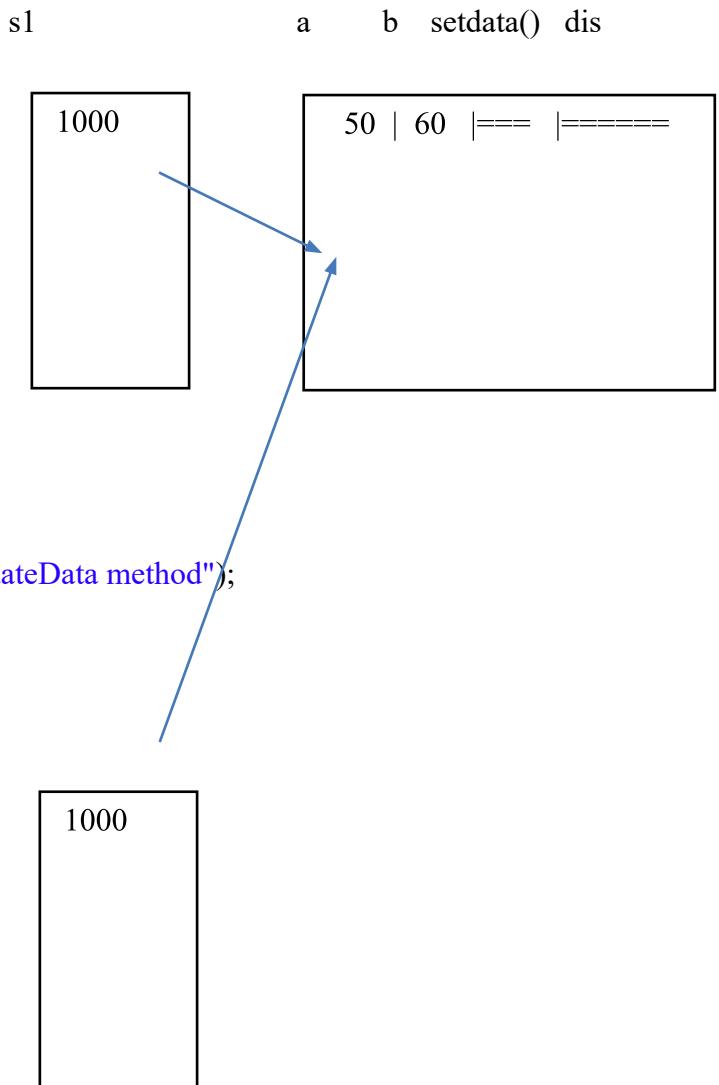
        updateData(s1);
    }

```

```

    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 argumnets, will reflect on actual copy of the Object .

When you 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;  
    }  
}
```

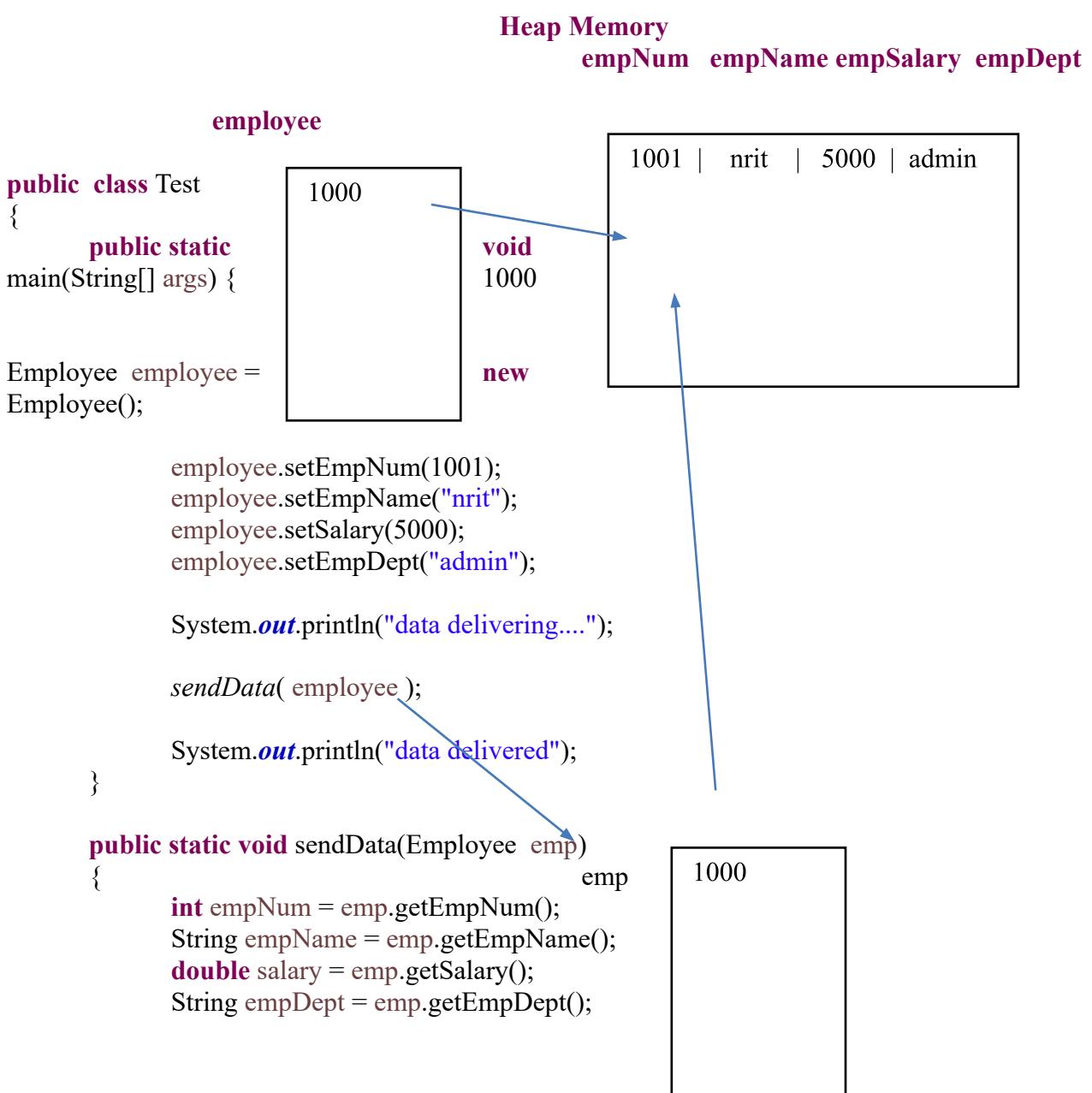
Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```

public String getEmpDept() {
    return empDept;
}

public void setEmpDept(String empDept) {
    this.empDept = empDept;
}
}

```



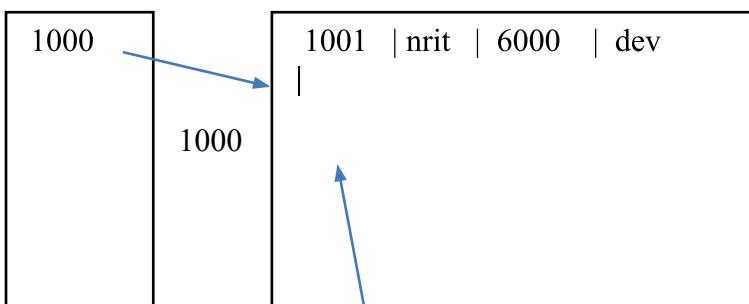
```
}
```

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");

}

}
```

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();

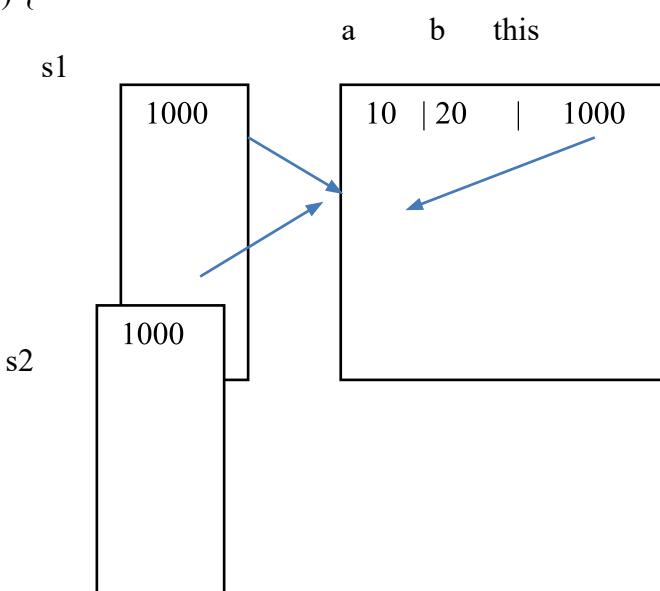
        System.out.println(empNum+"\t"+empName+"\t"+salary+"\t"+empDept);
    }

    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.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.

Method Overloading

Method overloading takes place between the same method with different signature

Method signature defines, name of the method, no.of args, type of args and return type of the method.

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 message.

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

Eg:

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");
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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  
  
    }  
}
```

2) Integer type of data passing real number type of arguments

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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 these two cases explicitly casting required

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

```
public class Sample {  
    public void setData(byte x)  
    {  
    }
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
    }

}

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 required

    }

}
```

1) Real number to integer

```
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 required  
        s1.setData( (int) d ); // real to int type, casting required  
    }  
}
```

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 (even 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.

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 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

These members can be shared by all objects of the class.

These can be accessed either by class name or through the object.

Eg:

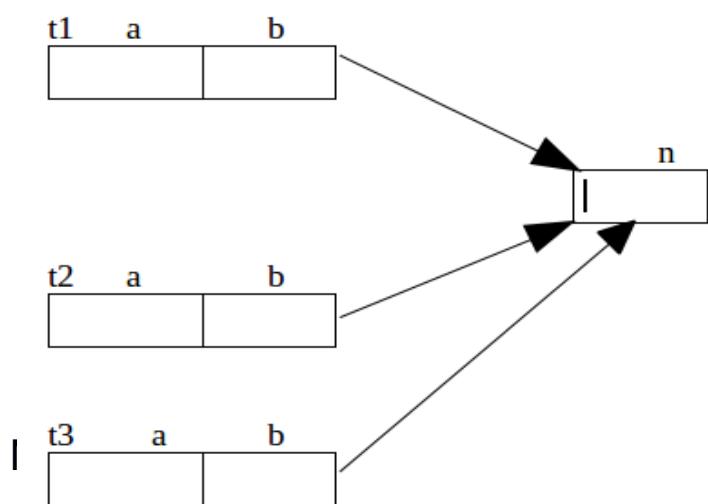
```
class Test {  
    int a, b ;  
  
    static int n :  
}
```

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 every object (three copies).

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

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

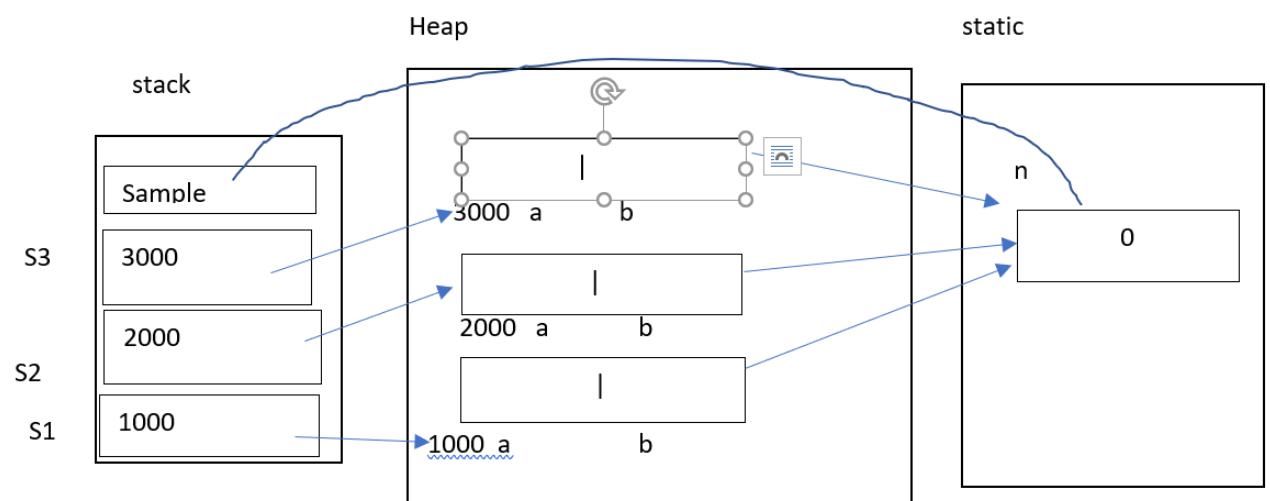
```
Test t3 = new Test();
```



Eg:

```
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.

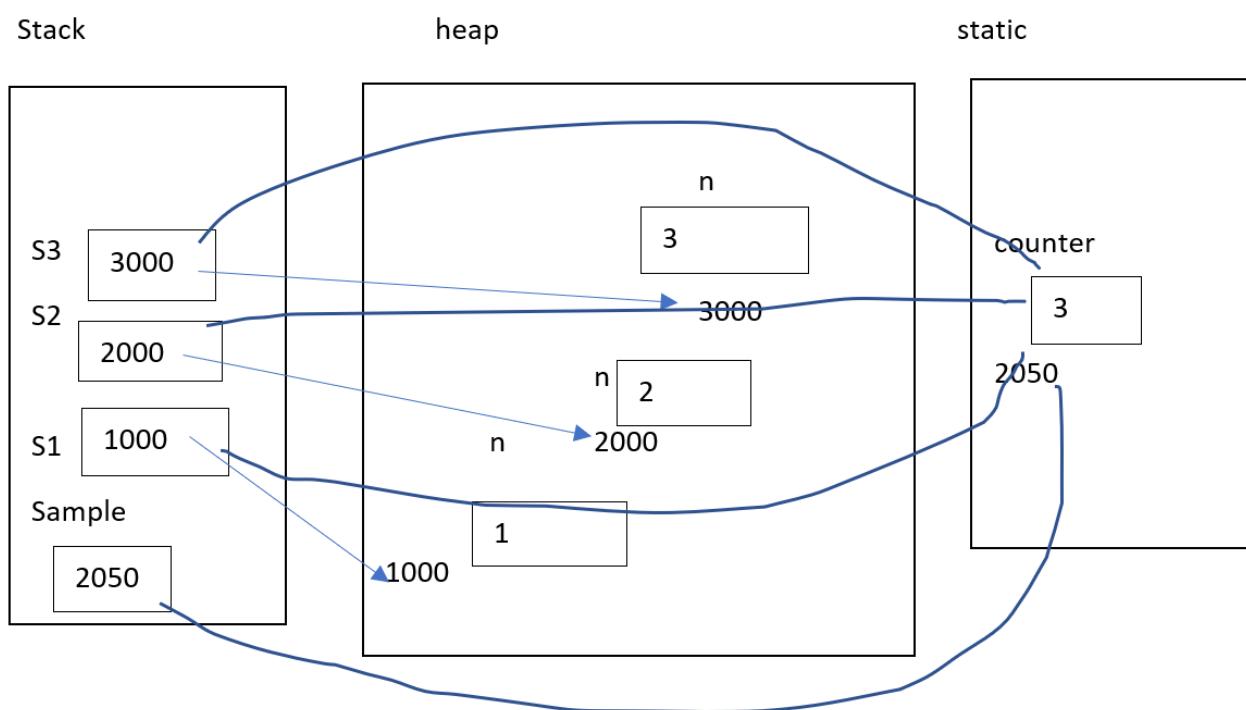
```
public class Sample {  
  
    private int n;  
  
    private static int counter;  
  
    public void setData() {  
        n = ++counter;  
    }  
  
    public void display() {  
        System.out.println ("Current Object " + n);  
        System.out.println ("Total Objects " + counter);  
    }  
}  
  
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();  
    }  
}
```

O/p:

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;
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        }
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 --> classname.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.

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

Static data member is class level.

Instance variable is a instance level.

Accessing static data member by the class name, 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 :

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

there are three types of variables:

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

eg:

```
public class Sample
{
    int a; // instance variable

    static int n; //class variable

    public void display()
    {
        int x; // local variable.

    }
}
```

local variables :

it is a variable, declared inside the methods

it is local to method

```
public void display()
{
    int a;
}
```

instance variables :

it is a variable, decalred in side the class .

it is a non-static varaiable.

it is a part of instance .

class variable :

it is a static variable, decalred inside the class .

it is not a part of instance .

it acquires with reference of class name .

since it acquires memory with reference of class name, it is called as class 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.

Eg:

```
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();  
    }  
}
```

}

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.

Eg :

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

eg:

```
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();

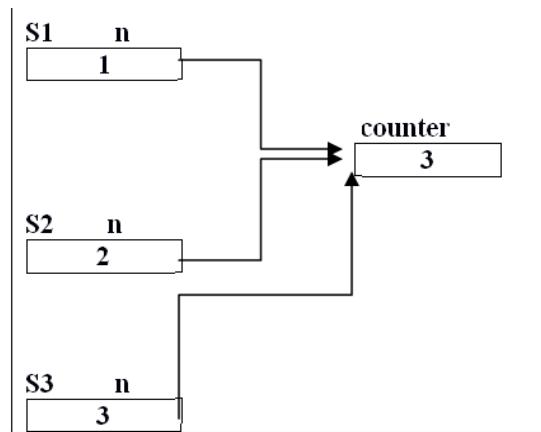
        s1.display();
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
s2.display();
s3.display();

        Sample.showCount();
    }

}
```



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:

1) instance method (non-static method)

it acquires memory with reference of object.

for this method object required

it is invoking through the object (instance)

instance method can access any data

it can make a call to any method , static as well non-static methods.

2) class method (static method)

it acquires memory with reference of class name.

for this method object is not required

it is invoking by the class name

static method can access only static data.

static method can make a call to only static methods.

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.

Eg:

```
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  
    }  
}
```

Eg:

```
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 = n " + Sample.n);
        System.out.println("class level = n " + 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.

eg:

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

why main() is static ?

ans:

since jvm makes a call to **main()** by the class name, static required .

jvm makes a call to main() by the class name, with out any object.

To invoke method by the class name , it has to acquire memory with reference of class name .

To acquire memory with reference of class name it should be static .

Only static method can be invoked b the class name.

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.*>

eg1:

package com.nritymnrao.y;

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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.nrity.mnrao.x;  
// below is a static import  
import static com.nrity.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.nrity.mnrao.x;  
  
import static com.nrity.mnrao.y.Sample.a;  
import static com.nrity.mnrao.y.Sample.b;  
import static com.nrity.mnrao.y.Sample.display;  
import static com.nrity.mnrao.y.Sample.show;  
  
public class Test {  
  
    public static void main(String[] args) {
```

```
System.out.println(a);
System.out.println(b);
display();
show();
}

}
```

Eg:

```
//System class importing as static

package com.nrity.mnrao.x;

import static java.lang.System.*;
import java.util.Scanner;

public class Test {

    public static void main(String[] args) {

        out.println("Hello world");

        err.println("error");

        Scanner sc = new Scanner(in);

        out.println("enter a number");

        int n = sc.nextInt();

        out.println("your number =" + n);
    }
}
```

Constructors

Constructor is a member of the class, it is like a method, which is invoked automatically when object is created.

The purpose of the constructor is to initialize objects with required values.

Constructors are invoked implicitly when object is created.

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) 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 :

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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;
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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);

    }
}
```

Eg:

Another example:

```
public class DatabaseServer {  
    private String dbHost;  
    private String dbName;  
    private int dbPortNum;  
    private String dbUserId;  
    private String dbPassword;  
  
    public DatabaseServer()  
    {  
        dbHost = "localhost";  
        dbName = "test";  
        dbPortNum = 1527;  
        dbUserId = "scott";  
        dbPassword = "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 int getDbPortNum() {  
        return dbPortNum;  
    }  
}
```

```
}

public void setDbPortNum(int dbPortNum) {
    this.dbPortNum = dbPortNum;
}

public String getDbUserId() {
    return dbUserId;
}

public void setDbUserId(String dbUserId) {
    this.dbUserId = dbUserId;
}

public String getDbPassword() {
    return dbPassword;
}

public void setDbPassword(String dbPassword) {
    this.dbPassword = dbPassword;
}

}

package com.durga.mnrao.oop;

public class Test {

    public static void main(String[] args) {

        DatabaseServer databaseServer = new DatabaseServer();

        System.out.println("Db Server initial details ");

        System.out.println( databaseServer.getDbHost() );
        System.out.println( databaseServer.getDbName() );
        System.out.println( databaseServer.getDbPortNum() );
        System.out.println( databaseServer.getDbUserId() );
        System.out.println( databaseServer.getDbPassword() );

        databaseServer.setDbHost("192.168.56.101");
        databaseServer.setDbName("prod");
        databaseServer.setDbPortNum(1521);
        databaseServer.setDbUserId("mnrao");
        databaseServer.setDbPassword("welcome");

        System.out.println("after changing Database details ");

        System.out.println( databaseServer.getDbHost() );
        System.out.println( databaseServer.getDbName() );
    }
}
```

```
        System.out.println( databaseServer.getDbPortNum() );
        System.out.println( databaseServer.getDbUserId() );
        System.out.println( databaseServer.getDbPassword() );
    }
}
```

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 constructors are used to initialize the object with different values .

Class with both default and parameterized constructors.

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public class Sample {  
    private int a;  
    private int b;  
  
    public Sample()  
    {  
        a = 0;  
  
        b = 0;  
    }  
  
    public Sample( 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();  
  
        Sample s2 = new Sample(10,20);  
  
        s1.display();  
        s2.display();  
  
    }  
  
}
```

if class **don't have** constructors, then compiler generates a default constructor for every class.

If class **contains** at least one user defined constructor, then compiler will not generate default constructor.

If class contains constructors then one of the constructors must be a default constructor.

when you defining a class with constructors,
then one of the constructors must be a default constructor.

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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();  
    }  
}
```

Initializing Object :

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

An Object can be initialized in three ways

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

class level initialization :

```
public class Sample {  
  
    private int a = 10;  
  
    private int b = 20;  
  
    public void display()  
    {  
        System.out.println("a = "+a+"\t b = "+b);  
    }  
  
}  
  
package com.durga.mnrao.oop;  
  
public class Test {  
  
    public static void main( String [] args) {  
  
        Sample s1 = new Sample();  
  
        s1.display();  
    }  
}
```

o/p:
a = 10 b = 20

Initialize / instance block :

```
public class Sample {  
  
    {  
        System.out.println("instance block");  
    }  
}
```

}

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

```
package com.durga.mnrao.oop;

public class Sample {

    {
        System.out.println("instance block invoked");
    }

    public Sample()
    {
        System.out.println("Constructor invoked ");
    }

}
```

```
package com.durga.mnrao.oop;

public class Test {

    public static void main( String [] args) {

        Sample s1 = new Sample();
    }

}
```

o/p:
instance block invoked
Constructor invoked

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

If class contains multiple initialize blocks then order of execution is top to bottom.

Purpose of initialize block is to initialize constants.

Static block :

```
public class Sample {  
  
    static  
    {  
        System.out.println("static block invoked");  
    }  
  
}
```

Static block executes at the time of loading .class file into memory

when first object is created, .class file loads into memory .

static block also loads into memory at the time of loading .class file.

when 1st object is created , sequence of operations as below

.class file loads

static block loads

object creates

instance block invokes

constructor invokes

when 2nd object is created

instance block invokes

constructor invokes

.class file and static block loads only once.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

static block executes only for the first time created object

static block loads only once.

it is only one copy for any no of objects.

for next time object (from 2nd object) only instance block and constructor executes

class with static block , instance block and constructor

```
public class Sample {  
  
    static  
    {  
        System.out.println("static block invoked");  
    }  
  
    {  
        System.out.println("instance block invoked ");  
    }  
  
    public Sample()  
    {  
        System.out.println("Constructor invoked ");  
    }  
}  
  
package com.durga.mnrao.oop;  
  
public class Test {  
  
    public static void main( String [] args) {  
  
        Sample s1 = new Sample();  
  
        Sample s2 = new Sample();  
  
        Sample s3 = new Sample();  
  
    }  
}
```

O/p :

static block invoked

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
instance block invoked  
Constructor invoked  
instance block invoked  
Constructor invoked  
instance block invoked  
Constructor invoked
```

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

Static block can be defined anywhere in the class.

A class can have multiple static blocks. The order of execution is from top to bottom.

purpose of **constructor** is to initialize variables

purpose of **instance block** is to initialize constants (final)

purpose of **static block** is to initialize static constants (final and static)

Constructor overloading :

This is similar to method overloading,

it takes place based on number of arguments and type of arguments.

At compile time, compiler first checks for the number of args, if no. of args are matched then checks for the type of args, if type of args is also matched between constructors then ambiguity, between the constructors and compilation error.

Overloading is based on number of arguments.

```
public class Sample {  
  
    public Sample()  
    {  
        System.out.println("No args Constructor ");  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public Sample(int x)
{
    System.out.println("One args Constructor   ");
}

public Sample(int x, int y )
{
    System.out.println("Two args Constructor   ");
}

public Sample(int x, int y, int z )
{
    System.out.println("Three args Constructor   ");
}

package com.durga.mnrao.oop;

public class Test {

    public static void main( String [] args) {

        Sample s1 = new Sample();

        Sample s2 = new Sample(1);

        Sample s3 = new Sample(10, 20);

        Sample s4 = new Sample(100,200,300);

    }
}
```

o/p:

No args Constructor
One args Constructor
Two args Constructor
Three args Constructor

Over loading with data

```
public class Sample {

    private int a;

    private int b;
```

```
private int c;

public Sample()
{
    a = 0;

    b = 0;

    c = 0;
}

public Sample(int x)
{
    a = x;

    b = 0;

    c = 0;
}

public Sample(int x, int y )
{
    a = x;

    b = y;

    c = 0;
}

public Sample(int x, int y, int z )
{
    a = x;

    b = y;

    c = z;
}

public void display()
{
    System.out.println("a = "+a+"\t b= "+b+"\t c = "+c);
}

package com.durga.mnrao.oop;

public class Test {
```

```
public static void main( String [] args) {  
    Sample s1 = new Sample();  
    Sample s2 = new Sample(1);  
    Sample s3 = new Sample(10, 20);  
    Sample s4 = new Sample(100,200,300);  
    System.out.println("s1 data");  
    s1.display();  
    System.out.println("s2 data");  
    s2.display();  
    System.out.println("s3 data");  
    s3.display();  
    System.out.println("s4 data");  
    s4.display();  
}  
}  
  
o/p:  
  
s1 data  
a = 0 b= 0 c = 0  
  
s2 data  
a = 1 b= 0 c = 0  
  
s3 data  
a = 10 b= 20 c = 0  
  
s4 data  
a = 100 b= 200 c = 300
```

Constructor Overloading based on type of arguments

```
public class Test {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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();
    }
}
```

```
}
```

Invoking current class / same class constructor:

```
package com.durga.mnrao.oop;

public class Sample {

    public Sample()
    {
        System.out.println("No args constructor");
    }

    public Sample(int x)
    {
        this();
        System.out.println("One args constructor");
    }

    public Sample(int x, int y )
    {
        this(x);

        System.out.println("Two args constructor");
    }

    public Sample(int x, int y, int z )
    {
        this(x, y); // calling same class constructor
        System.out.println("Three args constructor");
    }
}

public class Test {

    public static void main( String [] args) {

        Sample s1 = new Sample(1, 2, 3);
    }
}
```

```
}
```

```
}
```

O/P:

```
No args constructor  
One args constructor  
Two args constructor  
Three args constructor
```

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

Eg:

```
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 calling is not valid for constructors.  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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 acquires old class properties into new class.

Old class is a parent class and new class is child class.

Purpose of Inheritance is code reusability.

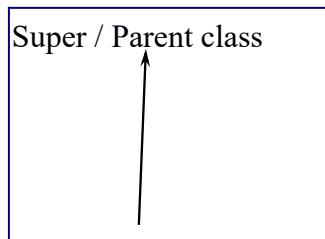
As per Java , Parent class is called as super class
And child class is called as sub class.

Types of inheritance :

Java provides three types of inheritance:

- 1) Single inheritance
- 2) Multi level inheritance
- 3) Hierarchical Inheritance

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.

Eg:

```
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();
    }
}
```

Method Overriding :

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 {

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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.

Eg:

```
public void display()  
{
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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();
    }
}
```

Eg:

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);
    }
}

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 :

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)

```
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;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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();
    }
}
```

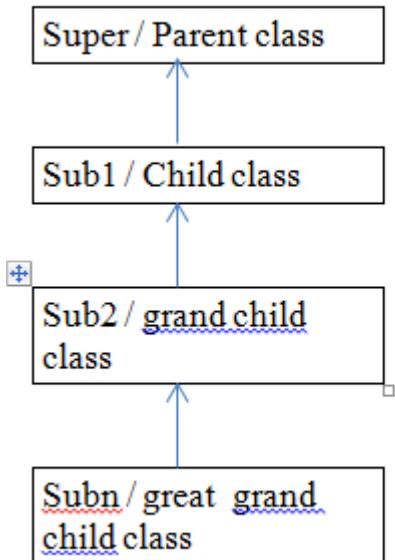
Private access specifier.

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 :



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();

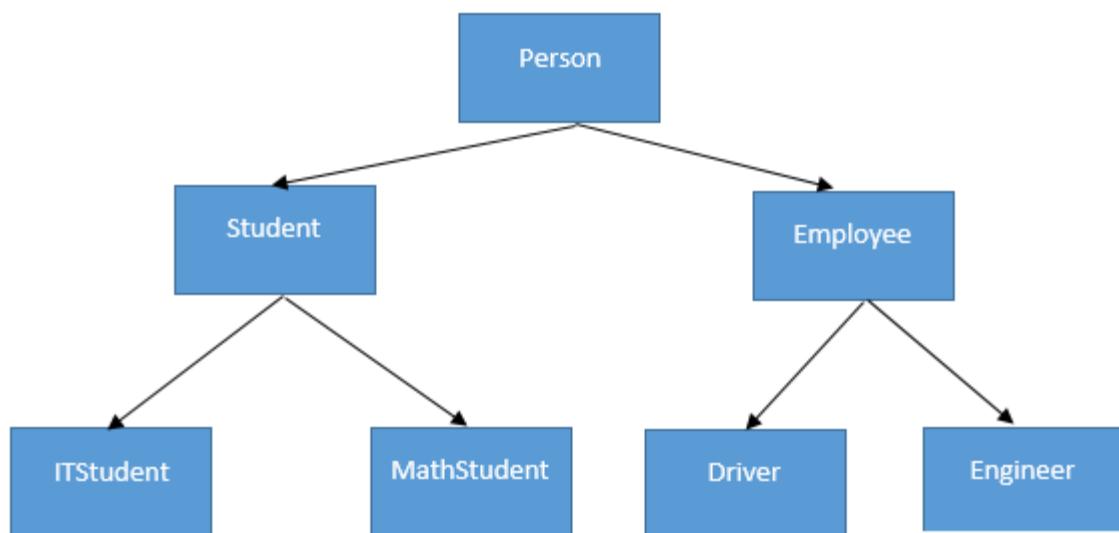
        System.out.println("=====");
        t.setData(1, 2, 3, 4);

        t.display();

        System.out.println("=====");
        t.setData(5, 15, 25, 35, 45, 55);
    }
}
```

```
    t.display();  
}  
  
}
```

Hierachial Inheritance:



eg:

```
package com.nrit.mnrao.hierarchical;  
  
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.hierachial;

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 solutions .

```
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 (no access specifier)

public is having more scope;

Descending order is

public,
protected,
default
private

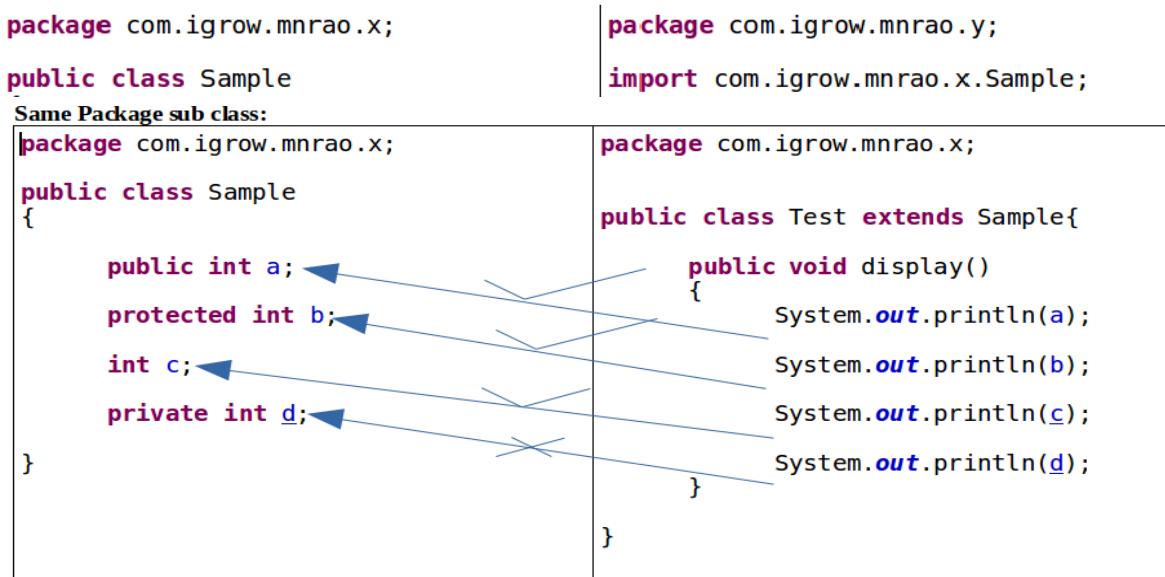
Scopes :

These are four levels

- 1) Outside the Java application world
- 2) From other packages (sub class)
- 3) Same package (sub class)
- 4) Within the same class

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

Other package sub class:



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
    }
}
```

as

to
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 retrive data from the object.

Code recommendations :

public is recommended for **static data** members, it is accessing by the class name.

access private

Access Modifiers

It defines **behaviour** of classes and it's members.

1) static

2) final

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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

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) if 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 since 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.  
    }  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        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 {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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
}
```

eg2:

```
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
    }
}
```

Eg:

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public class Sample
{
    final int A;

    {
        A=10;//valid as it is a first time assignment.
             // valid since initiliaze 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 / instance block is used to initialize **final data members** (its value can not be changed)

Eg:

```
public class DataBase {

    private String hostName;

    private String userId;

    private String password;

    final private String DB_PRODUCT;

    final private int PORT_NUM;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
// initialize block for constants .  
  
{  
    DB_PRODUCT = "mysql";  
  
    PORT_NUM = 3306;  
}  
  
// constructors for variable  
  
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 setId(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 methdos are to get the value (returning value)

Since Constants value can not be changed , final data members, don't have setter methods for

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

constants, 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, whereas 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 {  
    private static final 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- 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 initailize variables

instance block :

=====

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

to initialize **final** variables

static block :

=====

to initialize **final and static** variable

method as final:

it prevents from overriding in the child class:

eg:

```
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 classe, parent as well as child.**

class as final :

it prevents from the inheritance. Final class members can not be inherited into child class.

Eg:

```
final public class Sample
{
}

// in the below, extends Sample is not valid
public class Test extends Sample
{
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

}

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.

Concrete class **should** contain all concrete methods only

Abstract it is a partially implemented class.

Abstract class does not allow to create an object but allows to declare reference .

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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 (definitaion)
- 2) concrete method → it is a method with functionality (definitaion)

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 atleast one abstract method, then class also becomes as an abstract class.

```
public abstract class Sample
{
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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.

Eg:

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 definition

2) Abstract Method:

it is a declaration , don't have definition.

if class contains atleast 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

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**
- 2) interface data members are by default **public , static and final.**
- 3) 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
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public void display()
{
}

@Override
public void show()
{
}

}
```

all methods of an interface should be implemented in the child class, otherwise child class becomes an abstract class.

eg:

```
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

Eg:

```
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 ");  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        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 --> it is a method from parent class, which is overriding
at child class

@Override --> it is an optional ,
but to understand code , it is required
to identify parent class and child class methods it is required
it can not be used with child class own methods

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 {
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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() {
        // TODO Auto-generated method stub
    }

    @Override
    public void show() {
        // TODO Auto-generated method stub
    }

    @Override
    public void demo() {
        // TODO Auto-generated method stub
    }

    @Override
    public void put() {
        // TODO Auto-generated method stub
    }
}
```

Inheritance between classes and interface:

Parent	-->	class	interface	interface	class
		extends	implements	extends	Not Applicable

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

child --> **class** **class** **interface** **interface**

a class can implements 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() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void show() {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void demo() {  
        // TODO Auto-generated method stub  
    }  
  
    @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
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

{

}

If all Parent interfaces contains same data member, then duplicate copies creates at child class level, then it is an ambiguous 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 {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 Applicable

Child --->	class	class	interface	interface
----------------------	-------	-------	-----------	-----------

a class can implement multiple interfaces but extends of only class.

final Syntax of class :

```
accessSpecifier class childClassName extends ParentClassName implements Inter1, Inter2,  
Inter3...
```

```
{
```

```
}
```

Adapter class:

below interface using at project level

```
package com.durga.mnrao.x;
```

```
public interface MyInterface {
```

```
    public abstract void m1();
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 impelment

```
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 MyInterface ,

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 m10 {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void m20 {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void m30 {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void m40 {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void m50 {  
        // TODO Auto-generated method stub  
    }  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
}

@Override
public void m60 {
    // TODO Auto-generated method stub
}

@Override
public void m70 {
    // TODO Auto-generated method stub
}

@Override
public void m80 {
    // TODO Auto-generated method stub
}

@Override
public void m90 {
    // TODO Auto-generated method stub
}

@Override
public void m100 {
    // 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");
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
}

@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:

Polymorphism

Poly --> many

Morphism --> forms

def:

Same method, behaving differently for different purpose is called as polymorphism.

Same method, defining differently for different purpose is called as polymorphism.

Eg:

Picture (interface) → Parent

draw() → abstract

Circle Rectangle Triangle → child classes

draw() draw() draw() → Method Overriding

```
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

```

public interface Picture {
    public abstract void draw();
}

at runtime, call to child class method

public class Circle implements Picture{
    @Override
    public void draw() {
        // TODO Auto-generated method stub
        System.out.println(" Circle ");
    }
}

public class Rectangle implements Picture{
    @Override
    public void draw() {
        // TODO Auto-generated method stub
        System.out.println(" Rectangle ");
    }
}

public class Triangle implements Picture{
    @Override
    public void draw() {
        // TODO Auto-generated method stub
        System.out.println(" Triangle ");
    }
}

```

```

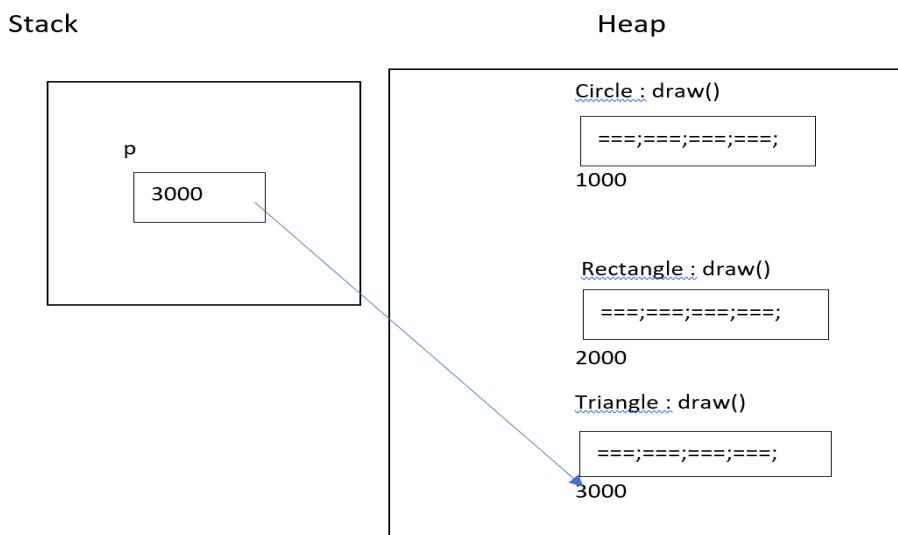
public class Test {
    public static void main(String[] args) {
        Picture p = new Circle();
        p.draw();

        p = new Rectangle();
        p.draw();

        p = new Triangle();
        p.draw();
    }
}

```

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 method argument type 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();
    }
}
```

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 {
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 args[])  
    {  
  
        Bank bank = new CitiBank();  
  
        System.out.println (bank.getRateOfInterest());  
  
        bank = new HdfcBank();  
  
        System.out.println (bank.getRateOfInterest());  
    }  
}
```

Ploymorphism 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;  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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() );

    }

    // if method argument type is parent type, then you can pass any child class instance
    // parent reference and child class instance

    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
=====

if program executed as per compiler linking , that is called as static binding

if compiler knows about run time process that is called static binding

it is a method overloading

it is called as compile time polymorphism

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

it happens in the same class

dynamic binding

=====

if program **not** executed as per compiler linking , that is called as dynamic binding

if compiler don't know about run time process, that is called as dynamic binding

it is a Method Overriding

it is called as run time polymorphism

it happens between parent and child

Static Binding	Dynamic Binding
If program executed as per compiler linking, then it is called as static binding	If program not executed as per compiler linking, then it is called as dynamic binding
If compiler knows about run time process, then it is a static binding	If compiler don't know about run time process, then it is a dynamic binding
Method overloading	It is Method Overriding
It happens in the same class	It happens between parent and child classes
Compile time polymorphism	Runtime polymorphism

Interview Questions

1.What is an abstract class.

Ans: abstract class, partially implemented class, if class contains atleast one abstract method, then class becomes abstract,
abstract class doesn't allow to create object.

2. Can we declare a class as Abstract without having any abstract method?

Ans: Yes we can create an abstract class by using abstract keyword before class name even if it doesn't have any abstract method. However, if a class has even one abstract method, it must be declared as abstract otherwise it will give an error.

3.What's the difference between an Abstract Class and Interface in Java?

Ans: The primary difference between an abstract class and interface is that an interface can only possess declaration of public static methods with no concrete implementation while an abstract class can have members with any access specifiers (public, private etc) with or without concrete

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

implementation.

Another key difference in the use of abstract classes and interfaces is that a class which implements an interface must implement all the methods of the interface while a class which inherits from an abstract class doesn't require implementation of all the methods of its super class.

A class can implement multiple interfaces but it can extend only one abstract class.

4. Does Importing a package imports its sub-packages as well in Java?

Ans: In java, when a package is imported, its sub-packages aren't imported and developer needs to import them separately if required.

For example, if a developer imports a package university.* , all classes in the package named university are loaded but no classes from the sub-package are loaded. To load the classes from its sub-package (say department), developer has to import it explicitly as follows:

Import university.department.*

5. Can we declare the main method of our class as private?

Ans: In java, main method must be public static in order to run any application correctly. If main method is declared as private, developer won't get any compilation error however, it will not get executed and will give a runtime error.

6. Can we override static methods of a class?

Ans: We cannot override static methods. Static methods belong to a class and not to individual objects and are resolved at the time of compilation (not at runtime). Even if we try to override static method, we will not get an compilation error, nor the impact of overriding when running the code.

7. Can a class be a super class and a sub-class at the same time? Give example.

Ans: If there is a hierarchy of inheritance used, a class can be a super class for another class and a sub-class for another one at the same time.

8. There are two classes named classA and classB. Both classes are in the same package. Can a private member of classA be accessed by an object of classB?

Ans: Private members of a class aren't accessible outside the scope of that class and any other class even in the same package can't access them.

9. What's the benefit of using inheritance?

Ans: Key benefit of using inheritance is reusability of code as inheritance enables sub-classes to reuse the code of its super class. Polymorphism (Extensibility) is another great benefit which allow new functionality to be introduced without effecting existing derived classes.

10. Give an example of use of Pointers in Java class.

Ans: There are no pointers in Java. So we can't use concept of pointers in Java.

11.How can we restrict inheritance for a class so that no class can be inherited from it?

Ans: If we want a class not to be extended further by any class, we can use the keyword Final with the class name.

12.What's the access scope of Access specifiers?

	Outside the Java 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	No	Yes	Yes
private	No	No	No	Yes

13.What is overloading and overriding in java?

When we have more than one method with same name in a single class but the arguments are different, then it is called as method overloading.

Overriding concept comes in picture with inheritance when we have two methods with same signature, one in parent class and another in child class. We can use @Override annotation in the child class overridden method to make sure if parent class method is changed, so as child class.

14.What is final keyword?

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can't extend it.

We can use final keyword with methods to make sure child classes can't override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

Java interface variables are by default final and static.

15.What is an interface?

Interfaces are core part of java programming language and used a lot not only in JDK but also java design patterns, most of the frameworks and tools. Interfaces provide a way to achieve abstraction in java and used to define the contract for the subclasses to implement.

Interfaces are good for starting point to define Type and create top level hierarchy in our code. Since

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

a java class can implements multiple interfaces, it's better to use interfaces as super class in most of the cases

16.What is an abstract class?

Abstract classes are used in java to create a class with some default method implementation for subclasses. An abstract class can have abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class. Abstract classes can't be instantiated and mostly used to provide base for sub-classes to extend and implement the abstract methods and override or use the implemented methods in abstract class.

17.What is the difference between abstract class and interface?

What is the difference between abstract class and interface?

abstract keyword is used to create abstract class whereas interface is the keyword for interfaces.

Abstract classes can have method implementations whereas interfaces can't.

A class can extend only one abstract class but it can implement multiple interfaces.

We can run abstract class if it has main() method whereas we can't run an interface.

18.Can an interface implement or extend another interface?

Interfaces don't implement another interface, they extend it. Since interfaces can't have method implementations, there is no issue of diamond problem. That's why we have multiple inheritance in interfaces i.e an interface can extend multiple interfaces.

19.What is instanceof keyword?

We can use instanceof keyword to check if an object belongs to a class or not. We should avoid it's usage as much as possible. Sample usage is:

```
public static void main(String [] args)
{
    Object str = new String("abc");

    if(str instanceof String){
        System.out.println ("String value:"+str);
    }

    if(str instanceof Integer){
        System.out.println ("Integer value:"+str);
    }
}
```

20.Java is Pass by Value or Pass by Reference?

This is a very confusing question, we know that object variables contain reference to the Objects in heap space. When we invoke any method, a copy of these variables is passed and gets stored in the stack memory of the method.

Passing object is call by reference where as passing variable is a call by value.

21.What is Polymorphism?

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

22.Can a constructor be made final?

No, this is not possible

23.What is Dynamic Binding(late binding)?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.

24.Can constructor be inherited?

No, constructor cannot be inherited.

25.Can we instantiate an interface?

You can't instantiate an interface directly, but you can instantiate a class that implements an interface.

26.Do interfaces have member variables?

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

27.Can we instantiate an abstract class?

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

28.What are the differences between Interface and Abstract class?

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code at all, just the signature.
In case of abstract class, a class may extend only one abstract class.	A Class may implement several interfaces.
An abstract class can have non-abstract methods.	All methods of an Interface are abstract.
An abstract class can have instance variables.	An Interface cannot have instance variables.
An abstract class can have any visibility: public, private, protected.	An Interface visibility must be public (or) none.
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method.
An abstract class can contain constructors .	An Interface cannot contain constructors .
Abstract classes are fast.	Interfaces are slow as it requires extra indirection to find corresponding method in the actual class.

29. When should I use abstract classes and when should I use interfaces?

Use Interfaces when...

You see that something in your design will change frequently.

If various implementations only share method signatures then it is better to use Interfaces.

you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

Use Abstract Class when...

If various implementations are of the same kind and use common behavior or status then abstract class is better to use.

When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.

Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

30. can there be an abstract class with no abstract methods in it?

Yes, there can be an abstract class without abstract methods.

31) Which class is the superclass of all classes?

`java.lang.Object` is the root class for all the java classes and we don't need to extend it.

32) What is Marker interface?

A marker interface is an empty interface without any method but used to force some functionality in implementing classes by Java. Some of the well known marker interfaces are Serializable and Cloneable.

33) Can you use this() and super() both in a constructor?

No. Because `super()` or `this()` must be the first statement.

34) Why we cannot override static method?

It is because the static method is the part of class and it is bound with class whereas instance method is bound with object and static gets memory in class area and instance gets memory in heap.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

35) Can we override the overloaded method?

Yes.

36) Can we define private and protected modifiers for variables in interfaces?

No, they are implicitly public.

37) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

Garbage Collection

Garbage collection

it is a process of cleaning the garbage generated by the java application

in java all unused objects are treated as garbage.

un referenced object is a garbage object.

Garbage Collector clears all un used objects

Garbage Collector is a daemon thread in JVM.

daemon --> background running thread or process.

it will run parallelly with java program.

garbage collector invoking automatically , when CPU goes to idle state.

if no program running in the system then CPU goes to idle state.

if cpu is busy for 24/7 , no idle time , then garbage collector never invoking

in that case , java developer can invoke garbage at regular intervals

invoking garbage collector :

```
System.gc();
```

when object is cleared or destroyed, then automatically call to finalize()

finalize() is a override method from the Object class.

since Object class is default parent of every class, finalize() can override in every class .

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();

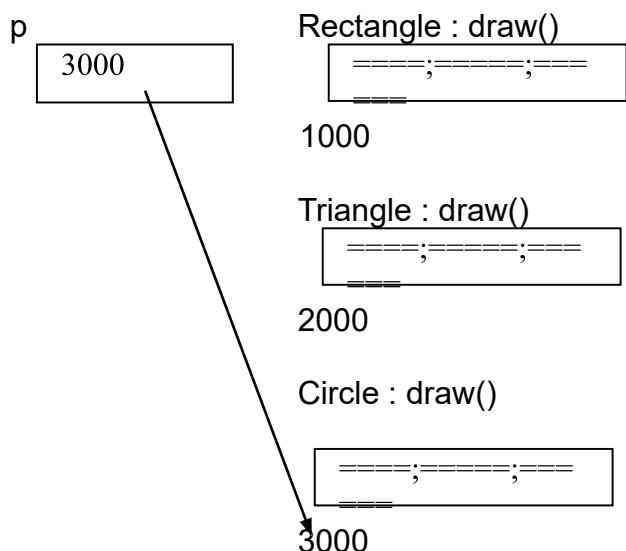
        System.gc();
    }
}
```

}

o/p:

```
drawing Rectangle
drawing Triangle
drawing Circle
cleaning Rectangle object
cleaning Triangle object
```

Memory Map for the above Program :



In the above Rectangle and Triangle objects are unreferenced , treated as garbage .

When called to Garbage collector , these objects get cleared from Heap Memory by the Garbage collector.

Nested Classes :

These are inner classes :

if any field contains multiple value, that case we can use inner class.

```
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);

}
```

```
}
```

Strings

String is a class from java.lang package.

It is not a basic data type. It is user defined type (non-primitive)

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.

Eg:

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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);  
    }  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```

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	<code>char charAt(int index)</code>	returns char value for the particular index
2	<code>int length()</code>	returns string length
5	<code>String substring(int beginIndex)</code>	
6	<code>String substring(int beginIndex, int endIndex)</code>	returns substring for given begin index and end index
7	<code>boolean contains(CharSequence s)</code>	returns true or false after matching the sequence of char value
10	<code>boolean equals(Object another)</code>	checks the equality of string with object
11	<code>boolean isEmpty()</code>	checks if string is empty
13	<code>String replace(char old, char new)</code>	replaces all occurrences of specified char value
14	<code>String replace(CharSequence old, CharSequence new)</code>	replaces all occurrences of specified CharSequence CharSequence new)
15	<code>String trim()</code>	returns trimmed string omitting leading and trailing spaces
16	<code>String [] split(String regex)</code>	returns splitted string matching regex
19	<code>int indexOf(int ch)</code>	returns specified char value index
20	<code>int indexOf(int ch, int fromIndex)</code>	returns specified char value index starting with given index
21	<code>int indexOf(String substring)</code>	returns specified substring index
22	<code>int indexOf(String substring, int fromIndex)</code>	returns specified substring index starting with given index
23	<code>String toLowerCase()</code>	returns string in lowercase.
25	<code>String toUpperCase()</code>	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:

- By equals() method
- By == operator
- 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

Eg:

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 ");
        }
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        }
    }
}
```

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 many 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” placed 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");  
        }  
    }  
}  
o/p; 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");  
}  
}  
}
```

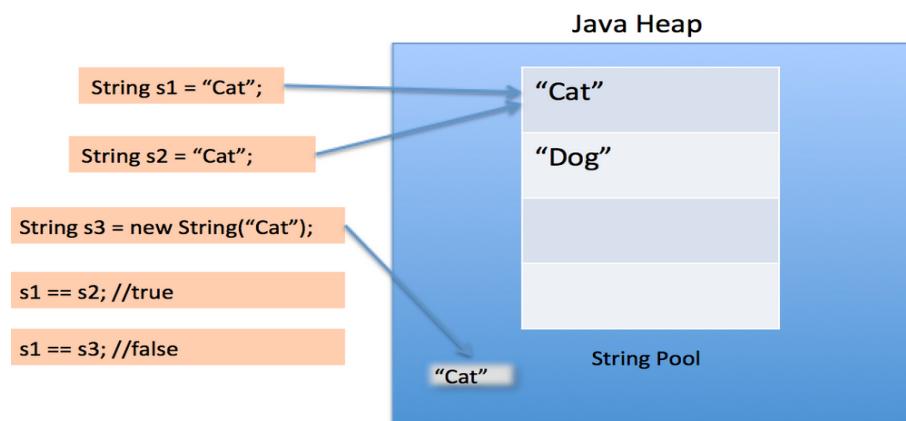
o/p :
different location

```
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

What is 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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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));  
    }  
}
```

O/P:

```
s1 == s2 :true  
s1 == s3 :false
```

how many strings 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
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
    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;
                }
            }
        }
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        }
    }
}

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 concat 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"
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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.

Spilt() 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]);  
    }  
}
```

O/p:

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.nrity.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 getUserId()
    {
        return uid;
    }

    public String getUserName(){
        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.getId()+"\t"+u1.getName());
        System.out.println(u2.getId()+"\t"+u2.getName());
        System.out.println(u3.getId()+"\t"+u3.getName()); }

    }
}
```

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,

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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 StringBuffer class is used to create mutable (modifiable) string. The StringBuffer 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 StringBuffer 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. Old data shifted new location at the time concatenation	StringBuffer is fast and consumes less memory when you concat strings. It grows dynamically.
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 StringBuffer 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.

1. Is String a data type in java?

Ans: String is not a primitive data type in java. When a string is created in java, it's actually an object of Java.Lang.String class that gets created. After creation of this string object, all built-in methods of String class can be used on the string object.

2. Why Strings in Java are called as Immutable?

Ans: In java, string objects are called immutable as once value has been assigned to a string, it can't be changed and if changed, a new object is created.

3. When a lot of changes are required in data, which one should be a preference to be used? String or StringBuffer?

Ans: Since StringBuffers are dynamic in nature and we can change the values of StringBuffer objects unlike String which is immutable, it's always a good choice to use StringBuffer when data is being changed too much. If we use String in such a case, for every data change a new String object will be created which will be an extra overhead.

4. How can we use primitive data types as objects?

Ans: Primitive data types like int can be handled as objects by the use of their respective wrapper classes. For example, Integer is a wrapper class for primitive data type int. We can apply different

methods to a wrapper class, just like any other object.

5. Can we use a default constructor of a class even if an explicit constructor is defined?

Ans: Java provides a default no argument constructor if no explicit constructor is defined in a Java class. But if an explicit constructor has been defined, default constructor can't be invoked and developer can use only those constructors which are defined in the class.

6.What are Wrapper classes?

Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing and unboxing allows easy conversion between primitive types and their corresponding wrapper classes.

7) Why Java is not pure Object Oriented language?

Java is not said to be pure object oriented because it supports primitive types such as int, byte, short, long etc. I believe it brings simplicity to the language while writing our code. Obviously java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.

As we know, for all the primitive types we have wrapper classes such as Integer, Long etc that provides some additional methods.

8) What is the use of System class?

Java System Class is one of the core classes. One of the easiest way to log information for debugging is System.out.print() method.

System class is final so that we can't subclass and override its behavior through inheritance. System class doesn't provide any public constructors, so we can't instantiate this class and that's why all of its methods are static.

Some of the utility methods of System class are for array copy, get current time, reading environment variables.

9) Does constructor return any value?

Ans: yes, that is current instance (You cannot use return type yet it returns a value).

10)What is object cloning?

The object cloning is used to create the exact copy of an object.

11) What is the basic difference between string and StringBuffer object?

String is an immutable object. StringBuffer is a mutable object.

12) What is the difference between StringBuffer and StringBuilder ?

StringBuffer is synchronized whereas StringBuilder is not synchronized.

13) How can we create immutable class in java ?

We can create immutable class as the String class by defining final class and

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()

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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)
    {
        String line = "java hadoop python oracle spark linux unix";

        StringTokenizer st = new StringTokenizer(line," ");

        while( st.hasMoreTokens() )
        {
            String str = st.nextToken();
            System.out.println(str);
        }
    }

    import java.util.StringTokenizer;

    public class Test {

        public static void main(String[] args)
        {
            String record = "1001:mnrao:manager:50505.50:admin:male:23";

            StringTokenizer st = new StringTokenizer(record,":");

            while( st.hasMoreTokens() )
            {
                String str = st.nextToken();
            }
        }
    }
}
```

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

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) Primitive type casting
- 2) Non-primitive type casting

1) Basic to Basic (primitive type casting)

Below all are Non-primitive type casting

2) Basic to Object

3) Object to Basic

4) Object to Object

5) String 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)

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Eg:

```
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

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

```
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 least 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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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:

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

Double d1= 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 is 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.nrity.mnrao.test;

public interface Picture {

    public void draw();

}
```

Child1 :

```
package com.nrity.mnrao.test;

public class Rectangle implements Picture{

    @Override
    public void draw() {
        // TODO Auto-generated method stub
        System.out.println("Ractangle draw");
    }
}
```

Child2:

```
package com.nrity.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 :

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
package com.nrity.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.nrity.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");
        }

    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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 {  
}  
}
```

Aftfter 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.nrity.mnrao.test;  
  
public class Example {  
    public void display(){  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        System.out.println("Exmaple display");
    }
}
```

Child class :

```
package com.nrity.mnrao.test;

public class Sample extends Example{

}
```

Main method for testing above one :

```
package com.nrity.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 shoud 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());
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

O/P :

Sample.

hashCode(); return address of object in form of unsigned integer.

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 available then it makes a call to Object class toString() method.

Object class toString() returns address (reference) of the object.

Overriding toString() method.

```
package com.nrity.mnrao.test;

package com.nrity.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 getUserName() {
    return uname;
}

public void setUserName(String uname) {
    this.uname = uname;
}

@Override
public String toString() {
    return "User [uid=" + uid + ", uname=" + uname + "]";
}

}

package com.nrity.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 assigning to parent
                  ( up casting)

Object obj = new Sample();
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
String str =(String) obj; //compiles but raises ClassCastException
```

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

//below is the valid coding for conversion.

```
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.nextLine();
```

```
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 converting 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];
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        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);
}
}
```

To covert Employee record, from **String format** to **Employee object** format , using POJO class

```
package com.durga.mnrao.oop;
public class Employee {
    private Integer empNum;
    private String empName;
    private String empJob;
    private Double empSalary;
    private String empDeptName;
    private String empGender;
    private Integer empAge;
    public Integer getEmpNum() {
        return empNum;
    }
    public void setEmpNum(Integer 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;
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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 Integer getEmpAge() {
    return empAge;
}

public void setEmpAge(Integer empAge) {
    this.empAge = empAge;
}

}

package com.durga.mnrao.oop;

public class Test {

    public static void main(String[] args) {

        String record = "1001:mnrao:manager:5050.50:admin:male:35";

        String [] fileds = record.split(":");
        Employee employee = new Employee();

        employee.setEmpNum(Integer.parseInt(fileds[0]));

        employee.setEmpName(fileds[1]);

        employee.setEmpJob(fileds[2]);

        employee.setEmpSalary(Double.parseDouble(fileds[3]));

        employee.setEmpDeptName(fileds[4]);

        employee.setEmpGender(fileds[5]);

        employee.setEmpAge(Integer.parseInt(fileds[6]));

        Integer empNum = employee.getEmpNum();
```

```
String empName = employee.getEmpName();
String empJob = employee.getEmpJob();
Double empSalary = employee.getEmpSalary();
String empDeptName = employee.getEmpDeptName();
String empGender = employee.getEmpGender();
Integer empAge = employee.getEmpAge();

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.

If 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 (java command)

Test → It is a .class file

hello java world ==> these are command line parameters

all 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 with three elements as below

args[0] ==> hello

args[1] ==> java

args[2] ==> world

D:\test>java Test **hello java**

array generates with two elements as below

args[0] ==> hello

args[1] ==> java

D:\test>java Test **hello**

array generates with only one element 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 ➔ no. of elements in args array = 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 parameters =" +n);
    }
}
```

compilation:

Windows:

D:\test> notepad Test.java

compilation:

D:\test> javac Test.java

execution:

D:\test> java Test hello java world

No.of parameters = 3

program to display command line parameters using for loop

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
```

parameter including space use double quote as below.

```
D:\test> java Test "hello java" world
```

```
parameters count = 2
```

```
hello java  
world
```

```
D:\test>java Test "Nageswar Rao" Mandru
```

```
parameters count = 2
```

```
Nageswar Rao  
Mandru
```

```
D:\test>java Test "Nageswar Rao Mandru"
```

```
parameters count = 1
```

```
Nageswar Rao Mandru
```

```
D:\test>java Test "hello java world" "mnrao abc xyz"
```

```
parameters count = 2
```

```
hello java world  
mnrao abc xyz
```

```
D:\test>java Test "hello java world" mnrao "abc xyz"
```

```
parameters count = 3
```

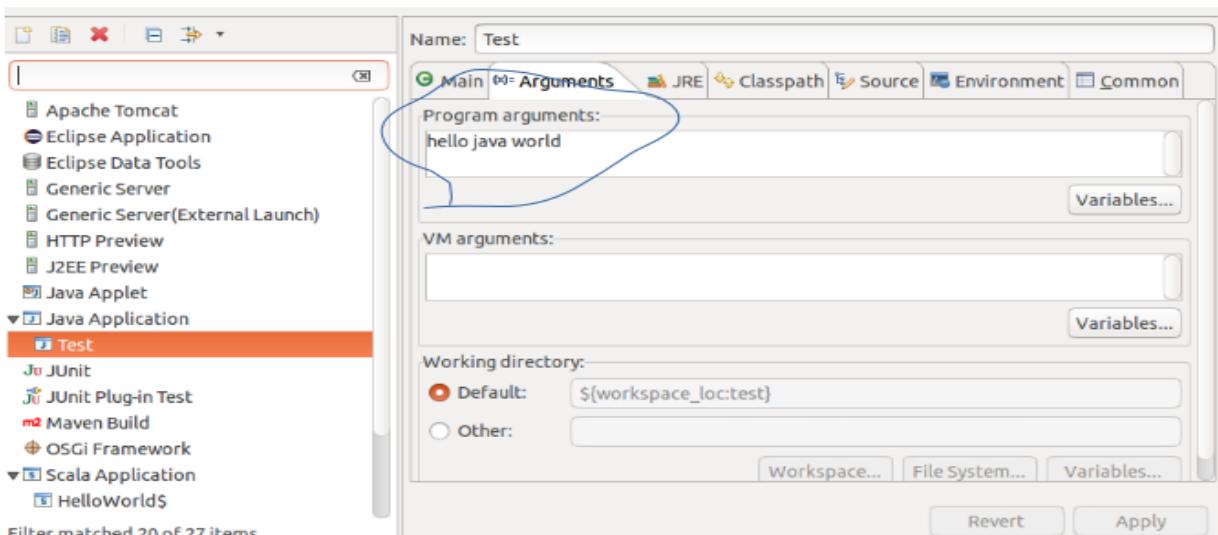
```
hello java world  
mnrao  
abc xyz
```

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 -->

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

then pass 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);
        }
    }
}
```

Exception Handling

before going to learn Exceptions, we should clear about, error, exception, bug and defect

Error : It is a compile time one. Such as syntactical errors.

Exception : It is a runtime one, it is an unkown instruction to processor at run time. when
Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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.

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:

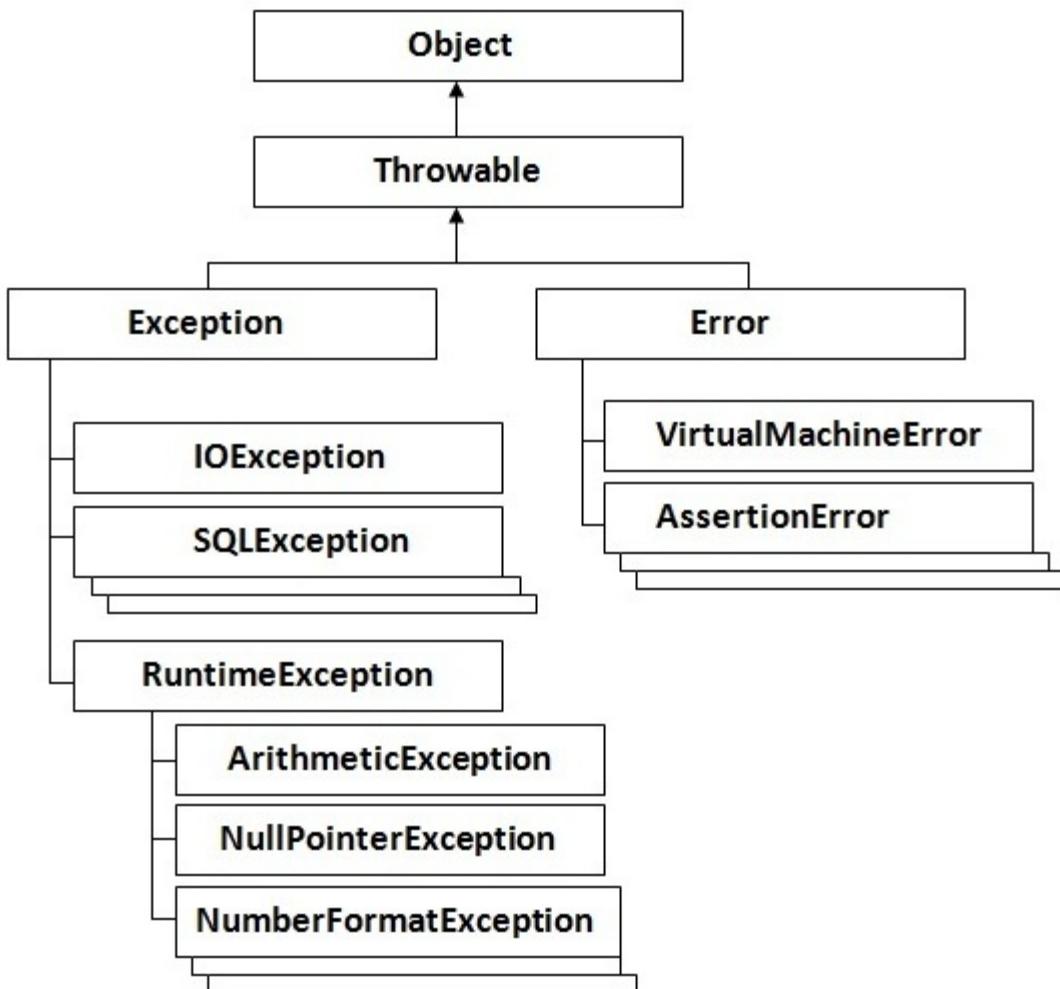
```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5;//exception occurs  
statement 6;  
statement 7;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
statement 8;  
statement 9;  
statement 10;
```

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.

Hierarchy of Java Exception classes



Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where **ArithmaticException** occurs

If we divide any number by zero, there occurs an **ArithmaticException**.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

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(ArithmetricException 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 above program with parameters and also 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");
    }
}
```

The diagram illustrates the execution flow of the Java program. It starts with the line `System.out.println("main start");`. Inside the `try` block, the line `System.out.println("try start");` is executed. Then, the variable `a` is assigned the value 10. The expression `int n = args.length;` is evaluated, and the value 1000 is stored in `n`. The expression `int b = a/n;` is then evaluated, resulting in the value 1000. The line `System.out.println("b = "+b);` is executed, printing the value 1000. After the `try` block, the line `System.out.println("try end");` is executed. In the `catch` block, the exception `new ArithmeticException("/ by zero")` is caught, and the message `divide by zero` is printed. Finally, the line `System.out.println("main end");` is executed.

}

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 → runConfiguration → 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(ArithmetricException e)  
        {  
            System.out.println("emp recordd missing, please submit");  
        }  
        catch(ArrayIndexOutOfBoundsException e)  
        {  
            System.out.println("Missing some fileds");  
        }  
        catch(NumberFormatException e)  
        {  
            System.out.println("non numeric value received ");  
        }  
    }  
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        }
    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]);
        }
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        System.out.println ("End of try");
    }
catch (ArithmaticException 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 (ArithmaticException 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;
            }
        }
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
        }
    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
```

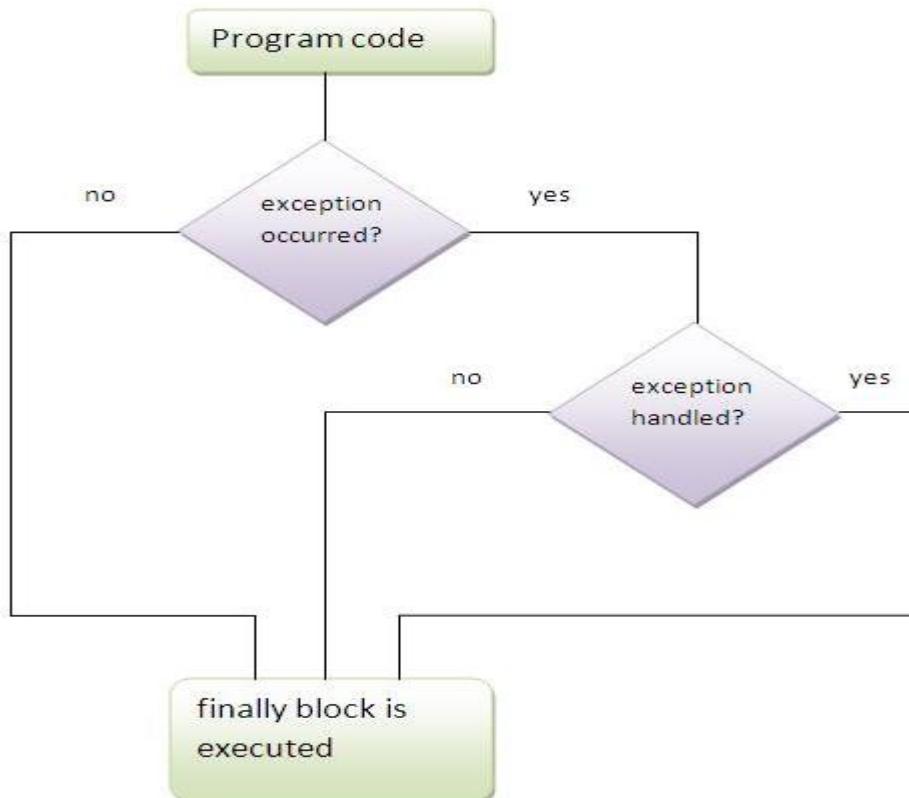
Java finally block

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

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 (ArithmaticException e)
        {
            System.out.println ("arithmetic ");
        }
        catch (NumberFormatException e)
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
    System.out.println ("Non numeric value");  
  
}  
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 scenarios

- 1) Normal termination
- 2) Abnormal termination

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 terminates 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 terminates abnormally.

```
public class Test {
    public static void main(String[] args)
    {
        System.out.println ("main start");
        try

```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
{  
    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 unchecked 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)
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
{  
    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 retruns null value.

Creating exception instance with text message.

```
throw new ArithmeticException("divide by zero ");
```

getMesage() 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){
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        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 Customized 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 defined 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 defined under which exception

checked / unchecked ?

Ans:

Checked ,

reason it is not from java.lang package

it is having its 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 using throws clause.

try –catch is used to get exception message, whereas 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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

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 handle by using try-catch.

Throws clause with multiple Exceptions

```
public static void main(String[] args) throws IOException, SampleException, 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);  
    }  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        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);
}

}
```

Rasing 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);
        }
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        catch(ArithmeticException e)
        {
            e.printStackTrace();
        }

    }
```

Rasing 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) {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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(ArithmaticException 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.	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 class can't be inherited, final method can't be overridden and final variable value can't be changed.		
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, its 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 is a block for exception handling, it executes for any conditions such as normally terminated or abnormally terminated.

Finalize: it is a method from Object. It will be called automatically, when object is cleared by the Garbage collector.

2. can we compile java program without main method.

Ans: Yes we can compile.

3. Is it compulsory for a Try Block to be followed by a Catch Block in Java for Exception handling?

Ans: Try block needs to be followed by either Catch block or Finally block or both. Any exception thrown from try block needs to be either caught in the catch block or else any specific tasks to be performed before code abortion are put in the Finally block.

4. Is there any way to skip Finally block of exception even if some exception occurs in the exception block?

Ans: If an exception is raised in Try block, control passes to catch block if it exists otherwise to finally block. Finally block is always executed when an exception occurs and the only way to avoid execution of any statements in Finally block is by aborting the code forcibly by writing following line of code at the end of try block:

`System.exit(0);`

Collection Frame Work

Collections :

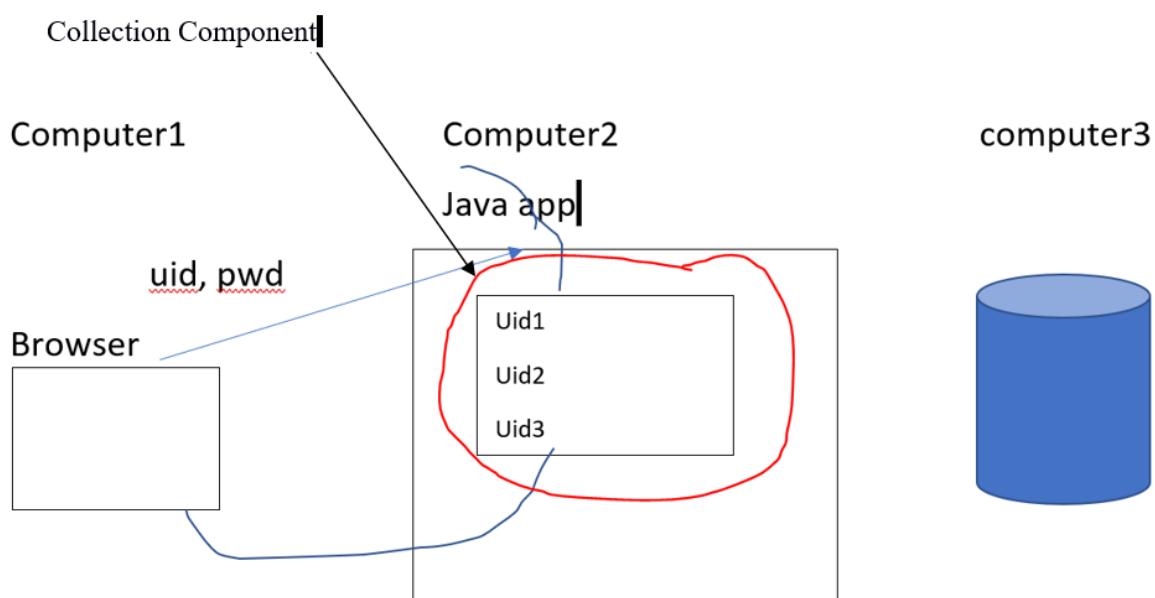
Collections is a frame work, it contains all readymade classes (pre-defined classes)

Collections components are used to maintain huze amount data(1 GB / 2 GB / 3 GB / 4 GB /10 GB) in the RAM memory

advantage is to improve the performance of an Application

Collections components are to maintain java objects

It is collection of java objects.



Collections in java is a framework that provides an architecture to store and manipulate the group of java objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue,

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

provides readymade architecture.

represents set of classes and interface.

Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

```
ArrayList al = new ArrayList(); //creating old non-generic arraylist
```

non-generic will take any object :

we can add any object to array list:

```
al.add(employee);  
al.add(student);  
al.add(customer);
```

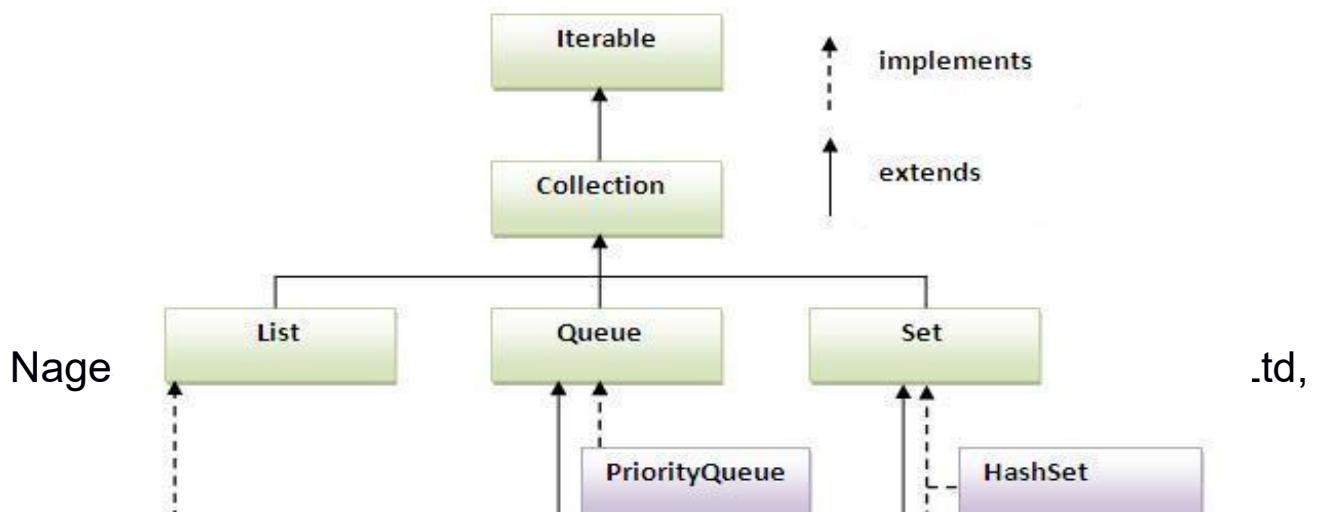
Let's see the new generic example of creating java collection.

```
ArrayList <String> al = new ArrayList <String>(); //creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives compile time error.

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework. The java.util package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1.	public boolean add(Object element)	is used to insert an element in this collection.
2.	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3.	public boolean remove(Object element)	is used to delete an element from this collection.
4.	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5.	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6.	public int size()	return the total number of elements in the collection.
7.	public void clear()	removes the total no of element from the collection.
8.	public boolean contains(Object element)	is used to search an element.
9.	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10.	public Iterator iterator()	returns an iterator.
11.	public Object [] toArray()	converts collection into array.
12.	public boolean isEmpty()	checks if collection is empty.
13.	public boolean equals(Object element)	matches two collection.
14.	public int hashCode()	returns the hashcode number for collection.
15.	public Object get(int index)	Return element at index

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

public boolean hasNext() it returns true if iterator has more elements.

public object next() it returns the element and moves the cursor pointer to the next element.

public void remove() it removes the last elements returned by the iterator. It is rarely used.

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.

Java ArrayList class can contain duplicate elements.

Java ArrayList class maintains insertion order.

Java ArrayList class is non synchronized.

Java ArrayList allows random access because array works at the index basis.

In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

ArrayList Vs Simple array

ArrayList :

ArrayList <Integer> list = new ArrayList < Integer >();

it contains multiple elments

max no of elements = no limit

it grows dynamically at runtime, it is called as dynamic array

Simple array:

int [] a = new int[5];

it contains multiple elments

max no of elements = 5

it a fixed size , it is called as static array

Example of Java ArrayList class with Integer elements

```
import java.util.ArrayList;  
  
public class Test {  
  
    public static void main(String[] args) {  
  
        ArrayList <Integer> list = new ArrayList <Integer> ();  
        System.out.println( "no of elements : " + list.size() );  
  
        list.add(10);  
        list.add(20);  
        list.add(30);  
        list.add(15);  
        list.add(20);  
        list.add(40);  
        list.add(50);  
  
        System.out.println("after adding ");  
        System.out.println( "no of elements : " + list.size() );  
        System.out.println("list of elements ");  
        System.out.println(list);  
  
    }  
}
```

O/p:
no of elements : 0
after adding
no of elements : 7
list of elements
[10, 20, 30, 15, 20, 40, 50]

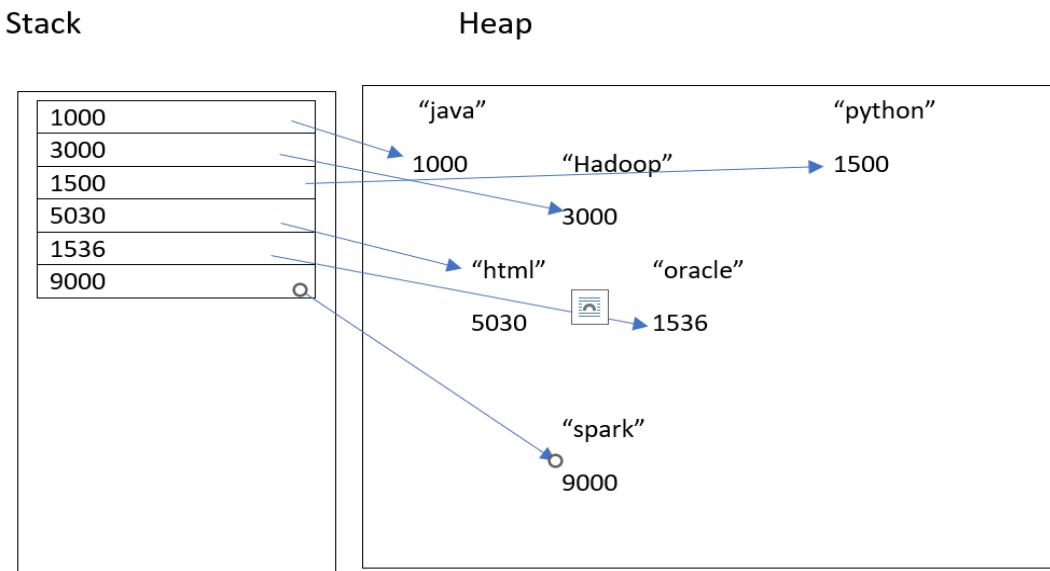
ArrayList with String elements

```
import java.util.ArrayList;  
  
public class Test {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public static void main(String[] args) {  
    ArrayList <String> list = new ArrayList <String> ();  
    System.out.println( "no of elements = "+ list.size() );  
    list.add("java");  
    list.add("hadoop");  
    list.add("python");  
    list.add("html");  
    list.add("oracle");  
    list.add("spark");  
    System.out.println("after adding ");  
    System.out.println( "no of elements = "+ list.size() );  
    System.out.println("list elements ");  
    System.out.println(list);  
}  
}  
  
o/p:  
no of elements = 0  
after adding  
no of elements = 6  
list elements  
[java, hadoop, python, html, oracle, spark]
```

Memory map for the above program :



get() → it returns elements at index :

```
String str = list.get(0);
String str = list.get(1);
String str = list.get(2);
```

Processing ArrayList elements in different ways

Recommended one is iterator

Since iterator using pointer concept, it can give more performance

```
import java.util.ArrayList;
import java.util.Iterator;

public class Test {

    public static void main(String[] args) {

        ArrayList <String> list = new ArrayList <String> ();

        list.add("java");
        list.add("hadoop");
        list.add("python");
        list.add("html");
        list.add("oracle");
        list.add("spark");

        System.out.println("list of elements ");
    }
}
```

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

System.out.println("using for loop ");

for(int i = 0; i<list.size(); i++)
{
    String str = list.get(i);
    System.out.println(str);
}

System.out.println("using foreach loop ");

for( String str : list )
{
    System.out.println(str);
}

System.out.println("using iterator ");

Iterator<String> iterator = list.iterator();

while( iterator.hasNext() )
{
    String str = iterator.next();

    System.out.println(str);
}

}
```

size(), isEmpty() and clear()

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList <String> list = new ArrayList <String>();

        System.out.println("no of elements = "+list.size());

        if( list.isEmpty() )
        {
            System.out.println("list is empty");
        }
        else
        {
            System.out.println("Not empty");
        }

        list.add("java");
        list.add("hadoop");
        list.add("python");
    }
}
```

```
        list.add("html");

        list.add("oracle");

        list.add("spark");

        System.out.println("after adding");

        System.out.println(list);

        System.out.println("no of elements = "+list.size());

        if( list.isEmpty() )
        {
            System.out.println("list is empty");
        }
        else
        {
            System.out.println("Not empty");
        }

        list.clear();

        System.out.println("after clearing");

        System.out.println(list);

        System.out.println("no of elements = "+list.size());

        if( list.isEmpty() )
        {
            System.out.println("list is empty");
        }
        else
        {
            System.out.println("Not empty");
        }

    }

}
```

Inserting elements at required index (at middle of the list)

add(int index , String str);

when elements are inserted into list, existing elements will be shifted to next position.

example for add() with index (insertion)

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList <String> list = new ArrayList <String>();
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
        list.add("java");
        list.add("hadoop");
        list.add("python");
        list.add("html");
        list.add("unix");
        list.add("spark");
        System.out.println(list);
        list.add(0, "oracle");
        System.out.println(list);
        list.add(2, "Linux");
        System.out.println(list);
    }
}
```

o/p:

```
[java, hadoop, python, html, unix, spark]
[oracle, java, hadoop, python, html, unix, spark]
[oracle, java, Linux, hadoop, python, html, unix, spark]
```

remove() → to remove elements from the list

remove() by index and by value

remove() by index → it removes elements at required index

remove() by value → it removes elements by value , if element is found then it removes and returns true, if not found returns false.

```
public class Test {
    public static void main(String[] args) {
        ArrayList <String> list = new ArrayList <String>();
        list.add("java");
        list.add("hadoop");
        list.add("python");
        list.add("html");
        list.add("unix");
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
list.add("spark");

System.out.println(list);

String str = list.remove(0); // remove by index

System.out.println("removed element = "+str);

System.out.println("after removing ");

System.out.println(list);

// remove by value

if( list.remove("python") )
{
    System.out.println(" python removed ");
}
else
{
    System.out.println("python not found");
}

System.out.println("after removing ");

System.out.println(list);
}

}
```

o/p:

```
[java, hadoop, python, html, unix, spark]
removed element = java
after removing
[hadoop, python, html, unix, spark]
python removed
after removing
[hadoop, html, unix, spark]
```

contains() → checking for existance of elements
if found retruns **true** , it not found **false**

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList <String> list = new ArrayList <String>();

        list.add("java");

        list.add("hadoop");

        list.add("python");

        list.add("html");
```

```
list.add("unix");
list.add("spark");
System.out.println(list);
if( list.contains("hadoop"))
{
    System.out.println("yes it contains hadoop");
}
else
{
    System.out.println("don't have element oralce");
}

if( list.contains("oracle"))
{
    System.out.println("yes it contains");
}
else
{
    System.out.println("don't have element");
}

}

o/p:
[java, hadoop, python, html, unix, spark]
yes it contains
don't have element
```

addAll() → to add one collection component to another collection component.

Program joining two lists into third into list.

```
import java.util.ArrayList;
public class Test {
    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<String>();
        list1.add("java");
        list1.add("html");
        list1.add("jdbc");
        list1.add("spring");
        list1.add("servlet");
        list1.add("jsp");
        ArrayList<String> list2 = new ArrayList<String>();
        list2.add("hadoop");
        list2.add("hdfs");
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
list2.add("flume");
list2.add("sqoop");
list2.add("hive");
list2.add("spark");

System.out.println(list1);

System.out.println(list2);

ArrayList<String> list3 = new ArrayList<String>();
list3.addAll(list1);
list3.addAll(list2);

System.out.println(list3);

}

}
```

}
o/p:

```
[java, html, jdbc, spring, servlet, jsp]
[hadoop, hdfs, flume, sqoop, hive, spark]
after joing list1 and list2
[java, html, jdbc, spring, servlet, jsp, hadoop, hdfs, flume, sqoop, hive, spark]
```

containsAll() → it returns true if a list contains all elements another list.

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<String>();

        list1.add("java");
        list1.add("html");
        list1.add("jdbc");
        list1.add("spring");
        list1.add("servlet");
        list1.add("jsp");

        ArrayList<String> list2 = new ArrayList<String>();
```

```
list2.add("hadoop");
list2.add("hdfs");
list2.add("flume");
list2.add("sqoop");
list2.add("hive");
list2.add("spark");

ArrayList<String> list3 = new ArrayList<String>();
list3.addAll(list1);
list3.addAll(list2);
System.out.println(list1);
System.out.println(list2);
System.out.println(list3);

if( list3.containsAll(list1) )
{
    System.out.println("yes list3 contains all elements of list1");
}
else
{
    System.out.println("no list3 don't have all elements of list1");
}

list3.remove("java");

System.out.println("after element from list3 ");

if( list3.containsAll(list1) )
{
    System.out.println("yes list3 contains all elements of list1");
}
else
{
    System.out.println("no list3 don't have all elements of list1");
}

}
```

}

O/p:

```
[java, html, jdbc, spring, servlet, jsp]
[hadoop, hdfs, flume, sqoop, hive, spark]
[java, html, jdbc, spring, servlet, jsp, hadoop, hdfs, flume, sqoop, hive, spark]
yes list3 contains all elements of list1
after element from list3
no list3 don't have all elements of list1
```

`removeAll()` → it removes all elements of one list from another list.
`equals()` → returns true if both lists are having same elements , other false

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<String>();

        list1.add("java");
        list1.add("html");
        list1.add("jdbc");
        list1.add("spring");
        list1.add("servlet");
        list1.add("jsp");

        ArrayList<String> list2 = new ArrayList<String>();

        list2.add("hadoop");
        list2.add("hdfs");
        list2.add("flume");
        list2.add("sqoop");
        list2.add("hive");
        list2.add("spark");

        ArrayList<String> list3 = new ArrayList<String>();

        list3.addAll(list1);
        list3.addAll(list2);

        System.out.println(list1);
        System.out.println(list2);
        System.out.println("after joining list1 and list2 ");
        System.out.println(list3);

        list3.removeAll(list2);
        System.out.println("after removing list2 from list3 ");
    }
}
```

```
System.out.println(list3);

if(list3.equals(list1))
{
    System.out.println("list1 and list3 same elements");
}
else
{
    System.out.println("list1 and list3 not same elements");
}

list3.remove("java");

System.out.println("after removing one element from list3 ");

if(list3.equals(list1))
{
    System.out.println("list1 and list3 same elements");
}
else
{
    System.out.println("list1 and list3 not same elements");
}

}

o/p:
```

```
[java, html, jdbc, spring, servlet, jsp]
[hadoop, hdfs, flume, sqoop, hive, spark]
after joing list1 and list2
[java, html, jdbc, spring, servlet, jsp, hadoop, hdfs, flume, sqoop, hive, spark]
after removing list2 from list3
[java, html, jdbc, spring, servlet, jsp]
list1 and list3 same elements
after removing one element from list3
list1 and list3 not same elements
```

retainAll() → it will retain all elements of required list and removes remaining all elements

```
import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<String>();

        list1.add("java");

        list1.add("html");

        list1.add("jdbc");

        list1.add("spring");
```

```
list1.add("servlet");
list1.add("jsp");
ArrayList<String> list2 = new ArrayList<String>();
list2.add("hadoop");
list2.add("hdfs");
list2.add("flume");
list2.add("sqoop");
list2.add("hive");
list2.add("spark");

ArrayList<String> list3 = new ArrayList<String>();
list3.addAll(list1);
list3.addAll(list2);

System.out.println(list1);
System.out.println(list2);
System.out.println("after joining list1 and list2 into list3");
System.out.println(list3);

list3.add("unix");
list3.add("linux");
list3.add("solaris");
System.out.println("after adding some elements to list3 ");

System.out.println(list3);

list3.retainAll(list2);
System.out.println("after retain all of list2 in list3 ");
System.out.println(list3);

if(list3.equals(list2))
{
    System.out.println("list2 and list3 having same elements");
}
else
{
    System.out.println("list2 and list3 don't have same elements");
}
```

```
}
```

o/p:

```
[java, html, jdbc, spring, servlet, jsp]  
[hadoop, hdfs, flume, sqoop, hive, spark]
```

after joining list1 and list2 into list3

```
[java, html, jdbc, spring, servlet, jsp, hadoop, hdfs, flume, sqoop, hive, spark]
```

after adding some elements to list3

```
[java, html, jdbc, spring, servlet, jsp, hadoop, hdfs, flume, sqoop, hive, spark, unix,  
linux, solaris]
```

after retaining all of list2 in list3

```
[hadoop, hdfs, flume, sqoop, hive, spark]
```

list2 and list3 having same elements

public Object [] toArray() :

this method is to convert Collection components (list) dynamic array to static array.

```
import java.util.ArrayList;  
  
public class Test {  
  
    public static void main(String[] args) {  
  
        ArrayList<String> list = new ArrayList<String>();  
  
        list.add("java");  
  
        list.add("html");  
  
        list.add("jdbc");  
  
        list.add("spring");  
  
        list.add("servlet");  
  
        list.add("jsp");  
  
        Object [] objects = list.toArray();  
  
        for( Object object : objects)  
        {  
            String str = (String) object;  
            System.out.println(str);  
        }  
    }  
}
```

}

User-defined class objects in Java ArrayList

```
package com.durga.mnrao.xyz;

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;
}

}

package com.durga.mnrao.xyz;

import java.util.ArrayList;
import java.util.Iterator;

public class Test {

    public static void main(String[] args) {
        String [] records = {
            "1001,ajay,manager,account,45000,male,38",
            "1002,aiswrya,clerk,admin,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,admin,23000,male,30",
            "1007,sunil,accountant,sales,13000,male,29",
            "1008,satvik,director,purchase,80000,male,45"
        };

        ArrayList<Employee> list = new ArrayList<Employee>();

        for(String record: records)
        {
            String[] fields = record.split(",");
            Employee employee = new Employee();
            employee.setEmpNum(Integer.parseInt(fields[0]));
            employee.setEmpName(fields[1]);
            employee.setEmpJob(fields[2]);
            employee.setEmpDeptName(fields[3]);
        }
    }
}
```

```
employee.setEmpSalary(Double.parseDouble(fields[4]));

employee.setEmpGender(fields[5]);

employee.setEmpAge(Integer.parseInt(fields[6]));

list.add(employee);

}

Iterator<Employee> iterator = list.iterator();

while(iterator.hasNext())
{
    Employee emp = iterator.next();

    int empNum = emp.getEmpNum();

    String empName = emp.getEmpName();

    String empJob = emp.getEmpJob();

    double empSalary = emp.getEmpSalary();

    String empDeptName = emp.getEmpDeptName();

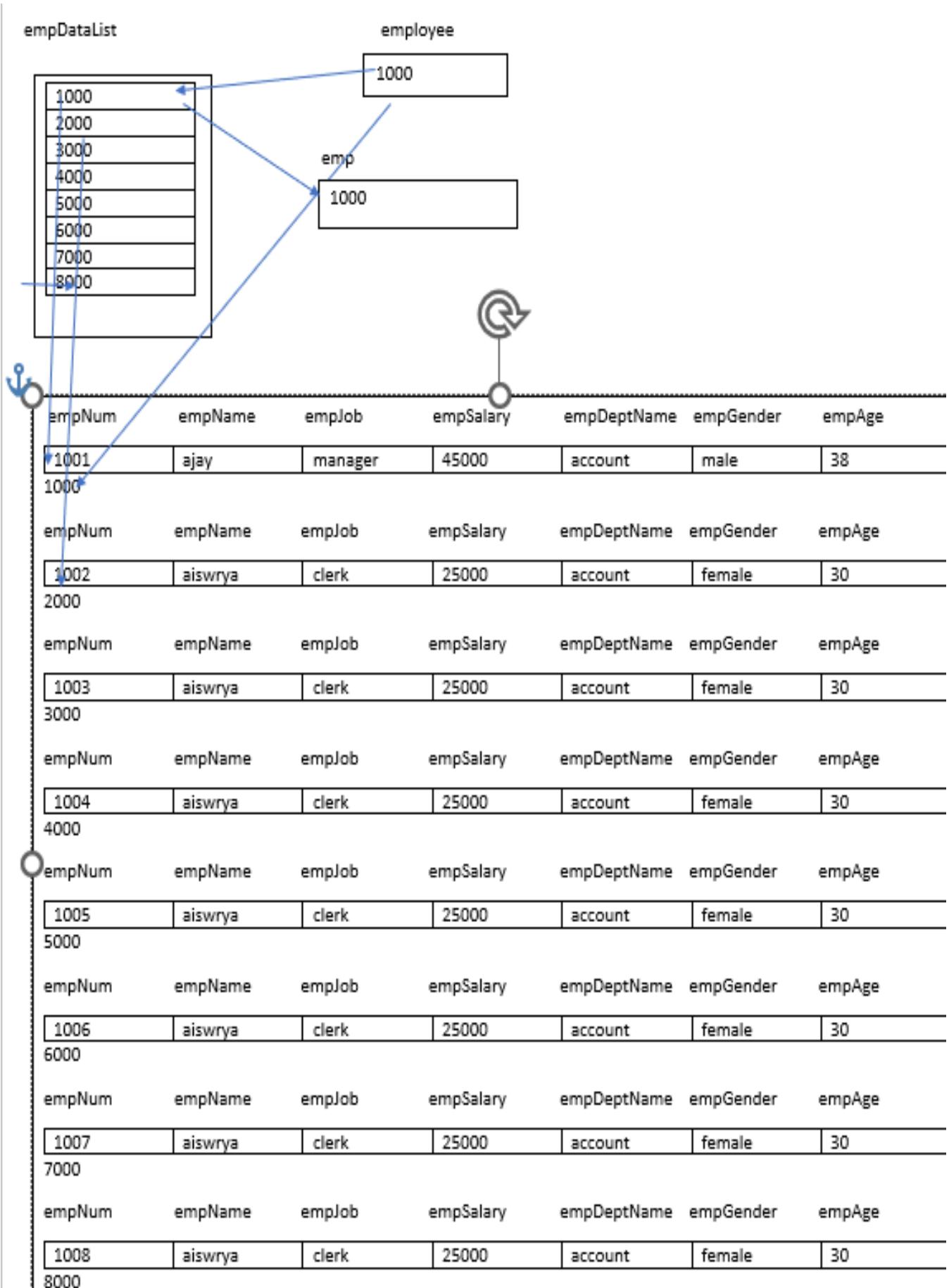
    String empGender = emp.getEmpGender();

    int empAge = emp.getEmpAge();

System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empSalary+"\t"+empDeptName+"\t"+
empGender+"\t"+empAge);
}

}
```

Memory map for the above program :

**Java LinkedList class**

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.

Java LinkedList class can contain duplicate elements.

Java LinkedList class maintains insertion order.

Java LinkedList class is non synchronized.

In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.

Java LinkedList class can be used as list, stack or queue.

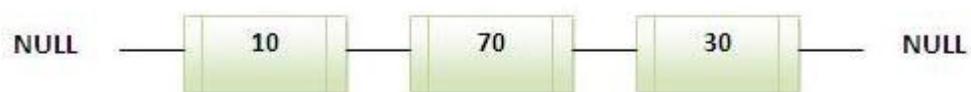


fig- doubly linked list

Java LinkedList Example

```
import java.util.*;
public class Test
{
    public static void main(String [] args)
    {
        LinkedList<String> al=new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        Iterator<String> itr=al.iterator();
        while(itr.hasNext())
        {
            System.out.println (itr.next());
        }
    }
}
```

Output:

Ravi
Vijay
Ravi
Ajay

Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses dynamic array to store the elements	LinkedList internally uses doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the key array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4) ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both.

```
import java.util.LinkedList;
import java.util.List;
import java.util.ArrayList;

public class Test
{
    public static void main(String [] args)
    {
        List<String> al = new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        List<String> al2 = new LinkedList<String>();
        al2.add("James");
        al2.add("Serena");
        al2.add("Swati");
        al2.add("Junaid");

        System.out.println ("arraylist: " + al);
        System.out.println ("linkedlist: " + al2);
    }
}
```

Output:

arraylist: [Ravi,Vijay,Ravi,Ajay]

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

linkedlist: [James,Serena,Swati,Junaid]

Vector:

The **java.util.Vector** class implements a growable array of objects. Similar to an Array, it contains components that can be accessed using an integer index.

The size of a Vector can grow or shrink as needed to accommodate adding and removing items.

Vector is synchronized.

. Class declaration

Following is the declaration for **java.util.Vector** class:

```
public class Vector<E>
    extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, Serializable
```

Constructor & Description

Vector()

This constructor is used to create an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.

Vector(Collection<? extends E> c)

This constructor is used to create a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Vector(int initialCapacity)

This constructor is used to create an empty vector with the specified initial capacity and with its capacity increment equal to zero.

Vector(int initialCapacity, int capacityIncrement)

This constructor is used to create an empty vector with the specified initial capacity and capacity increment.

Method & Description

Method & Description

boolean add(E e)

This method appends the specified element to the end of this Vector.

void add(int index, E element)

This method inserts the specified element at the specified position in this Vector

boolean addAll(Collection<? extends E> c)

This method appends all of the elements in the specified Collection to the end of this Vector.

boolean addAll(int index, Collection<? extends E> c)

This method inserts all of the elements in the specified Collection into this Vector at the specified position.

void addElement(E obj)

This method adds the specified component to the end of this vector, increasing its size by one.

int capacity()

This method returns the current capacity of this vector.

void clear()

This method removes all of the elements from this vector.

Clone clone()

This method returns a clone of this vector.

boolean contains(Object o)

This method returns true if this vector contains the specified element.

boolean containsAll(Collection<?> c)

This method returns true if this Vector contains all of the elements in the specified Collection.

void copyInto(Object[] anArray)

This method copies the components of this vector into the specified array.

E elementAt(int index)

This method returns the component at the specified index.

Enumeration<E> elements()

This method returns an enumeration of the components of this vector.

void ensureCapacity(int minCapacity)

This method increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

boolean equals(Object o)

This method compares the specified Object with this Vector for equality.

E firstElement()

This method returns the first component (the item at index 0) of this vector.

E get(int index)

This method returns the element at the specified position in this Vector.

int hashCode()

This method returns the hash code value for this Vector.

int indexOf(Object o)

This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

int indexOf(Object o, int index)

This method returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.

void insertElementAt(E obj, int index)

This method inserts the specified object as a component in this vector at the specified index.

boolean isEmpty()

This method tests if this vector has no components.

E lastElement()

This method returns the last component of the vector.

int lastIndexOf(Object o)

This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

int lastIndexOf(Object o, int index)

This method returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.

E remove(int index)

This method removes the element at the specified position in this Vector.

boolean remove(Object o)

This method removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.

boolean removeAll(Collection<?> c)

This method removes from this Vector all of its elements that are contained in the specified Collection.

void removeAllElements()

This method removes all components from this vector and sets its size to zero.

boolean removeElement(Object obj)

This method removes the first occurrence of the argument from this vector.

void removeElementAt(int index)

This method deletes the component at the specified index.

boolean retainAll(Collection<?> c)

This method retains only the elements in this Vector that are contained in the specified Collection.

E set(int index, E element)

This method replaces the element at the specified position in this Vector with the specified element.

void setElementAt(E obj, int index)

This method sets the component at the specified index of this vector to be the specified object.

void setSize(int newSize)

This method sets the size of this vector.

int size()

This method returns the number of components in this vector.

List <E> subList(int fromIndex, int toIndex)

This method returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.

object[] toArray()

This method returns an array containing all of the elements in this Vector in the correct order.

<T> T[] toArray(T[] a)

This method returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.

String toString()

This method returns a string representation of this Vector, containing the String representation of each element.

void trimToSize()

This method trims the capacity of this vector to be the vector's current size.

Program: Basic Vector Operations.

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

import java.util.Vector;

public class BasicVectorOperations {

    public static void main(String a[]) {

        Vector<String> vct = new Vector<String>();
        // adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");

        System.out.println(vct);

        // adding element at specified index
        vct.add(2, "Random");
    }
}
```

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

// getting elements by index
System.out.println("Element at index 3 is: " + vct.get(3));

// getting first element
System.out.println("The first element of this vector is: " + vct.firstElement());
// getting last element
System.out.println("The last element of this vector is: " + vct.lastElement());
// how to check vector is empty or not
System.out.println("Is this vector empty? " + vct.isEmpty());

}

}
```

Output:

```
[First, Second, Third]
[First, Second, Random, Third]
Element at index 3 is: Third
The first element of this vector is: First
The last element of this vector is: Third
Is this vector empty? false
```

Program: How to read all elements in vector by using iterator?

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

import java.util.Iterator;
import java.util.Vector;

public class VectorIterator {

    public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
        // adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");

        Iterator<String> itr = vct.iterator();

        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

Output:

```
First
Second
Third
Random
```

Program: How to read all elements in vector by using Enumeration?

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

import java.util.Enumeration;
import java.util.Vector;

public class VectorEnnumaratio {

    public static void main(String a[]) {

        Vector<String> vct = new Vector<String>();
        // adding elements to the end

        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");

        Enumeration<String> enm = vct.elements();

        while (enm.hasMoreElements()) {
            System.out.println(enm.nextElement());
        }
    }
}
```

Program: How to copy or clone a vector?

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

import java.util.Vector;

public class MyVectorClone {
    public static void main(String a[]){
        Vector<String> vct = new Vector<String>();
        //adding elements to the end
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
vct.add("First");
vct.add("Second");
vct.add("Third");
vct.add("Random");

System.out.println("Actual vector:" + vct);

Vector<String> copy = (Vector<String>) vct.clone();

System.out.println("Cloned vector:" + copy);
}

}
```

Program: How to add all elements of a list to vector?

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

import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

public class MyVectorNewCollection {

    public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
        // adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");
        System.out.println("Actual vector:" + vct);

        List<String> list = new ArrayList<String>();

        list.add("one");
        list.add("two");
        vct.addAll(list);
        System.out.println("After Copy: " + vct);
    }
}
```

Program: How to delete all elements from my vector?

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

import java.util.Vector;

public class ClearMyVector {

    public static void main(String a[]){
        Vector<String> vct = new Vector<String>();
```

```
//adding elements to the end
vct.add("First");
vct.add("Second");
vct.add("Third");
vct.add("Random");
System.out.println("Actual vector:" + vct);
vct.clear();
System.out.println("After clear vector:" + vct);
}
}
```

Program: How to find does vector contains all list elements or not?

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

import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
public class MyElementCheck {

    public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");

        List<String> list = new ArrayList<String>();
        list.add("Second");
        list.add("Random");

        System.out.println("Does vector contains all list elements?: " +
vct.containsAll(list));
        list.add("one");
        System.out.println("Does vector contains all list elements?: " +
vct.containsAll(list));
    }
}
```

Program: How to copy vector to array?

```
package com.nrit.mnrao.test;
import java.util.Vector;
public class MyVectorArrayCopy {

    public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");
```

```
System.out.println("Actual vector:" + vct);

String []copyArr = new String[vct.size()];

vct.copyInto(copyArr);
System.out.println("Copied Array content:");

for (String str : copyArr) {
    System.out.println(str);
}

}
```

Program: How to get sub list from vector?

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

import java.util.List;
import java.util.Vector;

public class MyVectorSubRange {

    public static void main(String a[]) {
        Vector<String> vct = new Vector<String>();
        // adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Random");
        vct.add("Click");
        System.out.println("Actual vector:" + vct);
        List<String> list = vct.subList(2, 4);
        System.out.println("Sub List: " + list);
    }
}
```

Difference between ArrayList and Vector:

ArrayList and Vector both implements List interface and maintains insertion order.

S.No.	ArrayList	Vector
1.	ArrayList is not synchronized.	Vector is synchronized.
2.	ArrayList increments 50% of current array size if number of element exceeds from its capacity.	Vector increments 100% means doubles the array size if total number of element exceeds than its capacity.
3.	ArrayList is not a legacy class, it is introduced in JDK 1.2.	Vector is a legacy class.
4.	ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized i.e. in multithreading environment, it will hold

		the other threads in runnable or non-runnable state until current thread releases the lock of object.
5.	ArrayList uses Iterator interface to traverse the elements.	Vector uses Enumeration interface to traverse the elements. But it can use Iterator also.

Stack Class

Stack is a subclass of **Vector** that implements a standard last-in, first-out stack. **Stack** only defines the default constructor, which creates an empty stack. **Stack** includes all the methods defined by **Vector**, and adds several of its own.

To put an object on the top of the stack, call **push()**. To remove and return the top element, call **pop()**. An **EmptyStackException** is thrown if you call **pop()** when the invoking stack is empty. You can use **peek()** to return, but not remove, the top object.

The **empty()** method returns **true** if nothing is on the stack. The **search()** method determines whether an object exists on the stack, and returns the number of pops that are required to bring it to the top of the stack. Here is an example that creates a stack, pushes several **Integer** objects onto it, and then pops them off again:

The **java.util.Stack** class represents a last-in-first-out (LIFO) stack of objects.

- When a stack is first created, it contains no items.
- In this class, the last element inserted is accessed first.

Class declaration

Following is the declaration for **java.util.Stack** class:

```
public class Stack<E>
    extends Vector<E>
```

Constructor & Description

Stack()

This constructor creates an empty stack.

Method & description

boolean empty()

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

This method tests if this stack is empty.

E peek()

This method looks at the object at the top of this stack without removing it from the stack.

E pop()

This method removes the object at the top of this stack and returns that object as the value of this function. If stack is empty it throws “**EmptyStackException**”

E push(E item)

This method pushes an item onto the top of this stack.

int search(Object o)

This method returns the 1-based position where an object is on this stack.

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

import java.util.Stack;

public class Test {

    public static void main(String[] args) {

        Stack <String> bookRack = new Stack<String>();

        bookRack.push("HADOOP");
        bookRack.push("JAVA");
        bookRack.push("ORACLE");
        bookRack.push("C");
        bookRack.push("LINUX");

        System.out.println("books in the Rack "+bookRack);

        System.out.println("Top book : "+bookRack.peek());

        String book = bookRack.pop();

        System.out.println("Removed Book "+book);

        book = bookRack.pop();

        System.out.println("Removed Book "+book);
    }
}
```

```
        System.out.println("Current books in the Rack "+bookRack);
    }
}
```

Program is to remove stack elements one by one.

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

import java.util.Stack;

public class Test {

    public static void main(String[] args) {

        Stack<String> bookRack = new Stack<String>();

        bookRack.push("HADOOP");
        bookRack.push("JAVA");
        bookRack.push("ORACLE");
        bookRack.push("C");
        bookRack.push("LINUX");

        while( ! bookRack.isEmpty() )
        {
            String book = bookRack.pop();
            System.out.println(book);
        }
    }
}
```

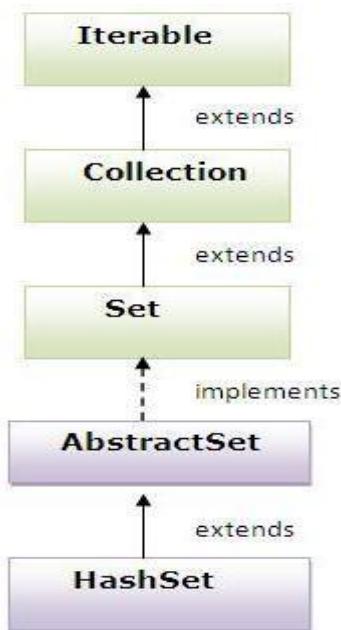
Java HashSet class

uses hashtable to store the elements. It extends AbstractSet class and implements Set interface. contains unique elements only.

Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class:



HashSet operations are same as List , it will not allow duplicate elements

Example of HashSet class:

```
import java.util.HashSet;

public class Test {

    public static void main(String[] args) {

        HashSet <String> hset = new HashSet <String>();

        hset.add("java");
        hset.add("hadoop");
        hset.add("java");
    }
}
```

```
    hset.add("linux");
    hset.add("java");
    hset.add("spark");
    System.out.println(hset);

}

}
```

o/p:

```
[java, spark, linux, hadoop]
```

TreeSet class:

TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

Contains unique elements only like HashSet.
Access and retrieval times are quiet fast.
Maintains ascending order.

```
import java.util.HashSet;
import java.util.TreeSet;

public class Test {

    public static void main(String[] args) {

        TreeSet <String> tset = new TreeSet <String>();
        tset.add("java");
        tset.add("hadoop");
        tset.add("java");
        tset.add("linux");
        tset.add("java");
        tset.add("spark");
        System.out.println(tset);

    }
}
```

o/p:

Ascending order

[hadoop, java, linux, spark]

Java Map Interface

A map contains values based on the key i.e. key and value pair.
Each pair is known as an entry.
Map contains only unique elements.

Commonly used methods of Map interface:

public Object put(object key, Object value): is used to insert an entry in this map.

public void putAll(Map map): is used to insert the specified map in this map.

public Object remove(object key): is used to delete an entry for the specified key.

public Object get(Object key): is used to return the value for the specified key.

public boolean containsKey(Object key): is used to search the specified key from this map.

public boolean containsValue(Object value): is used to search the specified value from this map.

public Set keySet(): returns the Set view containing all the keys.

public Set entrySet(): returns the Set view containing all the keys and values.

Java HashMap class

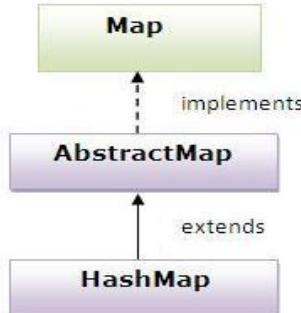
A HashMap contains values based on the key. It implements the Map interface and extends **AbstractMap class**.

It contains only unique elements.
It may have one null key and multiple null values.
It maintains no order.

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains entry(key and value).

Hierarchy of HashMap class:



Non generic Example for HashMap (any key and any value)

```
import java.util.HashMap;
import java.util.Stack;

public class Test {

    public static void main(String[] args) {

        HashMap hmap = new HashMap();

        hmap.put(1, "java");
        hmap.put("hello", 1);
        hmap.put(2, new Employee());

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

Generic HashMap Example:

```
import java.util.HashMap;

public class Test {

    public static void main(String[] args) {

        HashMap<Integer, String> hmap = new HashMap<Integer, String>();

        hmap.put(1, "java");

        hmap.put(2, "hadoop");

        hmap.put(3, "unix");
    }
}
```

```
        hmap.put(4, "spark");

        hmap.put(5, "html");

        hmap.put(6, "python");

        System.out.println(hmap);

    }

}
```

o/p:

```
{1=java, 2=hadoop, 3=unix, 4=spark, 5=html, 6=python}
```

get() → it takes key and return value, if key not found returns null value

```
import java.util.HashMap;

public class Test {

    public static void main(String[] args) {

        HashMap<Integer, String> hmap = new HashMap<Integer, String>();

        hmap.put(1, "java");

        hmap.put(2, "hadoop");

        hmap.put(3, "unix");

        hmap.put(4, "spark");

        hmap.put(5, "html");

        hmap.put(6, "python");

        System.out.println(hmap);

        String value1 = hmap.get(1);

        String value2 = hmap.get(2);

        String value3 = hmap.get(3);

        String value4 = hmap.get(4);

        String value5 = hmap.get(5);

        String value6 = hmap.get(6);

        System.out.println(value1);

        System.out.println(value2);

        System.out.println(value3);
```

```
System.out.println(value4);
System.out.println(value5);
System.out.println(value6);

}

}
```

Eg:

```
import java.util.HashMap;

public class Test {

    public static void main(String[] args) {

        HashMap<String, String> hmap = new HashMap<String, String>();
        hmap.put("j", "java");
        hmap.put("h", "hadoop");
        hmap.put("u", "unix");
        hmap.put("s", "spark");
        hmap.put("o", "oracle");
        hmap.put("p", "python");
        System.out.println(hmap);
    }
}
```

Hashtable :

```
import java.util.Hashtable;

public class Sample {

    public static void main(String[] args) {

        Hashtable<String, String> htable = new Hashtable<String, String>();
        htable.put("j", "java");
        htable.put("h", "hadoop");
        htable.put("u", "unix");
    }
}
```

```
        htable.put("s", "spark");
        htable.put("o", "oralce");
        htable.put("p", "python");
        System.out.println(htable);
    }
}
```

Using keyset iterator :

HashMap:

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class Test {

    public static void main(String[] args)  {

        HashMap<String, String> hmap = new HashMap<String, String>();
        hmap.put("j", "java");
        hmap.put("h", "hadoop");
        hmap.put("p", "python");
        hmap.put("o", "oracle");
        hmap.put("u", "unix");
        hmap.put("l", "linux");
        System.out.println(hmap);

        Set<String> keySet = hmap.keySet();

        Iterator<String> iterator = keySet.iterator();

        while( iterator.hasNext() )
        {
            String key = iterator.next();
            String value = hmap.get(key);
            System.out.println(key+"\t"+value);
        }
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

}

Hashtable:

```
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;

public class Sample {

    public static void main(String[] args) {

        Hashtable<String, String> htable = new Hashtable<String, String>();
        htable.put("j", "java");
        htable.put("h", "hadoop");
        htable.put("p", "python");
        htable.put("o", "oracle");
        htable.put("u", "unix");
        htable.put("l", "linux");
        System.out.println(htable);

        Set<String> keySet = htable.keySet();
        Iterator<String> iterator = keySet.iterator();

        while( iterator.hasNext() )
        {
            String key = iterator.next();
            String value = htable.get(key);
            System.out.println(key+"\t"+value);
        }
    }
}
```

Entrset

HashMap:

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
package com.durga.mnrao.xyz;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.Set;

public class Test {

    public static void main(String[] args) {

        HashMap<String, String> hmap = new HashMap<String, String>();

        hmap.put("j", "java");
        hmap.put("h", "hadoop");
        hmap.put("p", "python");
        hmap.put("o", "oracle");
        hmap.put("u", "unix");
        hmap.put("l", "linux");

        System.out.println(hmap);

        Set<Entry<String, String>> entrySet = hmap.entrySet();
        Iterator<Entry<String, String>> iterator = entrySet.iterator();

        while( iterator.hasNext() )
        {
            Entry<String, String> entry = iterator.next();
            String key = entry.getKey();
            String value = entry.getValue();
            System.out.println(key+"\t"+value);
        }

    }
}
```

Hashtable

```
package com.durga.mnrao.xyz;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.Map.Entry;
```

```
public class Sample {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public static void main(String[] args) {  
  
    Hashtable<String, String> htable = new Hashtable<String, String>();  
  
    htable.put("j", "java");  
  
    htable.put("h", "hadoop");  
  
    htable.put("p", "python");  
  
    htable.put("o", "oracle");  
  
    htable.put("u", "unix");  
  
    htable.put("l", "linux");  
  
    System.out.println(htable);  
  
  
    Set<Entry<String, String>> entrySet = htable.entrySet();  
  
    Iterator<Entry<String, String>> iterator = entrySet.iterator();  
  
    while( iterator.hasNext() )  
    {  
        Entry<String, String> entry = iterator.next();  
  
        String key = entry.getKey();  
  
        String value = entry.getValue();  
  
        System.out.println(key+"\t"+value);  
    }  
  
}  
  
}
```

Replacing element value

```
package com.nrit.mnrao.test;  
  
import java.util.HashMap;  
public class Test {  
  
    public static void main(String[] args) {  
  
        HashMap<Integer, String> hmap = new HashMap<Integer, String>();  
  
        hmap.put(1, "java");  
        hmap.put(2, "hadoop");  
        hmap.put(3, "oracle");  
        hmap.put(4, "unix");  
        hmap.put(5, "linux");  
    }  
}
```

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

if(hmap.containsKey(4) )
{
    if(hmap.replace(4, "unix", "linux"))
    {
        System.out.println("successfully replaced");
    }
    else
    {
        System.out.println("failed to replace");
    }
}

System.out.println(hmap);

}
```

Database properties

```
import java.util.HashMap;

public class Test
{
    public static void main(String[] args)
    {
        HashMap<String, String> DataBaseproperties = new HashMap<String, String>();

        DataBaseproperties.put("user", "demo");
        DataBaseproperties.put("path", "/home/demo");
        DataBaseproperties.put("host", "160.10.20.3");
        DataBaseproperties.put("dbname", "student");
        DataBaseproperties.put("uid", "root");
        DataBaseproperties.put("pwd", "admin123");

        String uidValue = DataBaseproperties.get("uid");
        String pwdValue = DataBaseproperties.get("pwd");
        String pathValue = DataBaseproperties.get("path");
        String userValue = DataBaseproperties.get("user");

        System.out.println (uidValue);
        System.out.println (pwdValue);
        System.out.println (pathValue);
        System.out.println (userValue);

        System.out.println (DataBaseproperties.toString());
    }
}
```

Hashtable class:

Hashtable class implements a Map interface, which maps keys to values. It inherits Dictionary class and implements the Map interface.

A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashCode() method. A Hashtable contains values based on the key.

It contains only unique elements.

It may have not have any null key or value.

It is synchronized

Difference between HashMap and Hashtable:

S.No	HashMap	Hashtable
1.	HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized. It is thread-safe and can be shared with many threads.
2.	HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value.
3.	HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
4.	HashMap is fast.	Hashtable is slow.
5.	We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6.	HashMap is traversed by Iterator.	Hashtable is traversed by Enumerator and Iterator.
7.	Iterator in HashMap is fail-fast.	Enumerator in Hashtable is not fail-fast.
8.	HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

HashMap	HashTable
Key and value	Key and value
Key is unique	Key is unique
Value can be duplicate	Value can be duplicate
Null key allowed but only one null allowed	Key should not be null
Any no of null values allowed	Value also should not be null
Non Synchronized	Synchronized
Not a thread safe	Thread safe

Collections class:

collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.

Java Collection class supports the polymorphic algorithms that operate on collections.

Java Collection class throws a NullPointerException if the collections or class objects provided to them are null.

Collections class declaration

```
public class Collections extends Object
```

Methods of Java Collections class

Method	Description
static <T> boolean addAll(Collection<? super T> c, T... elements)	It is used to add all of the specified elements to the specified collection.
static <T> Queue<T> asLifoQueue(Deque<T> deque)	It is used to return a view of a Deque as a Last-In-First-Out (LIFO) Queue.
static <T> int binarySearch(List<? extends T> list, T key, Comparator<? super T< c>)	It is used to search the specified list for the specified object using the binary search algorithm.
static <E> List<E> checkedList(List<E> list, Class<E> type)	It is used to return a dynamically typesafe view of the specified list.
static <E> Set<E> checkedSet(Set<E> s, Class<E> type)	It is used to return a dynamically typesafe view of the specified set.
static<E> SortedSet<E>checkedSortedSet(SortedSet<E> s, Class<E> type)	It is used to return a dynamically typesafe view of the specified sorted set
static void reverse(List<?> list)	It is used to reverse the order of the elements in the specified list.
static <T> T max(Collection<? extends T> coll, Comparator<? super T> comp)	It is used to return the maximum element of the given collection, according to the order induced by the specified comparator.
static <T extends Object & Comparable<? super T>>T min(Collection<? extends T> coll)	It is used to return the minimum element of the given collection, according to the natural ordering of its elements.
static boolean replaceAll(List list, T oldVal, T newVal)	It is used to replace all occurrences of one specified value in a list with another.

Collections Example:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Test
{
    public static void main(String a[])
    {
        List<String> list = new ArrayList<String>();
        list.add("C");
        list.add("Core Java");
        list.add("Advance Java");
        System.out.println ("Initial collection value:" + list);
        Collections.addAll(list, "Servlet", "JSP");
        System.out.println ("After adding elements collection value:" + list);
        String[] strArr = { "C#", ".Net" };
        Collections.addAll(list, strArr);
        System.out.println ("After adding array collection value:" + list);
    }
}

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Test
{
    public static void main(String a[])
    {
        List<Integer> list = new ArrayList<Integer>();
        list.add(46);
        list.add(67);
        list.add(24);
        list.add(16);
    }
}
```

```
        list.add(8);
        list.add(12);
        System.out.println ("Value of maximum element from the collection: " +
Collections.max(list));
    }
}

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

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

        List<Integer> list = new ArrayList<Integer>();
        list.add(46);
        list.add(67);
        list.add(24);
        list.add(16);
        list.add(8);
        list.add(12);

        System.out.println ("Value of minimum element from the collection:
"+Collections.min(list));
    }
}
```

Sorting in Collection:

We can sort the elements of:

- String objects
- Wrapper class objects
- User-defined class objects

Collections class provides static methods for sorting the elements of collection.

If collection elements are of Set type, we can use TreeSet.
But We cannot sort the elements of List.

Collections class provides methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements:

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

public void sort(List list): is used to sort the elements of List. List elements must be of Comparable type.

Example of Sorting the elements of List that contains string objects:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

public class Test
{
    public static void main(String [] args)
    {
        ArrayList<String> al = new ArrayList<String>();
        al.add("Viru");
        al.add("Saurav");
        al.add("Mukesh");
        al.add("Tahir");

        Collections.sort(al);
        Iterator itr = al.iterator();
        while (itr.hasNext()) {
            System.out.println (itr.next());
        }
    }
}
```

Example of Sorting the elements of List that contains Wrapper class objects:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
public class Test
{
    public static void main(String [] args)
    {
        ArrayList al = new ArrayList();

        al.add(Integer.valueOf(201));
        al.add(Integer.valueOf(101));
        al.add(230); // internally will be converted into objects as
                    // Integer.valueOf(230)
        Collections.sort(al);

        Iterator itr = al.iterator();
        while (itr.hasNext())
        {
            System.out.println (itr.next());
        }
    }
}
```

```
}
```

Interview Questions

1) What is the difference between ArrayList and Vector?

No. ArrayList

- 1) ArrayList is not synchronized.
- 2) ArrayList is not a legacy class.
- 3) ArrayList increases its size by 50% of the array size.

Vector

- Vector is synchronized.
- Vector is a legacy class.
- Vector increases its size by doubling the array size.

2) What is the difference between ArrayList and LinkedList?

No. ArrayList

- 1) ArrayList uses a dynamic array.
- 2) ArrayList is not efficient for manipulation because a lot of shifting is required.
- 3) ArrayList is better to store and fetch data.

LinkedList

- LinkedList uses doubly linked list.
- LinkedList is efficient for manipulation.
- LinkedList is better to manipulate data.

3) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

No. Iterator

ListIterator

- 1) Iterator traverses the elements in forward direction only. ListIterator traverses the elements in backward and forward directions both.
- 2) Iterator can be used in List, Set and Queue. ListIterator can be used in List only.

4) What is the difference between Iterator and Enumeration?

No. Iterator

Enumeration

- 1) Iterator can traverse legacy and non-legacy elements. Enumeration can traverse only legacy elements.
- 2) Iterator is fail-fast. Enumeration is not fail-fast.
- 3) Iterator is slower than Enumeration. Enumeration is faster than Iterator.

5) What is the difference between List and Set?

List can contain duplicate elements whereas Set contains only unique elements.

6) What is the difference between HashSet and TreeSet?

HashSet maintains **no order** whereas TreeSet maintains **ascending order**.

7) What is the difference between Set and Map?

Set contains values only whereas Map contains key and values both.

8) What is the difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key,value). HashSet can be iterated but HashMap need to convert into Set to be iterated.

9) What is the difference between HashMap and TreeMap?

HashMap maintains **no order** but TreeMap maintains **ascending order**.

10) What is the difference between HashMap and Hashtable?

No. HashMap

Hashtable

- 1) HashMap is not synchronized. Hashtable is synchronized.
- 2) HashMap can contain one null key and multiple null values. Hashtable cannot contain any null key or null value

11) What is the difference between Collection and Collections?

Collection is an interface whereas Collections is a class. Collection interface provides normal

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

functionality of data structure to List, Set and Queue. But, Collections class is to sort and synchronize collection elements.

12) What is the difference between Comparable and Comparator?

No. Comparable

- 1) Comparable provides only one sort of sequence.
- 2) It provides one method named compareTo().
- 3) It is found in java.lang package.
- 4) If we implement Comparable interface, actual class is modified.

Comparator

- Comparator provides multiple sort of sequences.
- It provides one method named compare().
- it is found in java.util package.

Actual class is not modified.

13) What is the advantage of Properties file?

If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.

14) What does the hashCode() method?

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same.

But, it is possible that two hash code numbers can have different or same keys.

15) Why we override equals() method?

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

16) How to synchronize List, Set and Map elements?

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

```
public static List synchronizedList(List l){}
public static Set synchronizedSet(Set s){}
public static SortedSet synchronizedSortedSet(SortedSet s){}
public static Map synchronizedMap(Map m){}
public static SortedMap synchronizedSortedMap(SortedMap m){}
```

17) What is the advantage of generic collection?

If we use generic class, we don't need typecasting. It is typesafe and checked at compile time.

18) What is hash-collision in Hashtable and how it is handled

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

in Java?

Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.

19) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

20) What is the default size of load factor in hashing based collection?

The default size of load factor is **0.75**. The default capacity is computed as initial capacity * load factor. For example, $16 * 0.75 = 12$. So, 12 is the default capacity of Map.

Input and Output streams

Java I/O (Input and Output) is used to process the input and produce the output based on the input.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform file handling in java by java IO API.

Stream:

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream (System errors i.e JVM error message)

Let's see the code to print output and error message to the console.

```
System.out.println ("simple message");
```

```
System.err.println ("error message");
```

```
int i=System.in.read();//returns ASCII code of 1st character .
```

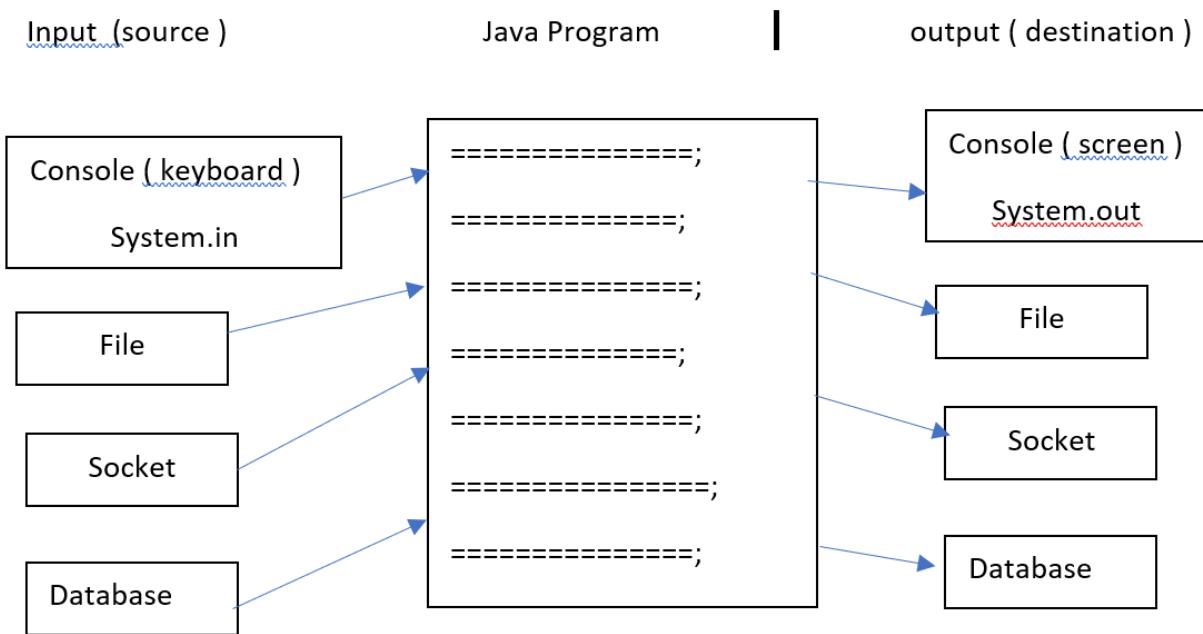
```
System.out.println ((char)i);//will print the character .
```

InputStream (Source of data)

Java application uses an input stream to read data from a source, it may be a file,an array,peripheral device or socket.

OutputStream (Destination of Data)

Java application uses an output stream to write data to a destination, it may be a file,an array,peripheral device or socket.



Character Stream Vs Byte Stream in Java

Byte Streams

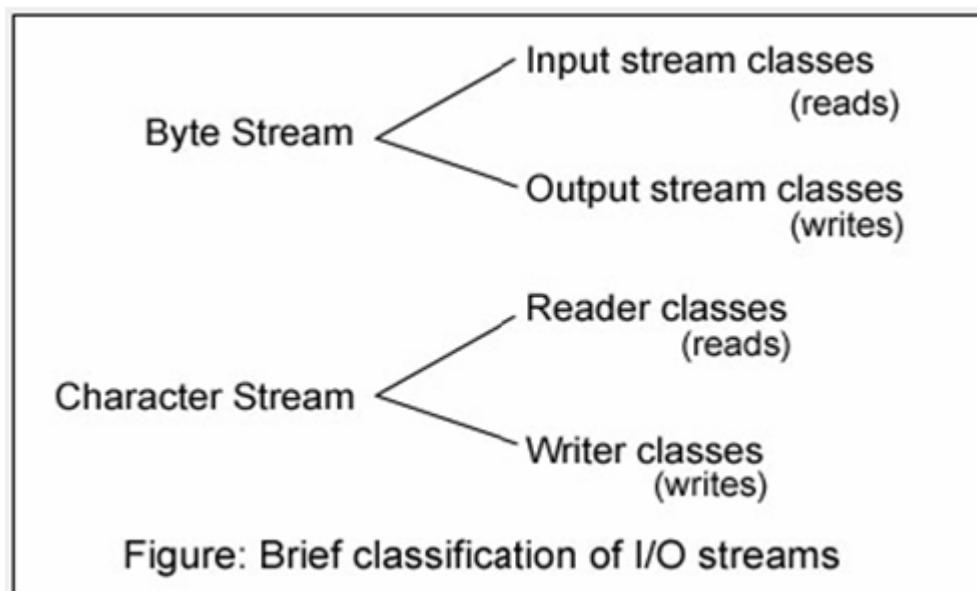
A byte stream access the file byte by byte. Java programs use byte streams to perform input and output of 8-bit bytes. It is suitable for any kind of file, however not quite appropriate for text files. For example, if the file is using a unicode encoding and a character is represented with two bytes, the byte stream will treat these separately and you will need to do the conversion yourself. Byte oriented streams do not use any encoding scheme while Character oriented streams use character encoding scheme(UNICODE). All byte stream classes are descended from InputStream and OutputStream .

Character Streams

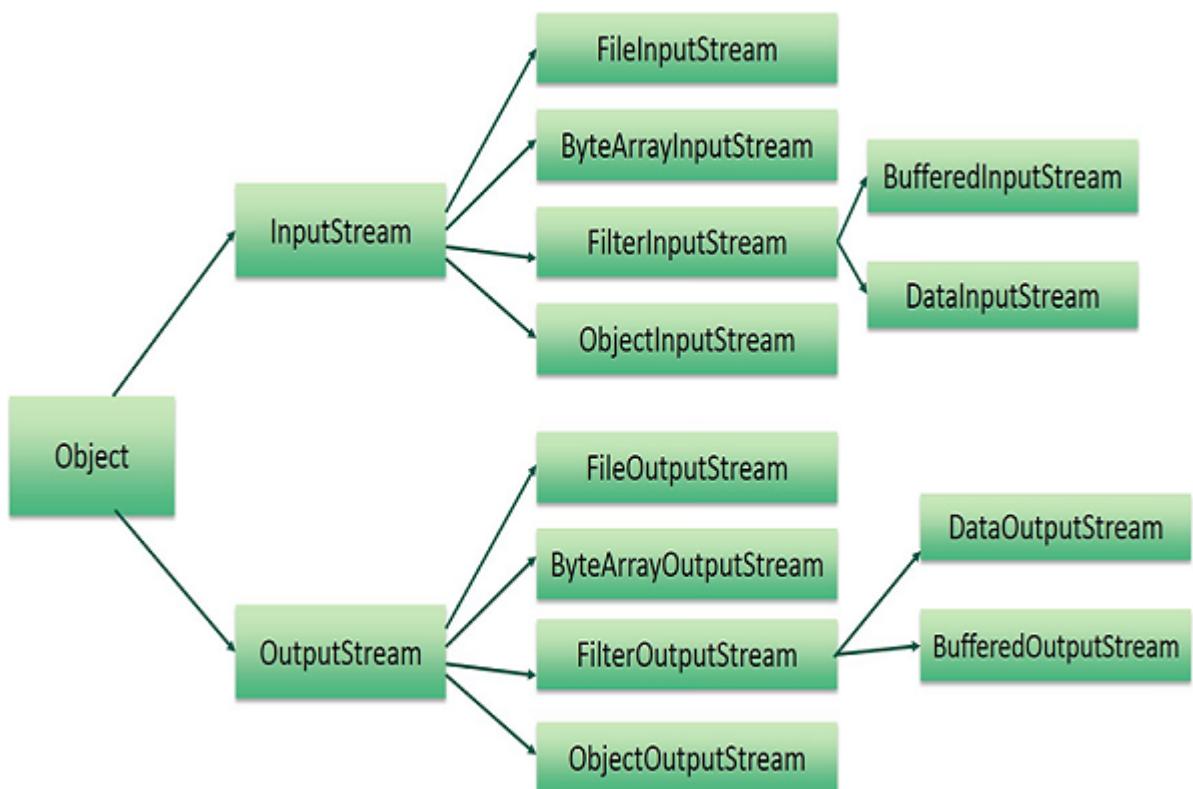
A character stream will read a file character by character. Character Stream is a higher level concept than Byte Stream . A Character Stream is, effectively, a Byte Stream that has been wrapped with

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

logic that allows it to output characters from a specific encoding . That means, a character stream needs to be given the file's encoding in order to work properly. Character stream can support all types of character sets ASCII, Unicode, UTF-8, UTF-16 etc. All character stream classes are descended from Reader and Writer .



Hierarchy of Byte Stream classes.



OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Commonly used methods of OutputStream class

- 1) public void write(int) throws IOException:
is used to write a byte to the required output stream.
- 2) public void write(byte []) throws IOException:
is used to write an array of byte to the required output stream.
- 3) public void flush() throws IOException:
flushes the required output stream.
- 4) public void close()throws IOException:
is used to close the required output stream.

InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

- 1) public abstract int read() throws IOException:
reads the next byte of data from the input stream. It returns -1 at the end of file
- 2) public int available()throws IOException
returns an estimate of the number of bytes that can be read from the required input stream.
- 3) public void close()throws IOException:
is used to close the required input stream.

File Class :

It is class from java.io package to get information about file attributes/properties.

File (String filePath) : constructor to create file object

File file = new File("C:\\project\\test\\file1"); → windows

File file = new File("/home/nrit/file1"); → linux

Since java is an independent of platform we should write common code for both OS.
File file = new File("c:/project/test/file1");

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Use / as path separator for any OS.

Eg:

```
File file = new File("C:/project/test/EMPLOYEE.dat");
```

file.exists() : checking for existence of the file.

file.isFile() : checking for file or not

file.isDirectory() : checking for directory or not

file.canRead() : checking for read permission

file.canWrite() : checking for write permission

file.canExecute() : checking for execute permission

file.length() : to get size of the file in bytes.

file.getParent : to get parent name

file.getName() : it return name of the file

file.renameTo(dest) : to rename a file

file.pathSeparator

file.pathSeparatorChar : OS file path separator.

file.lastModified() : last modified time.

Program for the all above methods.

```
package com.durga.mnrao.xyz;

import java.io.File;
import java.util.Date;

public class Test {

    public static void main(String[] args)  {

        if( args.length != 1 )
        {
            System.out.println("Missing input file Path");
            System.exit(0);
        }

        File filePath = new File( args[0] );
    }
}
```

```
if( ! filePath.exists() )
{
    System.out.println(args[0]+" does not exist");
    System.exit(0);
}

if( filePath.isFile() )
{
    System.out.println(args[0]+" is a file ");
}

if( filePath.isDirectory() )
{
    System.out.println(args[0]+" is dir ");
}

if( filePath.canRead() )
{
    System.out.println("it is readable file");
}

if( filePath.canWrite() )
{
    System.out.println("it can be writable file");
}

if( filePath.canExecute() )
{
    System.out.println("this file can be opened");
}

String path = filePath.getAbsolutePath();

System.out.println("file location = "+path);

String parent = filePath.getParent();

System.out.println("parent = "+parent);

String name = filePath.getName();

System.out.println("file name = "+name);

long size = filePath.length();

System.out.println("file size = "+size+" bytes");

System.out.println("file size = "+(size/1024.0)+" KB");

System.out.println("file size = "+(size/(1024.0*1024.0))+" MB");

System.out.println("file size = "+(size/(1024.0*1024.0*1024))+" GB");

long time = filePath.lastModified();

Date date = new Date(time);

System.out.println("time of last modification = "+date);
```

```
    }  
}
```

Pass file path as command line params and run the above program

runAs → RunConfiguration → arguments → "D:\MNRAO-Java-material\CoreJava\latest.pdf"

to create new file

```
File filePath = new File("D:\\NRIT_Java-material\\CoreJava\\emp.dat");  
(or)  
File filePath = new File(args[0]);  
  
if(filePath.createNewFile())  
{  
    System.out.println("created");  
}  
else  
{  
    System.out.println("failed");  
}
```

To delete file:

```
File filePath = new File(args[0]);  
  
filePath.deleteOnExit();
```

FileInputStream and FileOutputStream (File Handling)

In Java, FileInputStream and FileOutputStream classes are used for file handling in java.

Java FileOutputStream class

Java FileOutputStream is an output stream for writing data to a file.

If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

Program to create file using commandline params

```
D:\\test>java Create file1  
It should take only one parameter.  
It has to take data from keyboard  
i/p:
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

at the end shoud be ctrl+z

step1:

D:\test>notepad Create.java

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class Create {

    public static void main(String[] args) throws IOException {

        if(args.length!=1)
        {
            System.out.println("invalid syntax, usage: java Create <file_name>");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if(filePath.exists())
        {
            System.out.println(args[0]+" already exist, can not create");
            System.exit(0);
        }

        FileOutputStream fos=null;
        try
        {

            fos = new FileOutputStream(filePath);

            char ch = (char )System.in.read();

            while(ch!=(char)-1)
            {
                fos.write(ch);
                ch = (char )System.in.read();
            }
            fos.flush();
            fos.close();
            System.out.println("Successfully created");

        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        finally
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
{  
    if(fos!=null)  
    {  
        fos.close();  
    }  
}  
}  
}
```

D:\test> javac Create.java

D:\test> java Create file1
Hello this mnrao
Welcome to hyderabad
^z (ctrl + z)

Output:

File1 Successfully created

Program to display file data :

D:\test> notepad Display.java

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
  
public class Display {  
  
    public static void main(String[] args) throws IOException {  
  
        if(args.length!=1)  
        {  
            System.out.println("invalid syntax, usage: java Display <file_name>");  
            System.exit(0);  
        }  
  
        File inputFile = new File(args[0]);  
  
        if( !inputFile.exists() )  
        {  
            System.out.println(args[0]+" file not found ");  
            System.exit(0);  
        }  
  
        if( ! inputFile.isFile() )  
        {  
            System.out.println(args[0]+" is not a file ");  
            System.exit(0);  
        }  
    }  
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
System.out.println(args[0]+"not a file ");
System.exit(0);

}

FileInputStream fis = null;
try
{
    fis = new FileInputStream(inputFile);
    char ch = (char)fis.read();

    while(ch!=(char)-1)
    {
        System.out.print(ch);
        ch = (char)fis.read();
    }

    fis.close();
}

catch (Exception e) {
    e.printStackTrace();
}
finally
{
    if(fis!=null)
    {
        fis.close();
    }
}
}

}

$javac Display.java
```

\$java Display file1

Copying files.

D:\test>java MyCopy file1 file2

D:\test>notepad MyCopy.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class MyCopy {

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

```
if( args.length != 2 )
{
    System.out.println("Invalid Syntax, Usage : java Mycopy <src_file>
<dest_file>");

    System.exit(0);
}

File srcFilePath = new File(args[0]);

File destFilePath = new File(args[1]);

if(! srcFilePath.exists())
{
    System.out.println(args[0]+" source does not exist, can not copy");
    System.exit(0);
}

if( ! srcFilePath.isFile())
{
    System.out.println(args[0]+" exist but not a file");
    System.exit(0);
}

if( destFilePath.exists() )
{
    System.out.println(args[1]+" destination already exist can not copy
");

    System.exit(0);
}

FileInputStream fis = null;
FileOutputStream fos = null;

try
{
    fis = new FileInputStream(srcFilePath);

    fos = new FileOutputStream(destFilePath);

    char ch =(char) fis.read();

    while( ch!=(char)-1 )
    {
        fos.write(ch);

        ch =(char) fis.read();
    }

    fis.close();

    fos.close();

    System.out.println("Copied ");
}
catch(Exception e)
{
```

```
        e.printStackTrace();
    }
finally
{
    if(fis!=null)
    {
        fis.close();
    }

    if(fos!=null)
    {
        fos.close();
    }
}

}
```

\$ javac MyCopy.java

\$ java MyCopy file1 file2

Reading and writing lines:

DataInputStream provides readLine() method to read a line

FileInputStream fis= **null**;

DataInputStream dis = **null**;

fis= **new** FileInputStream("employee.dat");

dis = **new** DataInputStream(fis);

String line = dis.readLine(); --> returns null when reach to EOF.

Program to read employee records line by line

"employee.dat" file contains following data.

```
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
```

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class ReadLineByLine {

    public static void main(String[] args) throws IOException {

        if(args.length!=1)
        {
            System.out.println("Input file missing");

            System.exit(0);
        }

        File filePath = new File(args[0]);

        if( ! filePath.exists() )
        {
            System.out.println(args[0]+" not found ");

            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
try
{
    fis = new FileInputStream(filePath);

    dis = new DataInputStream(fis);

    String line = dis.readLine();

    while( line!=null )
    {
        System.out.println(line);

        line = dis.readLine();
    }

    dis.close();

    fis.close();

}

catch(Exception e)
{
    e.printStackTrace();
}

finally
{
    if(dis!=null)
    {
        dis.close();
    }

    if(fis!=null)
    {
        fis.close();
    }
}

}

}

RunAs → RunConfiguration → Java Application ( double click ) → ReadLineByLine → arguments → “C:\project\mnrao\employee.dat” → run
```

program to convert above formatted data into csv file (employee.dat to employee.csv)
Note : employee.csv should also be generated at same location of “employee.dat”

"employee.dat" file contains following data.

```
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
```

program:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class GenerateCSVFile {

    public static void main(String[] args) throws IOException {
        if(args.length!=2)
        {
            System.out.println("Missing input files, submit source and destination files ");
            System.exit(0);
        }

        File srcFilePath = new File(args[0]);
        File destFilePath = new File(args[1]);

        if(!srcFilePath.exists())
        {
            System.out.println(args[0]+" not found ");
        }
    }
}
```

```
        System.exit(0);
    }

    FileInputStream fis = null;
    DataInputStream dis = null;
    FileOutputStream fos = null;
    DataOutputStream dos = null;

    try
    {

        fis = new FileInputStream(srcFilePath);

        dis = new DataInputStream(fis);

        fos = new FileOutputStream(destFilePath);

        dos = new DataOutputStream(fos);

        String line = dis.readLine();

        while( line!=null )
        {

            String result = line.replaceAll(":", ",,");

            dos.writeBytes(result+"\n");

            line = dis.readLine();
        }

        dis.close();

        fis.close();

        dos.close();

        fos.close();

        System.out.println("Successfully data converted ");

    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
```

```
        if(dis!=null)
        {
            dis.close();
        }

        if(fis!=null)
        {
            fis.close();
        }

        if(dos!=null)
        {
            dos.close();
        }

        if(fos!=null)
        {
            fos.close();
        }
    }

}

}
```

RunAs → RunConfiguration → Java Application (double click) → GenerateCSVFile → arguments → “C:\project\mnrao\employee.dat” “C:\project\mnrao\employee.csv” → run

o/p file employee.csv as follows:

```
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
```

Program for selected fields for the below data :

```
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
```

```
package com.durga.mnrao.files;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class EmpDataSelectedFilelds {

    public static void main(String[] args) throws IOException {

        if( args.length!=1 )
        {
            System.out.println("Missing input file ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if(!filePath.exists())
        {
            System.out.println(args[0]+" not found");
            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null;

        try
        {
            fis = new FileInputStream(filePath);
            dis = new DataInputStream(fis);

            String record = dis.readLine();
            while( record != null && ! record.isEmpty() )
            {
                String[] fileds = record.split(",");
                System.out.println(fileds[0]+\t+fileds[1]+\t+fileds[3]+\t+fileds[5]);

                record = dis.readLine();
            }

            dis.close();
            fis.close();

        }
    }
}
```

```
        catch(Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if(dis!=null)
        {
            dis.close();
        }
        if(fis!=null)
        {
            fis.close();
        }

    }
}

}
```

Based conditions:

For the below data :

```
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
```

Only Managers :

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class EmpDataManagers {

    public static void main(String[] args) throws IOException {

        if( args.length!=1 )
        {
            System.out.println("Missing input file ");
            System.exit(0);
        }
    }
}
```

```
File filePath = new File(args[0]);

if(!filePath.exists())
{
    System.out.println(args[0]+" not found");
    System.exit(0);
}

FileInputStream fis = null;
DataInputStream dis = null;

try
{
    fis = new FileInputStream(filePath);

    dis = new DataInputStream(fis);

    String record = dis.readLine();

    while( record != null && ! record.isEmpty())
    {
        String[] fields = record.split(",");
        if(fields[2].equals("manager"))
        {
            System.out.println(record);
        }

        record = dis.readLine();
    }

    dis.close();
    fis.close();
}

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }
    if(fis!=null)
    {
        fis.close();
    }
}
```

```
    }  
}
```

Only Male Managers:

```
import java.io.DataInputStream;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
  
public class EmpDataMaleManagers {  
  
    public static void main(String[] args) throws IOException {  
  
        if( args.length!=1 )  
        {  
            System.out.println("Missing input file ");  
            System.exit(0);  
        }  
  
        File filePath = new File(args[0]);  
  
        if(!filePath.exists())  
        {  
            System.out.println(args[0]+" not found");  
            System.exit(0);  
        }  
  
        FileInputStream fis = null;  
        DataInputStream dis = null;  
  
        try  
        {  
            fis = new FileInputStream(filePath);  
            dis = new DataInputStream(fis);  
  
            String record = dis.readLine();  
            while( record != null && ! record.isEmpty())  
            {  
                String[] fields = record.split(",");  
                if(fields[2].equals("manager") && fields[5].equals("male"))  
                {  
                    System.out.println(record);  
                }  
            }  
        }  
    }  
}
```

```
        record = dis.readLine();
    }

    dis.close();

    fis.close();

}

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }
    if(fis!=null)
    {
        fis.close();
    }
}

}
```

Only sales dept, Male Managers

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class EmpDataMaleManagersDept10 {

    public static void main(String[] args) throws IOException {

        if( args.length!=1 )
        {
            System.out.println("Missing input file ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if(!filePath.exists())
        {
            System.out.println(args[0]+" not found");
            System.exit(0);
        }
    }
}
```

```
FileInputStream fis = null;
DataInputStream dis = null;

try
{
    fis = new FileInputStream(filePath);
    dis = new DataInputStream(fis);

    String record = dis.readLine();
    while( record != null && ! record.isEmpty())
    {
        String[] fields = record.split(",");
        if(fields[2].equals("manager") && fields[3].equals("sales")
&& fields[5].equals("male"))
        {
            System.out.println(record);
        }

        record = dis.readLine();
    }
    dis.close();
    fis.close();
}

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }
    if(fis!=null)
    {
        fis.close();
    }
}

}
```

Only sales dept , Male Managers with Salary Condition :

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class EmpDataMaleManagersDeptSalary {

    public static void main(String[] args) throws IOException {

        if( args.length!=1 )
        {
            System.out.println("Missing input file ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if(!filePath.exists())
        {
            System.out.println(args[0]+" not found");
            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null;

        try
        {
            fis = new FileInputStream(filePath);

            dis = new DataInputStream(fis);

            String record = dis.readLine();

            while( record != null && ! record.isEmpty())
            {
                String[] fields = record.split(",");
                if(fields[2].equals("manager") && fields[3].equals("sales")
&& fields[5].equals("male") && Double.parseDouble(fields[4])>=45000)
                {
                    System.out.println(record);
                }

                record = dis.readLine();
            }

            dis.close();
        }
    }
}
```

```
        fis.close();

    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if(dis!=null)
        {
            dis.close();
        }
        if(fis!=null)
        {
            fis.close();
        }

    }
}

}
```

Program to read records from text file, convert into Employee object and store into ArrayList and display from ArrayList

Data :

```
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
```

POJO class for Employye record

```
package com.durga.mnrao.list;

public class Employee {

    private int empNum;

    private String empName;

    private String empJob;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
private String empDeptName;

private double empSalary;

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 String getEmpDeptName() {
    return empDeptName;
}

public void setEmpDeptName(String empDeptName) {
    this.empDeptName = empDeptName;
}

public double getEmpSalary() {
    return empSalary;
}

public void setEmpSalary(double empSalary) {
    this.empSalary = empSalary;
}

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.list;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

public class EmpDataLoadArrayList {

    public static void main(String[] args) throws IOException {

        if( args.length != 1 )
        {
            System.out.println("Missing input data file ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if( ! filePath.exists() )
        {
            System.out.println(args[0]+" does not exist ");
            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null ;
```

```
try
{
    ArrayList<Employee> empDataList = new
ArrayList<Employee>();

    fis = new FileInputStream(filePath);

    dis = new DataInputStream( fis );

    String record = dis.readLine();

    while( record != null      &&      ! record.isEmpty())
    {
        String[] fields = record.split(",");
        Employee employee = new Employee();

        employee.setEmpNum( Integer.parseInt(fields[0]) );
        employee.setEmpName(fields[1]);
        employee.setEmpJob(fields[2]);
        employee.setEmpDeptName(fields[3]);

        employee.setEmpSalary(Double.parseDouble(fields[4]));

        employee.setEmpGender(fields[5]);
        employee.setEmpAge(Integer.parseInt(fields[6]));
        empDataList.add(employee);

        record = dis.readLine();
    }

    dis.close();
    fis.close();

    System.out.println("Data loaded into array list");

    Iterator<Employee> iterator = empDataList.iterator();

    while( iterator.hasNext() )
    {
        Employee emp = iterator.next();
    }
}
```

```
        int empNum = emp.getEmpNum();

        String empName = emp.getEmpName();

        String empJob = emp.getEmpJob();

        String empDeptName = emp.getEmpDeptName();

        double empSalary = emp.getEmpSalary();

        String empGender = emp.getEmpGender();

        int empAge = emp.getEmpAge();

System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+empDeptName+"\t"+
empSalary+"\t"+empGender+"\t"+empAge);
    }

}
catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if( dis != null )
    {
        dis.close();
    }

    if( fis != null)
    {
        fis.close();
    }
}

}

}
```

Program to find dept numbers of Employees data.

Data as follows “employee.csv”

```
1001,ajay,manager,10,45000,male,38
1002,aiswrya,clerk,20,25000,female,30
1003,varun,manager,30,50000,male,35
1004,amit,manager,10,47000,male,40
1005,kareena,executive,30,15000,female,24
1006,deepak,clerk,20,23000,male,30
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

1007,sunil,accountant,40,13000,male,29
1008,satvik,director,20,80000,male,45
1009,vijay,manager,30,40000,male,35
1010,sandeep,executive,10,50000,male,31

```
package com.durga.mnrao.list;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.TreeSet;

public class EmpDataDeptList {

    public static void main(String[] args) throws IOException {

        if(args.length!=1)
        {
            System.out.println("Missing input files");
            System.exit(0);
        }

        File srcFilePath = new File(args[0]);

        if(!srcFilePath.exists())
        {
            System.out.println(args[0]+" not found ");
            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null;

        try
        {
            fis = new FileInputStream(srcFilePath);
            dis = new DataInputStream(fis);

            TreeSet<Integer> empDeptList = new TreeSet<Integer>();

            String record = dis.readLine();

```

```
while( record != null && ! record.isEmpty() )
{
    String [] fileds = record.split(",");
    int deptNum = Integer.parseInt(fileds[3]);
    empDeptList.add(deptNum);
    record = dis.readLine();
}
dis.close();
fis.close();
System.out.println("Data loaded into list");
Iterator<Integer> iterator = empDeptList.iterator();
while(iterator.hasNext())
{
    Integer num = iterator.next();
    System.out.println(num);
}

}
catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }
    if(fis!=null)
    {
        fis.close();
    }
}
}
```

}

To display records order by deptnum ;

```
package com.durga.mnrao.list;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;

public class EmpDataOrderByDeptNum {

    public static void main(String[] args) throws IOException {

        if(args.length!=1)
        {
            System.out.println("Missing input files");

            System.exit(0);
        }

        File srcFilePath = new File(args[0]);

        if(!srcFilePath.exists())
        {
            System.out.println(args[0]+" not found ");

            System.exit(0);
        }

        FileInputStream fis = null;
        DataInputStream dis = null;

        try
        {
            fis = new FileInputStream(srcFilePath);

            dis = new DataInputStream(fis);

            TreeSet<Integer> empDeptList = new TreeSet<Integer>();
        }
    }
}
```

```
ArrayList<String> empRecordsList = new  
ArrayList<String>();  
  
String record = dis.readLine();  
  
while( record != null && ! record.isEmpty() )  
{  
    String [] fileds = record.split(",");  
    int deptNum = Integer.parseInt(fileds[3]);  
  
    empDeptList.add(deptNum);  
  
    empRecordsList.add(record);  
    record = dis.readLine();  
}  
  
dis.close();  
fis.close();  
  
System.out.println("Data loaded into list");  
  
Iterator<Integer> deptIterator = empDeptList.iterator();  
  
while(deptIterator.hasNext())  
{  
    Integer deptNum = deptIterator.next();  
  
    Iterator<String> empRecordsIterator =  
empRecordsList.iterator();  
  
while(empRecordsIterator.hasNext())  
{  
    String empRecord = empRecordsIterator.next();  
  
    String[] fileds = empRecord.split(",");  
    if( deptNum == Integer.parseInt(fileds[3]) )
```

```
        {
            System.out.println(empRecord);
        }

    }

}

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }

    if(fis!=null)
    {
        fis.close();
    }
}

}

}
```

EmpDataOrderByGender;

```
package com.durga.mnrao.list;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;

public class EmpDataOrderByGender {

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

```
if(args.length!=1)
{
    System.out.println("Missing input files");

    System.exit(0);
}

File srcFilePath = new File(args[0]);

if(!srcFilePath.exists())
{
    System.out.println(args[0]+" not found ");

    System.exit(0);
}

FileInputStream fis = null;

DataInputStream dis = null;

try
{
    fis = new FileInputStream(srcFilePath);

    dis = new DataInputStream(fis);

    TreeSet<String> empGenderList = new TreeSet<String>();

    ArrayList<String> empRecordsList = new
ArrayList<String>();

    String record = dis.readLine();

    while( record != null && ! record.isEmpty() )
    {

        String [] fileds = record.split(",");
        empGenderList.add(fileds[5]);

        empRecordsList.add(record);

        record = dis.readLine();
    }
}
```

```
        }

        dis.close();

        fis.close();

        System.out.println("Data loaded into list");

        Iterator<String> GenderIterator =
empGenderList.iterator();

        while(GenderIterator.hasNext())
{
    String gender = GenderIterator.next();

    Iterator<String> empRecordsIterator =
empRecordsList.iterator();

    while(empRecordsIterator.hasNext())
{

    String empRecord = empRecordsIterator.next();

    String[] fileds = empRecord.split(",");

    if( gender.equals(fileds[5]))
    {
        System.out.println(empRecord);
    }

}

}

}

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
{
```

```
        dis.close();
    }

    if(fis!=null)
    {
        fis.close();
    }

}
```

EmpDataOrderByJob:

```
package com.durga.mnrao.list;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.TreeSet;

public class EmpDataOrderByJob {

    public static void main(String[] args) throws IOException {

        if(args.length!=1)
        {
            System.out.println("Missing input files");

            System.exit(0);
        }

        File srcFilePath = new File(args[0]);

        if(!srcFilePath.exists())
        {
            System.out.println(args[0]+" not found ");

            System.exit(0);
        }

        FileInputStream fis = null;
```

```
DataStream dis = null;

try
{
    fis = new FileInputStream(srcFilePath);
    dis = new DataInputStream(fis);

    TreeSet<String> empJobList = new TreeSet<String>();
    ArrayList<String> empRecordsList = new
ArrayList<String>();

    String record = dis.readLine();

    while( record != null && ! record.isEmpty() )
    {
        String [] fileds = record.split(",");
        empJobList.add(fileds[2]);

        empRecordsList.add(record);
        record = dis.readLine();
    }

    dis.close();
    fis.close();
    System.out.println("Data loaded into list");
    Iterator<String> JobIterator = empJobList.iterator();

    while(JobIterator.hasNext())
    {
        String job = JobIterator.next();

        Iterator<String> empRecordsIterator =
empRecordsList.iterator();
```

```
        while(empRecordsIterator.hasNext())
        {
            String empRecord = empRecordsIterator.next();

            String[] fileds = empRecord.split(",");
            if( job.equals(fileds[2]))
            {
                System.out.println(empRecord);
            }

        }

    }

catch(Exception e)
{
    e.printStackTrace();
}
finally
{
    if(dis!=null)
    {
        dis.close();
    }

    if(fis!=null)
    {
        fis.close();
    }
}

}
```

Java SequenceInputStream Example

Example that reads the data from two files and writes screen

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class Test
{
    public static void main(String[] args) throws IOException
    {
        File file1 = new File("D:\\testin1.txt");
        File file2 = new File("D:\\testin2.txt");

        FileInputStream fis1= new FileInputStream(file1);

        FileInputStream fis2=new FileInputStream(file2);

        SequenceInputStream inst = new SequenceInputStream(fis1, fis2);

        char ch=(char) inst.read();

        while(ch!=(char)-1)
        {
            System.out.print(ch);
            ch=(char)inst.read();
        }

        inst.close();
        fis1.close();
        fis2.close();
    }
}
```

Example that reads the data from two files and writes into another file

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
import java.io.SequenceInputStream;

public class Test
{
    public static void main(String[] args) throws IOException
    {
        File file1 = new File("D:\\testin1.txt");
        File file2 = new File("D:\\testin2.txt");
        File file3 = new File("D:\\testout.txt");

        FileInputStream input1= new FileInputStream(file1);

        FileInputStream input2=new FileInputStream(file2);

        FileOutputStream output=new FileOutputStream(file3);

        SequenceInputStream inst=new SequenceInputStream(input1, input2);

        char ch=(char)inst.read();

        while(ch!=<char>-1)
        {
            output.write(ch);

            ch=(char)inst.read();
        }

        inst.close();
        input1.close();
        input2.close();
        output.close();
    }
}
```

Java Console Class:

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The `java.io.Console` class is attached with system console internally. The `Console` class is introduced since 1.5.

Let's see a simple example to read text from console.

```
public final class Console extends Object implements Flushable
```

Method	Description
Reader reader()	It is used to retrieve the reader object associated with the console
String readLine()	It is used to read a single line of text from the console.
String readLine(String fmt, Object... args)	It provides a formatted prompt then reads the single line of text from the console.
char[] readPassword()	It is used to read password that is not being displayed on the console.
char[] readPassword(String fmt, Object... args)	It provides a formatted prompt then reads the password that is not being displayed on the console.
Console format(String fmt, Object... args)	It is used to write a formatted string to the console output stream.
Console printf(String format, Object... args)	It is used to write a string to the console output stream.
PrintWriter writer()	It is used to retrieve the PrintWriter object associated with the console.
void flush()	It is used to flushes the console.

To read data from console:

```

import java.io.Console;

public class ConsoleTest
{
    public static void main(String[] args)
    {
        Console c = System.console();
        System.out.println ("Enter your name: ");
        String name= c.readLine();
        System.out.println ("Welcome "+name);
    }
}

```

Password example:

```

import java.io.Console;

public class ConsoleTest
{
    public static void main(String[] args)
    {
        Console c = System.console();

        System.out.println ("Enter your name: ");

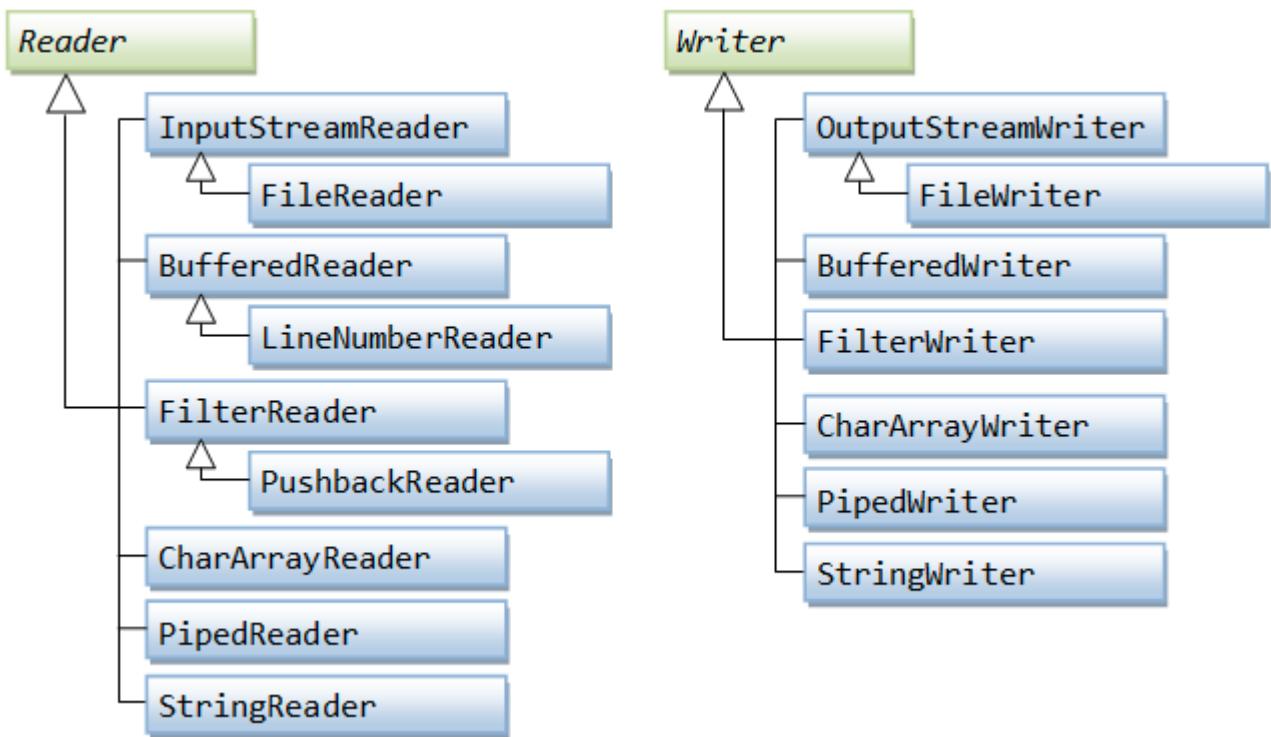
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
String name = c.readLine();
System.out.println ("Enter your passwd: ");
char [] ch = c.readPassword();
String pass = String.valueOf(ch);

if(name.equals("mnrao") && pass.equals("java"))
{
    System.out.println ("Valid User");
}
else
{
    System.out.println ("Invalid user");
}
}
```

Character Streams



FileWriter class to create a file

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Create {

    public static void main(String[] args) throws IOException {

        if( args.length!=1)
        {
            System.out.println("Invalid syntax, Usage : java
Create <file_name> ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        if( filePath.exists() )
```

```
{  
    System.out.println(args[0]+" already exist ");  
  
    System.exit(0);  
}  
  
FileWriter fw = null;  
  
try  
{  
    fw = new FileWriter(filePath);  
  
    char ch = (char) System.in.read();  
  
    while( ch != (char) -1 )  
    {  
        fw.write( ch );  
  
        ch = (char) System.in.read();  
    }  
  
    fw.close();  
  
    System.out.println("successfully created ");  
}  
catch(Exception e)  
{  
    e.printStackTrace();  
}  
finally  
{  
    if( fw != null )  
    {  
        fw.close();  
    }  
}  
  
}  
}
```

FileReader Example:

```
package com.durga.mnrao.cstream;  
  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;
```

```
public class Display {  
  
    public static void main(String[] args) throws IOException {  
  
        if( args.length!=1)  
        {  
            System.out.println("Invalid syntax, Usage : java Display  
<file_name> ");  
  
            System.exit(0);  
        }  
  
        File filePath = new File(args[0]);  
  
        if( ! filePath.exists() )  
        {  
            System.out.println(args[0]+" not found can not display  
");  
  
            System.exit(0);  
        }  
  
        FileReader fr =null;  
  
        try  
        {  
            fr = new FileReader(filePath);  
  
            char ch = (char) fr.read();  
  
            while( ch!=(char)-1 )  
            {  
                System.out.print(ch);  
                ch = (char) fr.read();  
            }  
  
            fr.close();  
        }  
        catch(Exception e)  
        {  
            e.printStackTrace();  
        }  
  
        finally  
        {  
            if(fr!=null)  
            {  
                fr.close();  
            }  
        }  
    }  
}
```

}

}

}

Java BufferedReader class:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ReadLineByLine {

    public static void main(String[] args) throws IOException {

        if( args.length!=1)
        {
            System.out.println("Invalid syntax, Usage : java Display
<file_name> ");

            System.exit(0);
        }

        File filePath = new File(args[0]);

        if( ! filePath.exists() )
        {
            System.out.println(args[0]+" not found can not display
");

            System.exit(0);
        }

        FileReader fr =null;
        BufferedReader br = null;

        try
        {
            fr = new FileReader(filePath);

            br = new BufferedReader(fr);

            String line = br.readLine();
        }
    }
}
```

```
        while( line !=null  )
    {
        System.out.println(line);
        line = br.readLine();
    }
    br.close();
    fr.close();
}
catch(Exception e)
{
    e.printStackTrace();
}

finally
{
    if(br!=null)
    {
        br.close();
    }
    if(fr!=null)
    {
        fr.close();
    }
}
}
```

BufferedWriter Example:

program to convert belowe format into csv file (employee.dat to employee.csv)
Note : employee.csv should also be generated at same location of “employee.dat”

"employee.dat" file contains following data.

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

Convert into employee.csv Format as below

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

```
package com.durga.mnrao.cstream;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class GenerateCSVFile {

    public static void main(String[] args) throws IOException {

        if(args.length!=2)
        {
            System.out.println("Missing input files, submit source
and destination files ");

            System.exit(0);
        }

        File srcFilePath = new File(args[0]);

        File destFilePath = new File(args[1]);

        if(!srcFilePath.exists())
        {
            System.out.println(args[0]+" not found ");

            System.exit(0);
        }

        FileReader fr = null;

        BufferedReader br = null;

        FileWriter fw = null;

        BufferedWriter bw = null;

        try
        {
            fr = new FileReader(srcFilePath);

            bw = new BufferedWriter(new FileWriter(destFilePath));
        }
        catch(IOException e)
        {
            System.out.println("Error "+e.getMessage());
        }
    }
}
```

```
br = new BufferedReader(fr);

fw = new FileWriter(destFilePath);

bw = new BufferedWriter(fw);

String line = br.readLine();

while( line!=null )
{

    String result = line.replaceAll(":", ", ");

    bw.write(result+"\n");

    line = br.readLine();
}

br.close();

fr.close();

bw.close();

fw.close();

System.out.println("Successfully data converted ");

}

catch(Exception e)
{
    e.printStackTrace();
}

finally
{
    if(br!=null)
    {
        br.close();
    }

    if(fr!=null)
    {
        fr.close();
    }

    if(bw!=null)
    {
        bw.close();
    }
}
```

```
        if(fw!=null)
    {
        fw.close();
    }
}

}
```

Above program submit as below

Runas → Runconfiguration → arguments

C:\project\mnrao\employee.dat C:\project\mnrao\employee.csv

Reading Line from console by InputStreamReader and BufferedReader

```
import java.io.*;

public class BufferedReaderExample
{
    public static void main(String [] args) throws Exception
    {
        InputStreamReader isr = new InputStreamReader(System.in);

        BufferedReader br = new BufferedReader( isr );

        System.out.println ("Enter your name");

        String name = br.readLine();

        System.out.println ("Welcome " + name);
    }
}
```

PrintStream class Example:

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

it provides print() and println () over loaded methods for different data types.

```
import java.io.FileOutputStream;
import java.io.PrintStream;
public class Test
{
    public static void main(String [] args) throws Exception
    {
        FileOutputStream fout=new FileOutputStream("D:\\testout.txt ");
        PrintStream pout = new PrintStream(fout);
        pout.println (2016);
        pout.println (2000.50);
        pout.println ("Welcome to NR IT Solutions");
        pout.close();
        fout.close();
        System.out.println ("Success?");
    }
}
```

PrintWriter class Example:

Example of writing the data on a console and in a text file testout.txt using Java PrintWriter class.

```
import java.io.File;
import java.io.PrintWriter;
public class Test
{
    public static void main(String[] args) throws Exception
    {
        // Data to write on Console using PrintWriter
        PrintWriter writer = new PrintWriter(System.out);

        writer.write("Welcome to NR IT Solutions");

        writer.flush();

        writer.close();

        // Data to write in File using PrintWriter
        PrintWriter writer1 = null;

        writer1 = new PrintWriter(new File("D:\\testout.txt"));

        writer1.write("Training on Java, Spring, Hibernate");

        writer1.flush();

        writer1.close();
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
}
```

System.out.printf():

```
=====
```

output formatting:

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

```
{
```

```
    System.out.printf("%5d %10s",10, "hello");
```

```
}
```

Loading Properties file data

Properties class provide following methods

1. Object setProperty(String key, String value) it takes key and value pair
2. String getProperty(String key)

Properties file data as below:

Path = C:\project\DatabaseProperties.prop

```
dbDriver=com.mysql.cj.jdbc.Driver
dbHost=localhost
dbPort=3305
dbUid=root
dbPasswd=root
dbName=mnrao
```

```
package com.durga.mnrao.prop;

import java.io.File;
import java.io.FileReader;
import java.util.Properties;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public class LoadDatabaseProperties {  
  
    public static void main(String[] args) {  
  
        if(args.length !=1 )  
        {  
            System.out.println("Missing properties file");  
  
            System.exit(0);  
        }  
  
        File filePath = new File(args[0]);  
  
        if( ! filePath.exists() )  
        {  
            System.out.println(args[0]+" does not exist ");  
  
            System.exit(0);  
        }  
  
        FileReader freader = null;  
  
        try  
        {  
  
            freader = new FileReader(filePath);  
  
            Properties dbProp = new Properties();  
  
            dbProp.load(freader);  
  
            freader.close();  
  
            System.out.println("Properties loaded ");  
  
            String driver = dbProp.getProperty("dbDriver");  
  
            String host = dbProp.getProperty("dbHost");  
  
            String port = dbProp.getProperty("dbPort");  
  
            String uid = dbProp.getProperty("dbUid");  
  
            String pwd = dbProp.getProperty("dbPasswd");  
  
            String name = dbProp.getProperty("dbName");  
  
            System.out.println("db driver = "+driver);  
  
            System.out.println("db host = "+host);  
  
            System.out.println("db prot num = "+port);  
  
            System.out.println("db uid = "+uid);  
  
            System.out.println("db password = "+pwd);  
  
            System.out.println("db name = "+name);  
  
        }  
    }  
}
```

```
        catch(Exception e)
        {
            e.printStackTrace();
        }

    }
```

runAs → RunConfiguartion → JavaApplication → LoadDatabaseProperties → arguments

C:\project\DatabaseProperties.prop

Apply → run

Excel Data Parsing

1001	scott	50000.	5	admin	male	28
1002	blake	30000.	2	sales	male	30
1003	dona	60000.	2	accounts marketin	female	29
1004	henry	45000	g	female	31	
1005	martin	35000	finance	female	31	
1006	david	75000	it	male	40	
1007	rathod	25000	accounts	female	22	
1008	jones	55000	it	male	30	
1009	sandeep	45000	admin	male	28	
1010	vijay	35000	sales	male	31	
1011	Dona	25000	accounts	female	24	
1012	Gourav Keerthan	35000	it	male	23	
1013	a	23000	sales	female	25	

Prepare excel file with above data

Pojo class

```
package com.durga.mnrao.exel;

public class Employee {

    private int empNum;

    private String empName;

    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;
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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.exel;

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
import java.util.Iterator;
import java.util.List;

import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ExcelParser {

    public List<Employee> parseExceldata(String filePath)
    {

        File file = new File(filePath);

        FileInputStream fis = null;

        DataInputStream dis = null;

        List<Employee> list = null;

        try
        {

            fis = new FileInputStream(file);

            dis = new DataInputStream(fis);

            list = new ArrayList<Employee>();

            XSSFWorkbook workBook = new XSSFWorkbook(dis);

            XSSFSheet sheet = workBook.getSheetAt(0);

            Iterator<Row> rowIterator = sheet.rowIterator();

            while( rowIterator.hasNext() )
            {
                Row row = rowIterator.next();

                Cell cell = row.getCell(0);

                int empNum = (int) cell.getNumericCellValue();

                cell = row.getCell(1);

                String empName = cell.getStringCellValue();
            }
        }
    }
}
```

```
        cell = row.getCell(2);

        double empSalary = cell.getNumericCellValue();

        cell = row.getCell(3);

        String empDeptName = cell.getStringCellValue();

        cell = row.getCell(4);

        String empGender = cell.getStringCellValue();

        cell = row.getCell(5);

        int empAge = (int) cell.getNumericCellValue();

Employee employee = new Employee();

employee.setEmpNum(empNum);
employee.setEmpName(empName);
employee.setEmpSalary(empSalary);
employee.setEmpDeptName(empDeptName);
employee.setEmpGender(empGender);
employee.setEmpAge(empAge);

list.add(employee);

}

workBook.close();

dis.close();

fis.close();

}

catch(Exception e )
{
    e.printStackTrace();
}

return list;
}

=====
```

```
package com.durga.mnrao.exel;

import java.util.Iterator;
import java.util.List;

public class ExcelFileReader {

    public static void main(String[] args) {

        if(args.length!=1)
        {
            System.out.println(args[0]+" missing excel file");

            System.exit(0);
        }

        String filePath = args[0];

        ExcelParser parser = new ExcelParser();

        List<Employee> empRecordsList =
parser.parseExceldata(filePath);

        Iterator<Employee> iterator = empRecordsList.iterator();

        while(iterator.hasNext())
        {
            Employee emp = iterator.next();

            int empNum = emp.getEmpNum();

            String empName = emp.getEmpName();

            double empSalary = emp.getEmpSalary();

            String empDeptName = emp.getEmpDeptName();

            String empGender = emp.getEmpGender();

            int empAge = emp.getEmpAge();

System.out.println(empNum+"\t"+empName+"\t"+empSalary+"\t"+empDeptName+"\t"+empGender);
        }
    }
}
```

}

}

In eclipse :

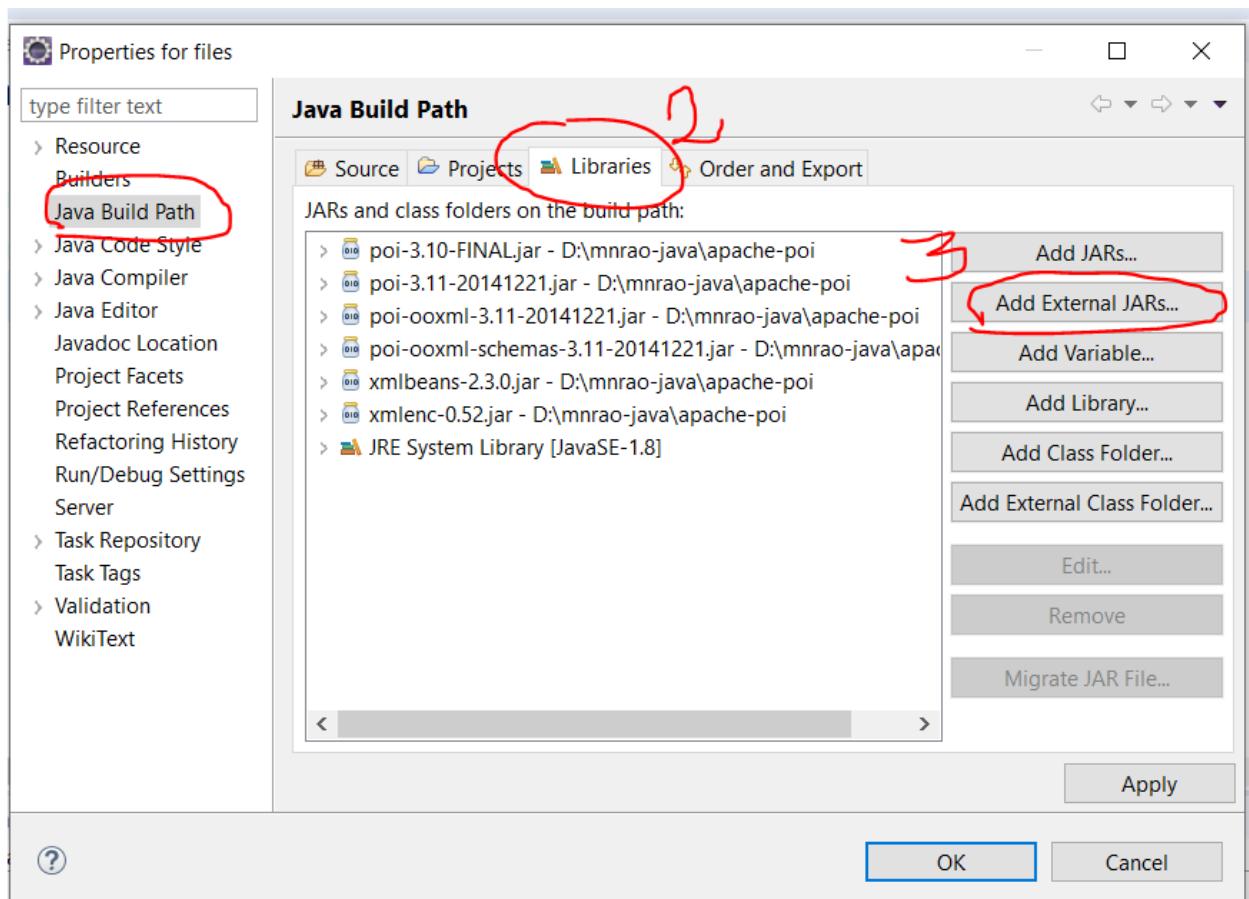
Right click on main () → RunAs → RunConfiguration → JavaApplication → Arguments tab:

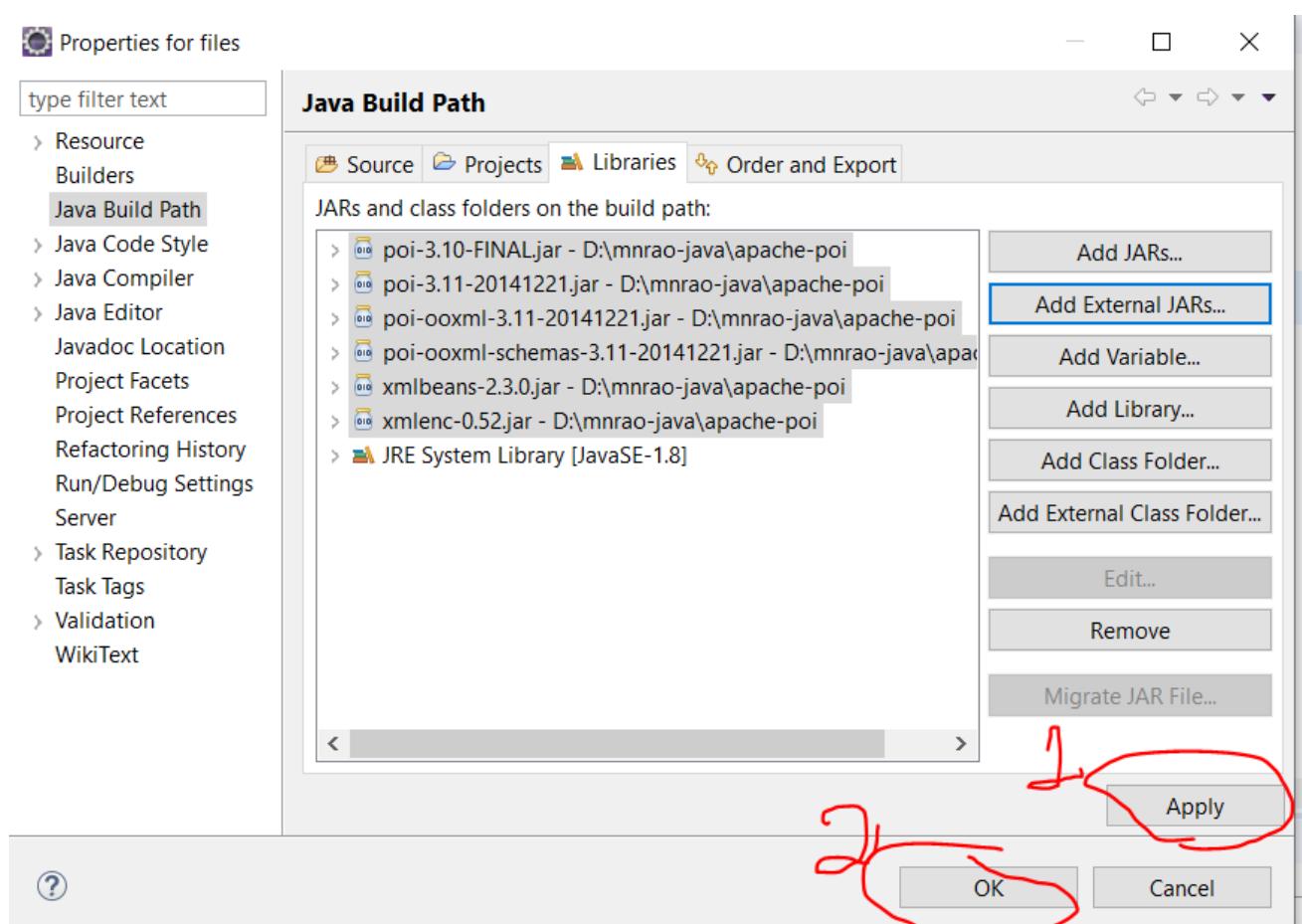
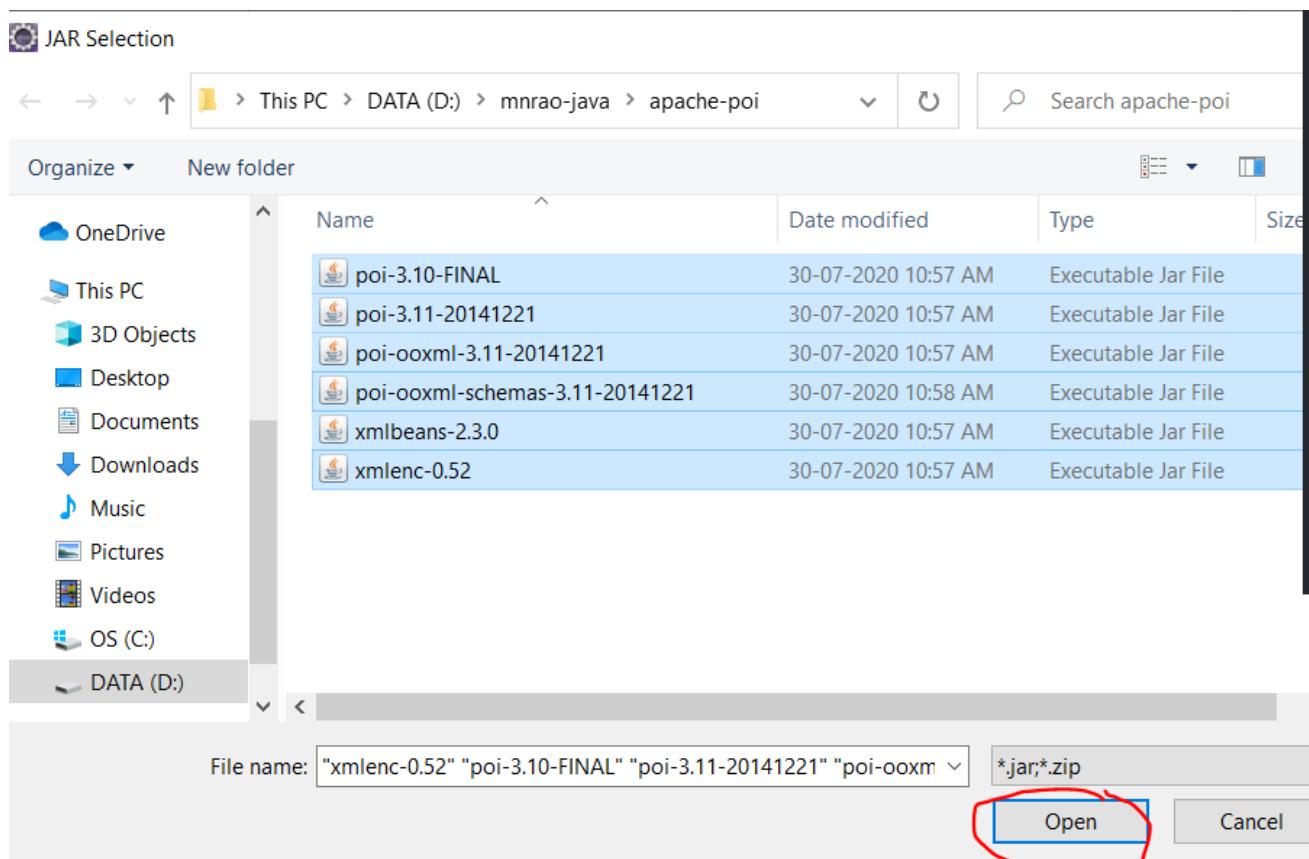
Windows:

C:\project\employee.xlsx

Adding external Apache poi jars

Right click on project → properties → java buildpath





Same above example Generating CSV File

ExcelParser and Employee Classes are same

(main to convert Excel file to CSV file)

```
package com.durga.mnrao.excel;

import java.io.IOException;
import java.util.List;

public class ExcelReaderTest {

    public static void main(String[] args) throws IOException{

        if(args.length!=2)
        {
            System.out.println("Missing input and output parameters ");
            System.exit(0);
        }

        String excelFilePath = args[0];
        String csvFilePath = args[1];
        List<Employee> empRecordsList = null;

        ExcelParser parser = new ExcelParser();
        empRecordsList = parser.parseExcelData(excelFilePath);

        CSVFileGenerator generator = new CSVFileGenerator();
        generator.generateCSVFile(csvFilePath, empRecordsList);
        System.out.println("Successfully Converted from excel to CSV ");

    }
}

package com.durga.mnrao.excel;

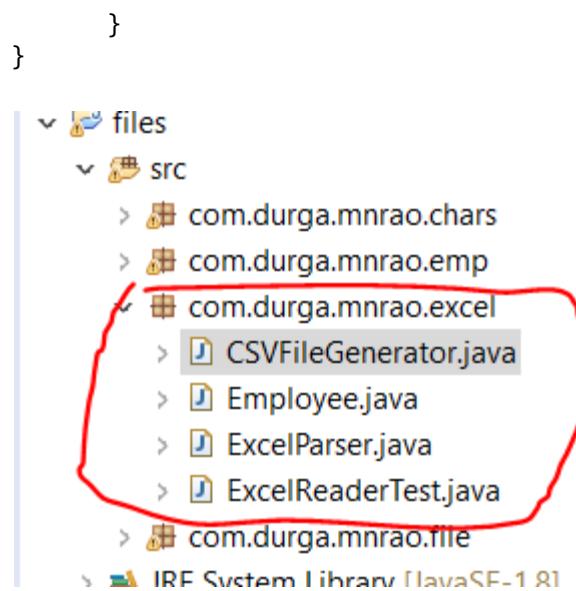
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
```

```
public class CSVFileGenerator {

    public void generateCSVFile(String csvFilePath, List<Employee> empRecordsList)
throws IOException
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
  
    File filePath = new File(csvFilePath);  
  
    FileOutputStream fos = null;  
  
    DataOutputStream dos = null;  
  
    try  
    {  
        fos = new FileOutputStream(filePath);  
  
        dos = new DataOutputStream(fos);  
  
        Iterator<Employee> iterator = empRecordsList.iterator();  
  
        while(iterator.hasNext())  
        {  
            Employee emp = iterator.next();  
  
            int empNum = emp.getEmpNum();  
  
            String empName = emp.getEmpName();  
  
            double empSalary = emp.getEmpSalary();  
  
            String empDeptName = emp.getEmpDeptName();  
  
            String empGender = emp.getEmpGender();  
  
            int empAge = emp.getEmpAge();  
  
            String record =  
empNum+", "+empName+", "+empSalary+", "+empDeptName+", "+empGender+", "+empAge;  
  
            dos.writeBytes(record+"\n");  
  
        }  
  
        dos.close();  
  
        fos.close();  
  
    }  
    catch(Exception e)  
    {  
        e.printStackTrace();  
    }  
    finally{  
  
        if(dos!=null)  
        {  
            dos.close();  
        }  
  
        if(fos!=null)  
        {  
            fos.close();  
        }  
    }  
}
```



XML Parser

Using DOM Parser:

Eg:

Sample Data:

Employee.xml:

```
=====  
<employees>  
  
<employee>  
  
<id>111</id>  
  
<firstName>Lokesh</firstName>  
  
<lastName>Gupta</lastName>
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
<location>India</location>

</employee>

<employee>

<id>222</id>
<firstName>Alex</firstName>

<lastName>Gussin</lastName>

<location>Russia</location>

</employee>

<employee>

<id>333</id>

<firstName>David</firstName>

<lastName>Feezor</lastName>

<location>USA</location>

</employee>

<employee>

<id>444</id>

<firstName>Russ</firstName>

<lastName>Dawson</lastName>

<location>USA</location>

</employee>

</employees>
```

Result file emp.csv:

111,Lokesh,Gupta,India
222,Alex,Gussin,Russia
333,David,Feezor,USA
444,Russ,Dawson,USA

package com.nrit.mnrao.xml;

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class XMLParser
{
    public String parseXMLData(String xmlFilePath, String keyElement)
        throws IOException, InterruptedException
    {
        File file = new File(xmlFilePath);

        FileInputStream fis = new FileInputStream(file);
        DataInputStream dis = new DataInputStream(fis);

        // Get Document Builder
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        Document document = null;
        try
        {
            builder = factory.newDocumentBuilder();
        } catch (ParserConfigurationException e1)
        {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

        // Build Document
        try
        {
            document = builder.parse(dis);
        } catch (SAXException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Normalize the XML Structure; It's just too important !!
        document.getDocumentElement().normalize();
    }
}
```

```
// Here comes the root node
Element root = document.getDocumentElement();
System.out.println (root.getNodeName());

// Get all employees
NodeList nList = document.getElementsByTagName(keyElement);

StringBuffer buffer = new StringBuffer();

System.out.println (nList.getLength());

for (int temp = 0; temp < nList.getLength(); temp++)
{
    Node node = nList.item(temp);
    System.out.println (""); // Just a separator
    if (node.getNodeType() == Node.ELEMENT_NODE)
    {
        // Print each employee's detail
        Element eElement = (Element) node;
        // buffer.append(eElement.getAttribute("id")+",");
        buffer.append(eElement.getElementsByTagName("id").item(0).getTextContent()
                    + ",");

        buffer.append(eElement.getElementsByTagName("firstName").item(0).getTextContent()
                    + ",");

        buffer.append(eElement.getElementsByTagName("lastName").item(0).getTextContent()
                    + ",");

        buffer.append(eElement.getElementsByTagName("location").item(0).getTextContent());
        buffer.append("\n");
    }
}

String data = buffer.toString();

System.out.println (data);

return data;
}

package com.nrit.mnrao.xml;

import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class CSVFile {  
  
    public void generateCSVFile(String [] records, String targetParth) throws IOException  
{  
  
    File file = new File(targetParth);  
    FileOutputStream fos = null;  
    DataOutputStream dos = null;  
  
    try  
    {  
        fos = new FileOutputStream(file);  
        dos = new DataOutputStream(fos);  
  
        for (String string : records) {  
  
            System.out.println (string);  
            dos.writeBytes(string+"\n");  
  
        }  
        dos.close();  
        fos.close();  
    }  
    catch(Exception e)  
    {  
  
    }  
    finally  
    {  
        if(dos!=null)  
        {  
            dos.close();  
        }  
  
        if(fos!=null)  
        {  
            fos.close();  
        }  
  
    }  
}  
}  
  
package com.visix.mnrao.xml;  
  
import java.io.IOException;  
  
public class Test {
```

```
public static void main(String[] args) throws IOException, InterruptedException
{
    String sourceFilePath = args[0];
    String keyElement=args[1];
    String targetFilePath = args[2];
    System.out.println (sourceFilePath);
    XMLParser parser = new XMLParser();

    String xmlData = parser.parseXMLData(sourceFilePath, keyElement);
    String [] records = xmlData.split("\n");
    CSVFile csvFile = new CSVFile();
    csvFile.generateCSVFile(records, targetFilePath);
    System.out.println("Successfully");
}
}
```

In eclipse :

Right click on main () RunAs RunConfiguration JavaApplication
Arguments tab:

Windows:

D:\\hadoop\\employee.xml employee D:\\hadoop\\emp.csv

Linux:

/home/demo/hadoop/ employee.xml employee /home/demo/hadoop/ employee.csv

(source path of data searchKey DestinationPath of Data)

Serialization

Serialization in java is a mechanism of *writing the state of an object into a byte stream.*

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called *deserialization* (Recollecting Object)

Advantage of Java Serialization

It is mainly used to travel object's state on the network (known as marshaling) or to store object into the file.

java.io.Serializable interface

Serializable is a marker interface (has no data member and method).

It is used to "mark" java classes so that objects of these classes may get certain capability.

The Cloneable and Remote are also marker interfaces.

The String class and all the wrapper classes implements *java.io.Serializable* interface by default.

Persistance → permanent storage

Object Persistance : it is a storing an Object into the file permanently

For object persistence serialization required.

By implementing Serializable interface we can make class object as Serializable object

```
package com.durga.mnrao.serde;  
  
import java.io.Serializable;  
  
public class Employee implements Serializable{  
  
    private int empNum;  
  
    private String empName;  
  
    private String empJob;  
  
    private double empSalary;  
  
    private String empDeptName;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
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 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;
}

public String getEmpJob() {
    return empJob;
}

public void setEmpJob(String empJob) {
    this.empJob = empJob;
}

}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

```
public ObjectOutputStream(OutputStream out) throws IOException  
creates an ObjectOutputStream that writes to the specified OutputStream.
```

Methods

- 1) public final void writeObject(Object obj) throws IOException writes the specified object to the ObjectOutputStream.
- 2) public void flush() throws IOException flushes the current output stream.
- 3) public void close() throws IOException closes the current output stream.

Storing Object into File:

```
package com.durga.mnrao.serde;  
  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.ObjectOutputStream;  
  
public class ObjectPersistTest {  
  
    public static void main(String[] args) {  
  
        if(args.length!=1)  
        {  
            System.out.println("Missing outytput file  
");  
            System.exit(0);  
        }  
  
        File filePath = new File(args[0]);  
  
        FileOutputStream fos = null;  
        ObjectOutputStream oos = null;  
  
        try  
        {  
            fos = new FileOutputStream(filePath);  
            oos = new ObjectOutputStream(fos);  
            Employee employee = new Employee();  
            employee.setEmpNum(1001);
```

```
employee.setEmpName("mnrao");
employee.setEmpJob("Architect");
employee.setEmpSalary(50505.50);
employee.setEmpDeptName("it");
employee.setEmpGender("male");
employee.setEmpAge(35);
oos.writeObject(employee);

employee = new Employee();
employee.setEmpNum(1002);
employee.setEmpName("scott");
employee.setEmpJob("Manager");
employee.setEmpSalary(60505.50);
employee.setEmpDeptName("admin");
employee.setEmpGender("female");
employee.setEmpAge(40);
oos.writeObject(employee);

employee = new Employee();
employee.setEmpNum(1003);
employee.setEmpName("flag");
employee.setEmpJob("clerk");
employee.setEmpSalary(30505.50);
employee.setEmpDeptName("sales");
employee.setEmpGender("male");
employee.setEmpAge(28);
oos.writeObject(employee);

oos.close();
fos.close();
System.out.println("Successfully saved to
file");
}
catch(Exception e)
```

```
        {
            e.printStackTrace();
        }

    }
}
```

Deserialization

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

public ObjectInputStream(InputStream in) throws IOException

creates an ObjectInputStream that reads from the specified InputStream.

1) public final Object readObject() throws IOException, ClassNotFoundException

→ reads an object from the input stream.

2) public void close() throws IOException closes **ObjectInputStream**

Reading Object from the file:

```
package com.durga.mnrao.serde;

import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class ObjectDePersistTest {

    public static void main(String[] args) {

        if(args.length!=1)
        {
            System.out.println("Missing input file ");
            System.exit(0);
        }

        File filePath = new File(args[0]);

        FileInputStream fis = null;
        ObjectInputStream ois = null;

        try
```

```
{  
  
    fis = new FileInputStream(filePath);  
  
    ois = new ObjectInputStream(fis);  
  
    Object obj = ois.readObject();  
  
    if( obj instanceof Employee)  
    {  
        Employee emp = (Employee) obj;  
  
        int empNum = emp.getEmpNum();  
  
        String empName = emp.getEmpName();  
  
        String empJob = emp.getEmpJob();  
  
        double empSalary =  
            emp.getEmpSalary();  
  
        String empDeptName =  
            emp.getEmpDeptName();  
  
        String empGender =  
            emp.getEmpGender();  
  
        int empAge = emp.getEmpAge();  
  
        System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+emp  
Salary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);  
    }  
  
    obj = ois.readObject();  
  
    if( obj instanceof Employee)  
    {  
        Employee emp = (Employee) obj;  
  
        int empNum = emp.getEmpNum();  
  
        String empName = emp.getEmpName();  
  
        String empJob = emp.getEmpJob();  
  
        double empSalary =  
            emp.getEmpSalary();  
  
        String empDeptName =  
            emp.getEmpDeptName();  
  
        String empGender =  
            emp.getEmpGender();  
  
        int empAge = emp.getEmpAge();  
  
        System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+emp  
Salary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);  
    }  
}
```

```
        }

        obj = ois.readObject();

        if( obj instanceof Employee)
        {
            Employee emp = (Employee) obj;

            int empNum = emp.getEmpNum();

            String empName = emp.getEmpName();

            String empJob = emp.getEmpJob();

            double empSalary =
                emp.getEmpSalary();

            String empDeptName =
                emp.getEmpDeptName();

            String empGender =
                emp.getEmpGender();

            int empAge = emp.getEmpAge();

            System.out.println(empNum+"\t"+empName+"\t"+empJob+"\t"+emp
Salary+"\t"+empDeptName+"\t"+empGender+"\t"+empAge);

        }

        ois.close();

        fis.close();

    }

}

}

}
```

Java Serialization with static data member

If there is any static data member in a class, it will not be serialized because static is the part of class but not part of an object.

```
package com.durga.mnrao.serde;

import java.io.Serializable;

public class Employee implements Serializable{

    private int empNum;
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
static private String empName;
private String empJob;
private double empSalary;
private String empDeptName;
private String empGender;
static 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 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;
}

public String getEmpJob()
```

```
        return empJob;
    }

    public void setEmpJob(String empJob) {
        this.empJob = empJob;
    }

}
```

Java Serialization with array or collection

Rule:

In case of array or collection, all the objects of array or collection must be serializable. If any object is not serializable, serialization will fail.

Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

```
package com.durga.mnrao.serde;

import java.io.Serializable;

public class Employee implements Serializable{

    private int empNum;

    private String empName;

    private String empJob;

    transient private double empSalary;

    private String empDeptName;

    transient private String empGender;

    static 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 double getEmpSalary() {
        return empSalary;
    }
}
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
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;
}

public String getEmpJob() {
    return empJob;
}

public void setEmpJob(String empJob) {
    this.empJob = empJob;
}

}
```

Reflection API

Java Reflection is a process of examining or modifying the run time behavior of a class at run time.

The `java.lang.Class` class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

Metadata: Data about data (information about other information)

The `java.lang` and `java.lang.reflect` packages provide classes for java reflection.

Where it is used

The Reflection API is mainly used in:

IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
Debugger, Test Tools etc.

Commonly used methods of `Class` class:

Method Description

1) public String getName()

returns the class name

2) public static Class forName(String className) throws ClassNotFoundException

loads the class and returns the reference of `Class` class.

3) public Object newInstance() throws InstantiationException, IllegalAccessException

creates new instance.

4) public boolean isInterface()

checks if it is interface.

5) public boolean isArray()

checks if it is array.

6) public boolean isPrimitive()

checks if it is primitive.

7) public Class getSuperclass()

returns the superclass class reference.

8) public Field[] getDeclaredFields() throws SecurityException

returns the total number of fields of this class.

9) public Method[] getDeclaredMethods() throws SecurityException

returns the total number of methods of this class.

10) public Constructor[] getDeclaredConstructors()throws SecurityException
returns the total number of constructors of this class.

**11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws
NoSuchMethodException,SecurityException**
returns the method class instance.

How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

forName() method of Class class
getClass() method of Object class

1) forName() method of Class class

is used to load the class dynamically.
returns the instance of Class class.
It should be used if you know the fully qualified name of class.
This cannot be used for primitive types only for the classes.

Let's see the simple example of forName() method.

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

public class Employee {

    private int empNum;
    private String empName;
    private double empSalary;
    private String empDept;

    public Employee()
    {

    }

    public Employee(int empNum,String empName,double empSalary,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 getEmpSalary() {
        return empSalary;
    }
    public void setEmpSalary(double empSalary) {
        this.empSalary = empSalary;
    }
    public String getEmpDept() {
        return empDept;
    }
    public void setEmpDept(String empDept) {
        this.empDept = empDept;
    }
    @Override
    public String toString() {
        return "Employee [empNum=" + empNum + ", empName=" + empName + ", empSalary=" + empSalary + ", empDept=" +
               + empDept + "]";
    }
}
```

```
package com.nrit.mnrao.sample;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Test {

    public static void main(String[] args) {

        try {
            Class c = Class.forName("com.nrit.mnrao.test.Employee");

            System.out.println("Successfully loaded");

            System.out.println("class name : "+c.getName());

            Constructor [] conNames = c.getConstructors();

            System.out.println("It provides following constructors");

            for (Constructor conName : conNames) {

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

}

```
Field [] fieldNames = c.getDeclaredFields();
System.out.println("It provides following Instance Variables");

for (Field fieldName : fieldNames) {
    System.out.println(fieldName);
}

Method [] methodNames = c.getDeclaredMethods();
System.out.println("It provides following Methods");
for (Method methodName : methodNames) {

    System.out.println(methodName);

}

Class superclass = c.getSuperclass();
System.out.println("superclass : "+superclass);

}

catch(ClassNotFoundException e)
{
    e.printStackTrace();
}

}
```

Creating object by the user in place of Class.forName()

```
package com.nrit.mnrao.sample;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

import com.nrit.mnrao.test.Employee;

public class Test {

    public static void main(String[] args) {

        try
        {
            Employee emp = new Employee();
        }
    }
}
```

```
Class<? extends Employee> c = emp.getClass();

System.out.println("class name : "+c.getName());

Constructor [] conNames = c.getConstructors();

System.out.println("It provides following constructors");

for (Constructor conName : conNames) {

    System.out.println(conName);

}

Field [] fieldNames = c.getDeclaredFields();

System.out.println("It provides following Data members");

for (Field fieldName : fieldNames) {
    System.out.println(fieldName);
}

Method [] methodNames = c.getDeclaredMethods();
System.out.println("It provides following Methods");
for (Method methodName : methodNames) {

    System.out.println(methodName);

}

Class superclass = c.getSuperclass();

System.out.println("superclass : "+superclass);

}

catch(Exception e)
{
    e.printStackTrace();
}

}
```

Multi Threading

Mutli Tasking:

It is a running multiple jobs simultaneously.

Multi tasking can be in two ways.

- 1) Process based Multi Tasking
- 2) Thread based Muti Tasking

Process based Multi Tasking:

In this for every job separate process creates and loads the job into the process.

These processes are created bt Operating System.

OS has to manage all the processes.

If jobs are java program, then JVM loads into every process to execute those programs.

Two JVMs can not share the data. Every one runs in its processes.

Two JVM can not communicate each other.

RMI is the way to communicate between Two JVMs.

For example there are four jobs. Then OS creates four childs and loads four JVMs into that processes.

Processs based job scheduling can be

- 1) Roud robin scheduling (Cirucluar)
- 2) Priority based scheduling (four ground job will be given more priority and next is its back ground

Advantage:

1.Process based is a safe one as no chance of crashing.

2.No chance of corrupting data as every process has it's own memory space.

Dis-Advantage :

1.OS has to manage many processes. Hence heavy load on the OS. It is an heavy weight process.

2.Performance is less.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Eg:

UNIX OS

Thread based Muti Tasking:

In this concept, only one child process creates for any no of threads and all threads will share same memory space.

Advantage:

It is light weight technology and more performance.

Dis-Advantage

Chance of crashing threads.

Developer should expert in writing Multi thread coding.

Thread based Muti Tasking can be a round robin or priority based.

For process based recommended one is round robin based

For thread based recommended one priority based.

Threading:

Thread is a part of an application.

Thread is an independent path of execution.

Def:

Execution of different parts of same application at the same time is called as multi-threading.

For any no of threads only one child process will be created and that can be shared by all threads of application.

Adv:

maximum utilization of CPU Time, so that we can get more performance.

What is a Thread ?

Ans:

it is an Independent Path of execution.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

it is a part of application.

What is Multi Threading ?

Ans:

Multiple Parts of Same application running in parallel is called as Multi Threading

What is purpose of Multi Threading?

ans:

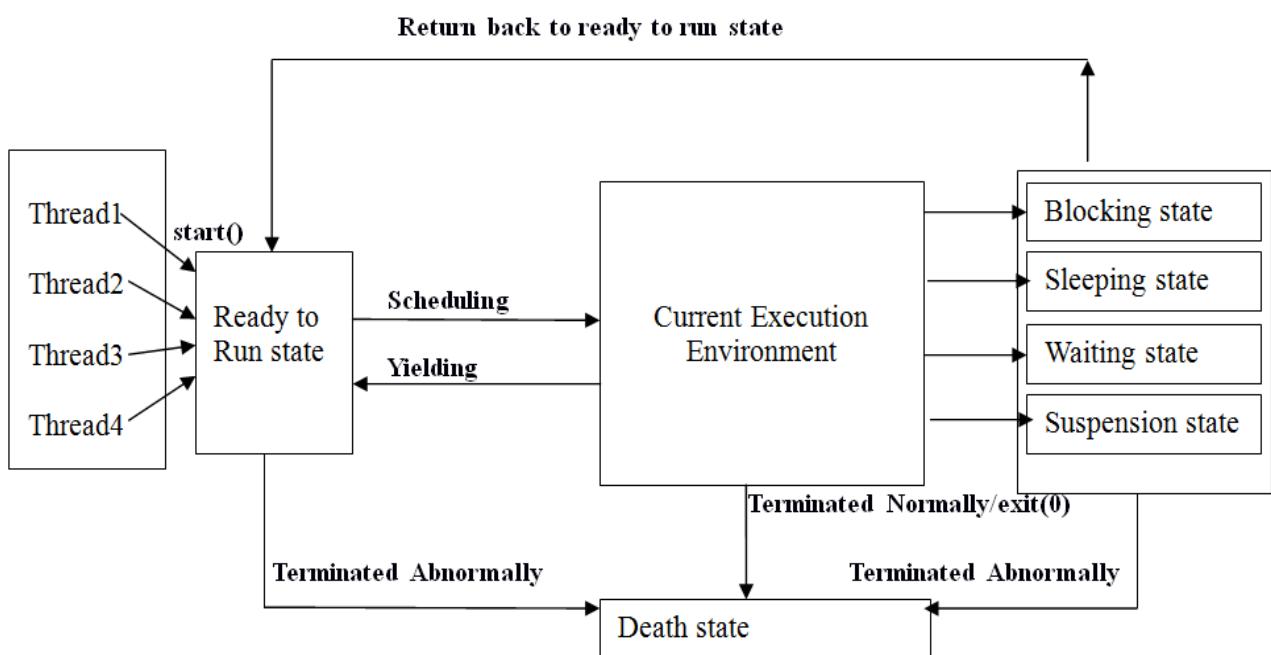
to Utilize Maximum CPU Time.

What is Advantage of Multi Threading.

Ans:

to Improve the performance of application.

Different states of thread:



JVM provides **Thread Scheduler**, which manages different states of the thread.

1) Ready to run state:

It is a state of waiting for CPU time. A Thread comes to ready to run state, When start() method is called on thread.

2) Current execution state :

It is a state of assigning CPU time to thread

3) Blocking state:

Thread goes to blocking state when performing I/O operations with external resources, such as keyboard, database server and other servers.

4) Sleeping state:

It is a state of waiting for shared resources for specific amount of time. A thread goes to sleeping state when sleep() is called on thread.

5) waiting state

It is state of waiting for the shared resources until getting the resources. Thread goes to waiting state when wait() method called on thread.

6) suspend state:

If any child thread is going to perform any illegal transactions on the system resources, then main thread will suspend the child thread. If any thread is in suspend state it should be revoked by some other thread.

7) Death state:

A thread goes to death state, when terminated normally or abnormally (or) called exit(0) method on thread.

Once thread goes to death state it can not be invoked.

8) Scheduling : thread scheduler schedules the threads for execution. Thread scheduling can be done any one of the two ways.

- 1) priority scheduling (or) primitive scheduling
- 2) Round robin process.

9) yielding :

If any thread enters into ready to run state (new or old thread) thread scheduler checks priority of current execution thread with threads which are waiting for CPU time. If waiting thread is having more priority than the current execution thread, then thread scheduler forcibly brings the current execution thread into ready to run state.

Creation of a thread :

A thread can be created in two ways:

- 1) by extending from Thread class
- 2) by implementing Runnable interface

Recommended one is implements Runnable.

Thread class and Runnable interfaces are from java.lang package.

Runnable interface;

```
public interface Runnable {  
    public void run();  
}
```

run() should be implemented in the child class, which is acting as thread. run() contains thread coding.

Thread class:

This class implements the Runnable interface.

```
public class Thread implements Runnable  
{  
}  
}
```

Constructors of Thread class:

- 1) Thread()

Thread t = new Thread (); → default name of the thread is “child”

- 2) Thread(String name)

Thread t = new Thread (“chid_name”);

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

3) Thread(Runnable r, String name)

```
Thread t = new Thread(this, "child_name");
```

Methods of Thread class:

- 1) String name = t.getName() returns name of the thread
- 2) String parent = t.getParent(); return parent name of the thread.
- 3) t.start() to take thread into ready to run state.
- 4) public static void sleep(int milliSeconds) to take thread into sleeping state.

1 sec = 1000 milli sec

Creation of a thread by extending from Thread class.

```
public class MyThread extends Thread
{
    public MyThread()
    {
        super("child");
    }

    public void run()
    {
        System.out.println ("Child thread started");
        try
        {
            for(int i=0;i<5;i++)
            {
                System.out.println ("child : "+i);
                Thread.sleep(200);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println ("Child Interrupted");
        }
        System.out.println ("end of child thread");
    }
}
```

```
public class SingleThreadTest
{
```

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

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
{  
    System.out.println ("parent started");  
  
    Thread t = new MyThread();  
  
    System.out.println ("child created");  
  
    t.start();  
  
    try  
    {  
        for (int i = 0; i < 5; i++)  
        {  
            System.out.println ("Parent : " + i);  
            Thread.sleep(200);  
        }  
    }  
    catch (InterruptedException e)  
    {  
        System.out.println ("Parent Interrupted");  
    }  
  
    System.out.println ("end of Parent thread");  
  
}  
  
}  
  
o/p:  
parent started  
child created  
Parent : 0  
Child thread started  
child : 0  
Parent : 1  
child : 1  
Parent : 2  
child : 2  
Parent : 3  
child : 3  
Parent : 4  
child : 4  
end of Parent thread  
end of child thread
```

Creation of thread by implementing Runnable interface.

```
public class MyThread implements Runnable  
{  
    Thread t;  
  
    public MyThread()
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
{  
    t=new Thread(this,"child");  
    t.start();  
}  
  
public void run()  
{  
    System.out.println ("Child thread started");  
    try  
    {  
        for(int i=0;i<5;i++)  
        {  
            System.out.println ("child : "+i);  
            Thread.sleep(200);  
        }  
    }  
    catch(InterruptedException e)  
    {  
        System.out.println ("Child Interrupted");  
    }  
  
    System.out.println ("end of child thread");  
}  
}  
  
public class ThreadTest  
{  
    public static void main(String[] args)  
    {  
        System.out.println ("parent started");  
  
        new MyThread();  
  
        try  
        {  
            for (int i = 0; i < 5; i++)  
            {  
                System.out.println ("Parent : " + i);  
                Thread.sleep(200);  
            }  
        }  
        catch (InterruptedException e)  
        {  
            System.out.println ("Parent Interrupted");  
        }  
  
        System.out.println ("end of Parent thread");  
    }  
}
```

o/p:
parent started

```
child created  
Parent : 0  
Child thread started  
child : 0  
Parent : 1  
child : 1  
Parent : 2  
child : 2  
Parent : 3  
child : 3  
Parent : 4  
child : 4
```

end of child thread

end of Parent thread

Creating Multiple threads:

```
public class MyThread implements Runnable  
{  
    Thread t;  
  
    public MyThread(String name)  
    {  
        t=new Thread(this,name);  
        t.start();  
    }  
  
    public void run()  
    {  
        String name = t.getName();  
  
        System.out.println ("Child Name : "+name);  
  
        try  
        {  
            for(int i=0;i<5;i++)  
            {  
                System.out.println (name+" : "+i);  
  
                Thread.sleep(200);  
            }  
        }  
        catch(InterruptedException e)  
        {  
            System.out.println ("Child Interrupted name : "+name);  
        }  
  
        System.out.println ("end of child thread name : "+name);  
    }  
}
```

}

```
public class ThreadTest
{
    public static void main(String[] args)
    {
        System.out.println ("parent started");

        String [] threadNames = {"one","two","three","four","five"};

        try
        {
            for (int i = 0; i < threadNames.length; i++)
            {
                new MyThread(threadNames[i]);
            }

            Thread.sleep(8000);
        }
        catch (InterruptedException e)
        {
            System.out.println ("Parent Interrupted");
        }

        System.out.println ("end of Parent thread");
    }
}
```

Thread priority:

Priority is Ranging from 1 to 10

Thread.MAX_PRIORITY=10;

Thread.MIN_PRIORITY = 1;

Thread.NORM_PRIORITY = 5; → it is a default priority.

Methods

public void setPriority(int priority) : to set priority of the thread.

public int getPriority() : to get priority of the thread.

```
public class MyThread implements Runnable
{
    Thread t;
    public MyThread(String name, int priority)
    {
        t=new Thread(this, name);
        t.setPriority(priority);
        t.start();
    }

    public void run()
    {
        System.out.println ("Child Name : "+t.getName());

        System.out.println ("Priority : "+t.getPriority());

        System.out.println ("End ");
    }
}
```

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

        String []
threadNames={"One","Two","Three","Four","Five","Six","Seven","eight","nine"};

        Thread currentThread = Thread.currentThread();

        currentThread.setPriority(10);

        for (int i = 0; i < threadNames.length; i++)
        {
            new MyThread(threadNames[i], i+1);
        }

        System.out.println ("Childs are created");
        System.out.println ("End of parent");
    }
}
```

In the above example parent thread dies first and then child . Hence child threads becomes orphaned threads.

In this scenario we can use join() to wait for the child threads to complete.

Methods:

- 1) public boolean isAlive(); return true if thread is alive otherwise false.
- 2) public void join(); to wait for the child thread.

```
public class MyThread implements Runnable  
{
```

```
    Thread t;
```

```
    public MyThread(String name)  
{
```

```
        t = new Thread(this, name);
```

```
        t.start();
```

```
}
```

```
    public void run()  
{
```

```
        System.out.println ("Child Name : "+t.getName());
```

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

```
}
```

```
}
```

```
public class ThreadTest  
{
```

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

```
        MyThread t1 = new MyThread("One");
```

```
        MyThread t2 = new MyThread("Two");
```

```
        MyThread t3 = new MyThread("Three");
```

```
        System.out.println ("One is in Alive "+t1.t.isAlive());
```

```
        System.out.println ("Two is in Alive "+t2.t.isAlive());
```

```
        System.out.println ("Three is in Alive "+t3.t.isAlive());
```

```
try
{
    t1.t.join();
    t2.t.join();
    t3.t.join();
}

catch (InterruptedException e)
{
    System.out.println (e.getMessage());
}

System.out.println ("One is in Alive "+t1.t.isAlive());
System.out.println ("Two is in Alive "+t2.t.isAlive());
System.out.println ("Three is in Alive "+t3.t.isAlive());

System.out.println ("End of parent");
}

}
```

Thread synchronization:

When Multiple threads are trying to access shared resources, there is a chance of crashing threads. To avoid thread crashing, developer has to write synchronization coding.

Methods to synchronize a thread.

- 1) public void wait()
this method takes the thread into waiting state. It throws InterruptedException.
- 2) public void notify()
this method is to invoke the first waiting thread in the waiting queue.
- 3) public void notifyAll()

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

this method is to invoke all waiting threads, in waiting queue

the above three methods are from Object class, not from thread class. Hence these can be called directly.

```
public class Queue
{
    int data=0;
    boolean state = false;

    public synchronized void pop()
    {
        if( ! state )
        {
            try
            {
                wait();
            }
        }
    }
}
```

```
        }
        catch(InterruptedException e)
        {
            System.out.println ("Consumer Interrupted");
        }
    }

    System.out.println ("consumed Data "+data);

    state=false;
    notify();
}

public synchronized void push(int num)
{
    if(state)
    {
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            System.out.println ("Consumer Interrupted");
        }
    }
    data=num;

    System.out.println ("Produced Data "+data);

    state=true;
    notify();
}

public class Producer implements Runnable
{
    Thread t;
    Queue q1;

    public Producer(Queue q)
    {
        q1=q;

        t=new Thread(this,"producer");

        t.start();
    }
}
```

```
public void run()
{
    System.out.println ("Producer started");

    int i=0;

    while(true)
    {
        i++;
        q1.push(i);
    }
}

public class Consumer implements Runnable
{
    Thread t;
    Queue q1;
    public Consumer(Queue q)
    {
        q1=q;
        t=new Thread(this,"consumer");
        t.start();
    }

    public void run()
    {
        System.out.println ("consumer started");
        while(true)
        {
            q1.pop();
        }
    }
}

public class ProdConsumer
{
    public static void main(String[] args)
    {
        Queue q = new Queue();

        Consumer c1 = new Consumer(q);

        Producer p1 = new Producer(q);

        System.out.println ("Press Ctrl + C for End ");

    }
}
```

Interview Questions

1) What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Its main advantage is:

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between process is low.

2) What is thread?

A thread is a lightweight subprocess. It is a separate path of execution. It is called separate path of execution because each thread runs in a separate stack frame.

3) What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

4) What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

5) What is difference between wait() and sleep() method?

wait()

1) The wait() method is defined in Object class.

2) wait() method releases the lock.

sleep()

The sleep() method is defined in Thread class.

The sleep() method doesn't releases the lock.

6) Is it possible to start a thread twice?

No, there is no possibility to start a thread twice. If we do, it throws an exception.

7) Can we call the run() method instead of start()?

yes, but it will not work as a thread rather it will work as a normal object so there will not be context-switching between the threads.

8) What about the daemon threads?

The daemon threads are basically the low priority threads that provides the background support to the user threads. It provides services to the user threads.

9)Can we make the user thread as daemon thread if thread is started?

No, if you do so, it will throw `IllegalThreadStateException`

10)What is shutdown hook?

The shutdown hook is basically a thread i.e. invoked implicitly before JVM shuts down. So we can use it perform clean up resource

11)When should we interrupt a thread?

We should interrupt a thread if we want to break out the sleep or wait state of a thread.

12) What is synchronization?

Synchronization is the capability of control the access of multiple threads to any shared resource. It is used:

1. To prevent thread interference.
2. To prevent consistency problem.

13) What is the purpose of Synchronized block?

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

14)Can Java object be locked down for exclusive use by a given thread?

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

15) What is static synchronization?

If you make any static method as synchronized, the lock will be on the class not on object.

16)What is the difference between `notify()` and `notifyAll()`?

The `notify()` is used to unblock one waiting thread whereas `notifyAll()` method is used to unblock all the threads in waiting state.

17)What is deadlock?

Deadlock is a situation when two threads are waiting on each other to release a resource. Each thread waiting for a resource which is held by the other waiting thread.

Networking

Java Networking Terminology

The widely used java networking terminologies are given below:

Network :

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

It is a physical connections between the computer. It can be established by using hub / switch and Cables.

Networking :

It is communication system between two computers in the Network.

LAN (Local Area Network)

It is a short distance (in the same Data Centre) , it can achieved by using HUB or SWITCH.

WAN (Wide Area Network)

For long distance (between two Data Centres located at different location of same building) . it can be through the switches or routers.

MAN (Metropolitan Area Network)

For very larger distance between different location of city. It can be through router.

Eg:

WIFI connection.

Intranet :

Network of similar networks. It can be through **bridge** .

Internet :

Network of dissimilar type of Networks. It can be through **Gateway**.

To achieve networking Hardware and Software components are required.

Hardware components:

- 1) NIC (Network Interface Card)
- 2) Media of communication.

NIC will maintain the IP address of the system.

The choice of media of communication depends on the following factors.

- 1) Amount of distance
- 2) Amount of data transferring
- 3) How frequently data is transferring.

Topology:

It defines how physical connections have been made between computers.

- 1) Ring Topology 2) Bus Topology 3) Star

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Mostly used one is physical Bus and logical star Topology

Architecture:

If defines how communication would be done between computers.

- 1) Point – to – point communication
- 2) Broad casting

Point – to – point communication:

In this communication first routing established between source and destination, then communication starts.

It is a connection oriented (or) TCP based communication

Eg: telephone

Broad casting:

In this communication data will be delivered to all systems in the network and the system will respond to which data intended.

Eg: radio

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of four octets that ranging from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP

- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communication.

Difference between TCP & UDP:

S.No	TCP	UDP
1.	Transmission Control Protocol	User Datagram Protocol
2.	Connection Oriented	Connectionless
3.	Byte stream protocol	Packet stream protocol
4.	TCP expects for the acknowledgement	Never expects for acknowledgement
5.	There is a guarantee for destination data	No guarantee for destination of data
6.	Point – to – point communication	Broad casting

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Methods used Network Programming:

public InputStream getInputStream() : returns the InputStream attached with this socket.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

public OutputStream getOutputStream(): returns the OutputStream attached with this socket.

public synchronized void close(): closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients

1) public Socket accept():

returns the socket and establish a connection between server and client.

2) public synchronized void close():

closes the server socket.

TCP Echo Server:

```
package com.durga.mnrao.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class TcpEchoServer {

    public static void main(String[] args) {

        ServerSocket server = null;
        DataInputStream dis = null;
        DataOutputStream dos = null;

        try
        {
```

```
server = new ServerSocket(4321);

System.out.println("Waiting for client request");

Socket socket = server.accept();

System.out.println("Received client request");

dis = new DataInputStream(socket.getInputStream()) ;

dos = new DataOutputStream(socket.getOutputStream()) ;

String msg = "";

while(true)
{

    msg = dis.readLine();

    if(msg.equals("exit"))
    {
        break;
    }

    System.out.println("Message from client = "+msg);

    dos.writeBytes(msg+"\n");
}

dos.close();
dis.close();
socket.close();
server.close();

}

catch(Exception e)
{
    e.printStackTrace();
}
}

}
```

TCP ECHO Client:

```
package com.durga.mnrao.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;

public class TcpEchoClient {

    public static void main(String[] args) {

        Socket clientSocket =null;
        DataInputStream dis = null;
```

```
DataStream dos = null;
DataInputStream kb = null;

try
{
    clientSocket = new Socket("localhost",4321);

    dis = new DataInputStream(clientSocket.getInputStream());
    dos = new DataOutputStream(clientSocket.getOutputStream());

    kb = new DataInputStream(System.in);

    while(true)
    {
        System.out.println("Enter your message : ");

        String input = kb.readLine();
        dos.writeBytes(input+"\n");

        if(input.equals("exit"))
        {
            break;
        }

        String reply = dis.readLine();
        System.out.println("Reply = "+reply);

    }

    dos.close();
    dis.close();
    kb.close();
    clientSocket.close();

}
catch(Exception e)
{
    e.printStackTrace();
}

}
```

Chatting Application :

Server :

```
package com.durga.mnrao.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class TcpChatServer {  
  
    public static void main(String[] args) {  
  
        ServerSocket server = null;  
  
        DataInputStream dis = null;  
  
        DataOutputStream dos = null;  
  
        DataInputStream kb = null;  
  
        try  
        {  
  
            server = new ServerSocket(4321);  
  
            System.out.println("Waiting for client request");  
  
            Socket socket = server.accept();  
  
            System.out.println("Received client request");  
  
            dis = new DataInputStream(socket.getInputStream());  
  
            dos = new DataOutputStream(socket.getOutputStream());  
  
            kb = new DataInputStream(System.in);  
  
            String msg = "";  
  
            while(true)  
            {  
  
                msg = dis.readLine();  
  
                System.out.println("from client : "+msg);  
  
                if(msg.equals("no"))  
                {  
                    break;  
                }  
  
                System.out.println("Your Reply : ");  
  
                msg = kb.readLine();  
  
                dos.writeBytes(msg+"\n");  
            }  
  
            dos.close();  
            dis.close();  
            socket.close();  
            server.close();  
  
        }  
        catch(Exception e)  
        {  
            e.printStackTrace();  
        }  
    }  
}
```

}

Client :

```
package com.durga.mnrao.net;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;

public class TcpChatClient {

    public static void main(String[] args) {

        Socket clientSocket = null;
        DataInputStream dis = null;
        DataOutputStream dos = null;
        DataInputStream kb = null;

        try
        {
            clientSocket = new Socket("localhost",4321);

            dis = new DataInputStream(clientSocket.getInputStream());
            dos = new DataOutputStream(clientSocket.getOutputStream());

            kb = new DataInputStream(System.in);

            while(true)
            {
                System.out.println("Enter your message : ");

                String input = kb.readLine();
                dos.writeBytes(input+"\n");

                if(input.equals("no"))
                {
                    break;
                }

                String reply = dis.readLine();
                System.out.println("Reply from Server : "+reply);

            }

            dos.close();
            dis.close();
            kb.close();
            clientSocket.close();
        }
    }
}
```

```
        }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

UDP communication:

Java DatagramSocket and DatagramPacket

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Constructors of DatagramSocket class

- **DatagramSocket() throws SocketException:** it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port) throws SocketException:** it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address) throws SocketException:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port):** it creates a datagram packet. This constructor is used to send the packets.

Server:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Date;

public class UDPServer {

    public static void main(String[] args) {
        try
        {
            DatagramSocket ds = new DatagramSocket();

            InetAddress localHost = InetAddress.getLocalHost();
            while(true)
            {
                Date currentDate = new Date();

                String str = currentDate.toString();
            }
        }
    }
}
```

```
byte [] b = str.getBytes();

DatagramPacket dp = new DatagramPacket(b

,b.length,localhost,2341);

ds.send(dp);

Thread.sleep(5000);

}

catch(Exception e)
{
    e.printStackTrace();
}

}

}
```

Client:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPClient {

    public static void main(String[] args) {
        try
        {
            DatagramSocket ds = new DatagramSocket(2341);

            while(true)
            {
                byte [] b = new byte[1024];

                DatagramPacket dp = new DatagramPacket(b,b.length);

                ds.receive(dp);
            }
        }
    }
}
```

```
b = dp.getData();

String str = new String(b);

System.out.println ("Server Date = "+ str);

Thread.sleep(4000);

}

}

catch(Exception e)
{
    e.printStackTrace();
}

}

}
```

Server Sending data to Multiple clients:

```
package com.durga.mnrao.net;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Date;

public class UDPMultiplePacketsServer {

    public static void main(String[] args) {

        try
        {
            DatagramSocket ds = new DatagramSocket();

            InetAddress localHost = InetAddress.getLocalHost();

            int count = 0;

            while(true)
            {
                Date currentDate = new Date();

                String str = currentDate.toString();

                DatagramPacket dp =null;

                if(count%2==0)


```

```
        {
            str = "even packet --> " + str;

            byte [] b = str.getBytes();

            dp = new DatagramPacket( b ,b.length,localhost,2342);

        }
    else
    {
        str = "odd packet --> " + str;

        byte [] b = str.getBytes();

        dp = new DatagramPacket( b ,b.length,localhost,2341);

    }

    ds.send(dp);

    count++;

    Thread.sleep(5000);

}

catch(Exception e)
{
    e.printStackTrace();
}
}

}
```

}

Client1:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPClient1 {

    public static void main(String[] args) {
        try
        {
            DatagramSocket ds = new DatagramSocket(2341);

            while(true)
            {
                byte [] b = new byte[1000];

                DatagramPacket dp = new DatagramPacket(b,b.length);

                ds.receive(dp);
            }
        }
    }
}
```

```
b = dp.getData();

String str = new String(b);

System.out.println ("Server Date = "+ str);

Thread.sleep(4000);

}

}

catch(Exception e)
{
    e.printStackTrace();
}

}

}
```

Client2:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPClient2 {

    public static void main(String[] args) {
        try
        {
            DatagramSocket ds = new DatagramSocket(2342);

            while(true)
            {
                byte [] b = new byte[1000];

                DatagramPacket dp = new DatagramPacket(b,b.length);

                ds.receive(dp);

                b = dp.getData();

                String str = new String(b);

                System.out.println ("Server Date = "+ str);

                Thread.sleep(4000);

            }

            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

```
        }  
    }  
}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.

For example:

<https://www.google.com:8020/index.jsp>

A URL contains many information:

1. **Protocol:** In this case, https is the protocol.
2. **Server name or IP Address:** In this case, www.google.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http// www.google.com:8020/ index.jsp /, 8020 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

java.net.URL class provides following methods

- 1) public String getProtocol() : it returns the protocol of the URL.
- 2) public String getHost() : it returns the host name of the URL.
- 3) public String getPort() : it returns the Port Number of the URL.
- 4) public String getFile() : it returns the file name of the URL.
- 5) public URLConnection openConnection() : it returns the instance of URLConnection i.e. associated with this URL.

```
import java.net.URL;  
  
public class URLTest  
{  
    public static void main(String[] args)  
    {  
        try  
        {  
            URL url = new URL("https://www.google.com:8020/index.jsp");  
            System.out.println ("Protocol: "+url.getProtocol());  
            System.out.println ("Host Name: "+url.getHost());  
        }  
    }  
}
```

```
        System.out.println ("Port Number: "+url.getPort());  
  
        System.out.println ("File Name: "+url.getFile());  
  
    }  
    catch(Exception e)  
    {  
        System.out.println (e);  
    }  
}
```

Java InetAddress class

Java InetAddress class represents an IP address. The `java.net.InetAddress` class provides methods to get the IP of any host name *for example* `www.gmail.com`, `www.google.com`, `www.facebook.com` etc.

Methods of InetAddress class

1) `public static InetAddress getByName(String host) throws UnknownHostException`

It returns the instance of `InetAddress` containing LocalHost IP and name.

2) `public static InetAddress getLocalHost() throws UnknownHostException`

it returns the instance of `InetAddress` containing local host name and address.

3) `public String getHostName()`

it returns the host name of the IP address.

4) `public String getHostAddress()`

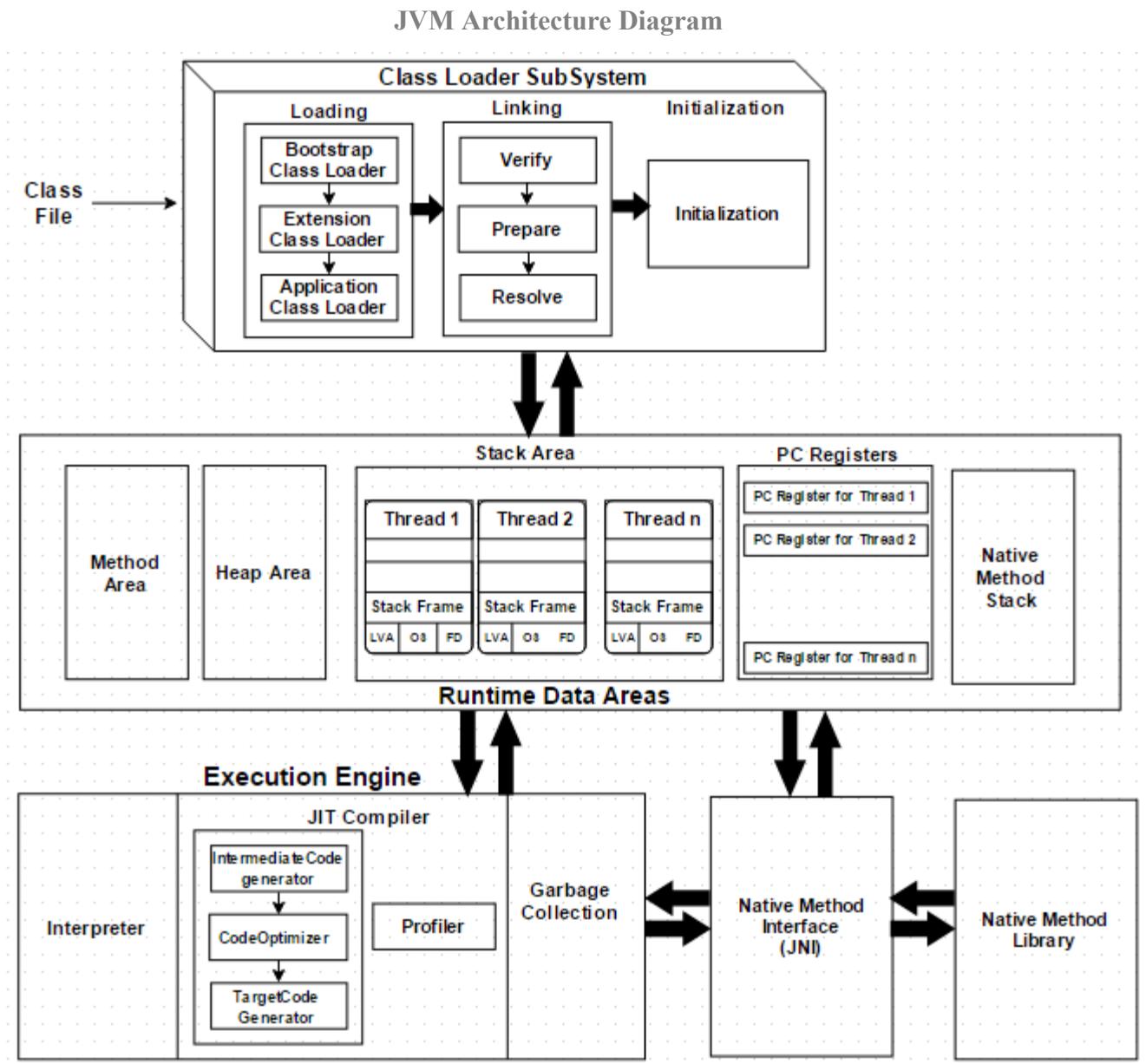
it returns the IP address in string format.

```
import java.net.InetAddress;  
  
public class InetTest {  
    public static void main(String[] args) {  
        try {  
            InetAddress ip = InetAddress.getByName("www.facebook.com");  
  
            System.out.println ("Host Name: " + ip.getHostName());  
            System.out.println ("IP Address: " + ip.getHostAddress());  
        } catch (Exception e) {  
            System.out.println (e);  
        }  
    }  
}
```

JVM Architecture

What is the JVM?

A **Virtual Machine** is a software implementation of a physical machine. Java was developed with the concept of **WORA (Write Once Run Anywhere)**, which runs on a **VM**. The **compiler** compiles the Java file into a Java **.class** file, then that **.class** file is input into the **JVM**, which Loads and executes the class file. Below is a diagram of the Architecture of the **JVM**.



How Does the JVM Work?

As shown in the above architecture diagram, the JVM is divided into three main subsystems:

- 1. Class Loader Subsystem**
- 2. Runtime Data Area**
- 3. Execution Engine**

1. Class Loader Subsystem

Java's **dynamic class loading** functionality is handled by the class loader subsystem. It loads, links, and initializes the class file when it refers to a class for the first time at **runtime**, not **compile time**.

1.1 Loading

Classes will be loaded by this component. Boot Strap class Loader, Extension class Loader, and Application class Loader are the three class loader which will help in achieving it.

1. **Boot Strap ClassLoader** – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar**. Highest priority will be given to this loader.
2. **Extension ClassLoader** – Responsible for loading classes which are inside **ext** folder (**jre\lib**).
3. **Application ClassLoader** – Responsible for loading **Application Level Classpath**, path mentioned Environment Variable etc.

The above **Class Loaders** will follow **Delegation Hierarchy Algorithm** while loading the class files.

1.2 Linking

1. **Verify** – Bytecode verifier will verify whether the generated bytecode is proper or not if verification fails we will get the **verification error**.
2. **Prepare** – For all static variables memory will be allocated and assigned with **default values**.
3. **Resolve** – All **symbolic memory references** are replaced with the **original references** from **Method Area**.

1.3 Initialization

This is the final phase of Class Loading, here all **static variables** will be assigned with the original values, and the **static block** will be executed.

2. Runtime Data Area

The Runtime Data Area is divided into 5 major components:

1. **Method Area** – All the **class level data** will be stored here, including **static variables**. There is only one method area per JVM, and it is a shared resource.
2. **Heap Area** – All the **Objects** and their corresponding **instance variables** and **arrays** will be stored here. There is also one Heap Area per JVM. Since the **Method** and **Heap areas** share memory for multiple threads, the data stored is not thread **safe**.
3. **Stack Area** – For every thread, a separate **runtime stack** will be created. For every **method call**, one entry will be made in the stack memory which is called as **Stack Frame**. All **local variables** will be created in the stack memory. The stack area is thread safe since it is not a shared resource. The Stack Frame is divided into three subentities:
 - a) **Local Variable Array** – Related to the method how many **local variables** are involved and the corresponding values will be stored here.
 - b) **Operand stack** – If any intermediate operation is required to perform, **operand stack** acts as runtime workspace to perform the operation.
 - c) **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.
4. **PC Registers** – Each thread will have separate **PC Registers**, to hold the address of **current executing instruction** once the instruction is executed the PC register will be **updated** with the next instruction.
5. **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

3. Execution Engine

The bytecode which is assigned to the **Runtime Data Area** will be executed by the Execution Engine. The Execution Engine reads the bytecode and executes it piece by piece.

- 1) **Interpreter** – The interpreter interprets the bytecode faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
- 2) **JIT Compiler** – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
 - 1) **Intermediate Code generator** – Produces intermediate code
 - 2) **Code Optimizer** – Responsible for optimizing the intermediate code generated above
 - 3) **Target Code Generator** – Responsible for Generating Machine Code or Native Code
 - 4) **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.
- 3) **Garbage Collector**: Collects and removes unreferenced objects. Garbage Collection can be triggered by calling "*System.gc()*", but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface (JNI): JNI will be interacting with the **Native Method Libraries** and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: It is a collection of the Native Libraries which is required for the Execution Engine.

Interview Questions

1. What's the difference between an array and Vector?

Ans: An array groups data of same primitive type and is static in nature while vectors are dynamic in nature and can hold data of different data types.

2. What is multi-threading?

Ans: Multi threading is a programming concept to run multiple tasks in a concurrent manner within a single program. Threads share same process stack and running in parallel. It helps in performance improvement of any program.

3. How garbage collection is done in Java?

Ans: In java, when an object is not referenced any more, garbage collection takes place and the object is destroyed automatically. For automatic garbage collection java calls either System.gc() method or Runtime.gc() method.

4. How can we make copy of a java object?

Ans: We can use the concept of cloning to create copy of an object. Using clone, we create copies with the actual state of an object.

Clone() is a method of Cloneable interface and hence, Cloneable interface needs to be implemented for making object copies.

5 In java, how we can disallow serialization of variables?

Ans: If we want certain variables of a class not to be serialized, we can use the keyword transient while declaring them. For example, the variable trans_var below is a transient variable and can't be serialized:

6. Which types of exceptions are caught at compile time?

Ans: Checked exceptions can be caught at the time of program compilation. Checked exceptions must be handled by using try catch block in the code in order to successfully compile the code.

7.What is multi-catch block in java?

Java 7 one of the improvement was multi-catch block where we can catch multiple exceptions in a single catch block. This makes are code shorter and cleaner when every catch block has similar code.

If a catch block handles multiple exception, you can separate them using a pipe (|) and in this case exception parameter (ex) is final, so you can't change it.

```
catch(IOException | SQLException | Exception ex){
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
logger.error(ex);
throw new MyException(ex.getMessage());
}
```

8.What is Classloader in Java?

Java Classloader is the program that loads byte code program into memory when we want to access any class. We can create our own classloader by extending ClassLoader class and overriding loadClass(String name) method.

9.Can we have try without catch block?

Yes, we can have try-finally statement and hence avoiding catch block.

10.What is Garbage Collection?

Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, process of deallocating memory is handled automatically by the garbage collector.

We can run the garbage collector with code Runtime.getRuntime().gc() or use utility method System.gc().

11.What is Serialization and Deserialization?

We can convert a Java object to an Stream that is called Serialization. Once an object is converted to Stream, it can be saved to file or send over the network or used in socket connections.

The object should implement Serializable interface and we can use java.io.ObjectOutputStream to write object to file or to any OutputStream object.

12.How to run a JAR file through command prompt?

We can run a jar file using java command but it requires Main-Class entry in jar manifest file. Main-Class is the entry point of the jar and used by java command to execute the class.

13.What is a transient variable?

A transient variable is a variable that may not be serialized during Serialization and which is initialized by its default value during de-serialization,

14.What is the difference between error, exception, bug and defect ?

Error :- It is a compile time one.

Exception :- It is a runtime one, 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.

15.What are the advantages of ArrayList over arrays?

ArrayList can grow dynamically and provides more powerful insertion and search mechanisms than arrays.

16.What is daemon thread?

Daemon thread is a low priority thread, which runs intermittently in the background doing the garbage collection operation for the java runtime system.

17.How do you prevent a method from being overridden?

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

17.Difference between ArrayList and Vector ?

ArrayList	Vector
ArrayList is NOT synchronized by default.	Vector List is synchronized by default.
ArrayList can use only Iterator to access the elements.	Vector list can use Iterator and Enumeration Interface to access the elements.
The ArrayList increases its array size by 50 percent if it runs out of room.	A Vector defaults to doubling the size of its array if it runs out of room
ArrayList has no default size.	While vector has a default size of 10.

19.Difference between HashMap and Hashtable ?

HashMap	Hashtable
HashMap lets you have null values as well as one null key.	HashTable does not allow null values as key and value.
The iterator in the HashMap is fail-safe (If you change the map while iterating, you'll know).	The enumerator for the Hashtable is not fail-safe.
HashMap is unsynchronized.	Hashtable is synchronized.

Note: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

Some more Questions and Answers.

20) Why Java doesn't support multiple inheritance?

Java doesn't support multiple inheritance in classes because of "Diamond Problem". However multiple inheritance is supported in interfaces. An interface can extend multiple interfaces because they just declare the methods and implementation will be present in the implementing class. So there is no issue of diamond problem with interfaces.

21) What is overloading and overriding in java?

When we have more than one method with same name in a single class but the arguments are different, then it is called as method overloading.

Overriding concept comes in picture with inheritance when we have two methods with same signature, one in parent class and another in child class. We can use @Override annotation in the child class overridden method to make sure if parent class method is changed, so as child class.

22) Can we overload main method?

Yes, we can have multiple methods with name “main” in a single class. However if we run the class, java runtime environment will look for main method with syntax as public static void main(String [] args).

23) Can we have multiple public classes in a java source file?

We can't have more than one public class in a single java source file. A single source file can have multiple classes that are not public.

24) What is Java Package and which package is imported by default?

Java package is the mechanism to organize the java classes by grouping them. The grouping logic can be based on functionality or modules based. A java class fully classified name contains package and class name. For example, java.lang.Object is the fully classified name of Object class that is part of java.lang package.

java.lang package is imported by default and we don't need to import any class from this package explicitly.

25) What is final keyword?

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can't extend it.

We can use final keyword with methods to make sure child classes can't override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

Java interface variables are by default final and static.

26) What is static keyword?

static keyword can be used with class level variables to make it global i.e all the objects will share the same variable.

static keyword can be used with methods also. A static method can access only static variables of class and invoke only static methods of the class.

27) What is finally and finalize in java?

finally block is used with try-catch to put the code that you want to get executed always, even if any exception is thrown by the try-catch block. finally block is mostly used to release resources created in the try block.

finalize() is a special method in Object class that we can override in our classes. This method gets called by garbage collector when the object is getting garbage collected. This method is usually overridden to release system resources when object is garbage collected.

28) What is try-with-resources in java?

One of the Java 7 features is try-with-resources statement for automatic resource management. Before Java 7, there was no auto resource management and we should explicitly close the resource. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks when we forgot to close the resource.

From Java 7, we can create resources inside try block and use it. Java takes care of closing it as soon as try-catch block gets finished.

29) What is multi-catch block in java?

Java 7 one of the improvement was multi-catch block where we can catch multiple exceptions in a single catch block. This makes are code shorter and cleaner when every catch block has similar code.

If a catch block handles multiple exception, you can separate them using a pipe (|) and in this case exception parameter (ex) is final, so you can't change it.

30) What is static block?

Java static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader. It is used to initialize static variables of the class. Mostly it's used to create static resources when class is loaded.

31) What is an interface?

Interfaces are core part of java programming language and used a lot not only in JDK but also java design patterns, most of the frameworks and tools. Interfaces provide a way to achieve abstraction in java and used to define the contract for the subclasses to implement.

32) What is an abstract class?

Abstract classes are used in java to create a class with some default method implementation for subclasses. An abstract class can have abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class. Abstract classes can't be instantiated and mostly used to provide base for sub-classes to extend and implement the abstract methods and override or use the implemented methods in abstract class.

33) What is the difference between abstract class and interface?

abstract keyword is used to create abstract class whereas interface is the keyword for interfaces.

Abstract classes can have method implementations whereas interfaces can't.

A class can extend only one abstract class but it can implement multiple interfaces.

We can run abstract class if it has main() method whereas we can't run an interface.

34) Can an interface implement or extend another interface?

Interfaces don't implement another interface, they extend it. Since interfaces can't have method implementations, there is no issue of diamond problem. That's why we have multiple inheritance in interfaces i.e an interface can extend multiple interfaces.

35) What are Wrapper classes?

Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing and unboxing allows easy conversion between primitive types and their corresponding wrapper classes.

36) What is Enum in Java?

Enum was introduced in Java 1.5 as a new type whose fields consists of fixed set of constants. For example, in Java we can create Direction as enum with fixed fields as EAST, WEST, NORTH, SOUTH.

enum is the keyword to create an enum type and similar to class. Enum constants are implicitly static and final.

37) What is Java Annotations?

Java Annotations provide information about the code and they have no direct effect on the code they annotate. Annotations are introduced in Java 5. Annotation is metadata about the program embedded in the program itself. It can be parsed by the annotation parsing tool or by compiler. We can also specify annotation availability to either compile time only or till runtime also. Java Built-in annotations are @Override, @Deprecated and @SuppressWarnings.

38) What is Java Reflection API? Why it's so important to have?

Java Reflection API provides ability to inspect and modify the runtime behavior of java application. We can inspect a java class, interface, enum and get their methods and field details. Reflection API is an advanced topic and we should avoid it in normal programming. Reflection API usage can break the design pattern such as Singleton pattern by invoking the private constructor i.e violating the rules of access modifiers.

Even though we don't use Reflection API in normal programming, it's very important to have. We can't have any frameworks such as Spring, Hibernate or servers such as Tomcat, JBoss without Reflection API. They invoke the appropriate methods and instantiate classes through reflection API and use it a lot for other processing.

39) What is composition in java?

Composition is the design technique to implement has-a relationship in classes. We can use Object composition for code reuse.

Java composition is achieved by using instance variables that refers to other objects. Benefit of using composition is that we can control the visibility of other object to client classes and reuse only what we need.

40) What is the benefit of Composition over Inheritance?

One of the best practices of java programming is to "favor composition over inheritance". Some of the possible reasons are:

- Any change in the superclass might affect subclass even though we might not be using the superclass methods. For example, if we have a method test() in subclass and suddenly somebody introduces a method test() in superclass, we will get compilation errors in subclass. Composition will never face this issue because we are using only what methods we need.

- Inheritance exposes all the super class methods and variables to client and if we have no control in designing superclass, it can lead to security holes. Composition allows us to provide restricted access to the methods and hence more secure.
- We can get runtime binding in composition where inheritance binds the classes at compile time. So composition provides flexibility in invocation of methods.

41) How to sort a collection of custom Objects in Java?

We need to implement Comparable interface to support sorting of custom objects in a collection. Comparable interface has compareTo(T obj) method which is used by sorting methods and by providing this method implementation, we can provide default way to sort custom objects collection.

However, if you want to sort based on different criteria, such as sorting an Employees collection based on salary or age, then we can create Comparator instances and pass it as sorting methodology.

42) What is inner class in java?

We can define a class inside a class and they are called nested classes. Any non-static nested class is known as inner class. Inner classes are associated with the object of the class and they can access all the variables and methods of the outer class. Since inner classes are associated with instance, we can't have any static variables in them.

We can have local inner class or anonymous inner class inside a class

43) What is anonymous inner class?

A local inner class without name is known as anonymous inner class. An anonymous class is defined and instantiated in a single statement. Anonymous inner class always extend a class or implement an interface.

Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class. Anonymous inner classes are accessible only at the point where it is defined.

44) What is Classloader in Java?

Java Classloader is the program that loads byte code program into memory when we want to access any class. We can create our own classloader by extending ClassLoader class and overriding loadClass(String name) method.

45) What are different types of classloaders?

There are three types of built-in Class Loaders in Java:

1. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
2. Extensions Class Loader – It loads classes from the JDK extensions directory, usually \$JAVA_HOME/lib/ext directory.
3. System Class Loader – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

46) What does super keyword do?

super keyword can be used to access super class method when you have overridden the method in the child class.

We can use super keyword to invoke super class constructor in child class constructor but in this case
Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

it should be the first statement in the constructor method.

```
public class SuperClass {  
    public SuperClass(){  
    }  
  
    public SuperClass(int i){}  
  
    public void test(){  
        System.out.println ("super class test method");  
    }  
}
```

Use of super keyword can be seen in below child class implementation.

```
public class ChildClass extends SuperClass {  
  
    public ChildClass(String str){  
        //access super class constructor with super keyword  
        super();  
  
        //access child class method  
        test();  
  
        //use super to access super class method  
        super.test();  
    }  
  
    @Override  
    public void test(){  
        System.out.println ("child class test method");  
    }  
}
```

47) What is break and continue statement?

We can use break statement to terminate for, while, or do-while loop. We can use break statement in switch statement to exit the switch case.

The continue statement skips the current iteration of a for, while or do-while loop. We can use continue statement with label to skip the current iteration of outermost loop.

48) 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.

```
//constructor  
public Point(int x, int y) {  
    this.x = x;  
    this.y = y;
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

}

We can also use this keyword to invoke other constructors from a constructor.

```
public Rectangle() {  
    this(0, 0, 0, 0);  
}  
public Rectangle(int width, int height) {  
    this(0, 0, width, height);  
}  
public Rectangle(int x, int y, int width, int height) {  
    this.x = x;  
    this.y = y;  
    this.width = width;  
    this.height = height;  
}
```

49) What is default constructor?

No argument constructor of a class is known as default constructor. When we don't define any constructor for the class, java compiler automatically creates the default no-args constructor for the class. If there are other constructors defined, then compiler won't create default constructor for us.

50) Can we have try without catch block?

Yes, we can have try-finally statement and hence avoiding catch block.

51) What is Garbage Collection?

Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, process of deallocating memory is handled automatically by the garbage collector.

We can run the garbage collector with code Runtime.getRuntime().gc() or use utility method System.gc().

52) What is Serialization and Deserialization?

We can convert a Java object to an Stream that is called Serialization. Once an object is converted to Stream, it can be saved to file or send over the network or used in socket connections.

The object should implement Serializable interface and we can use java.io.ObjectOutputStream to write object to file or to any OutputStream object.

The process of converting stream data created through serialization to Object is called deserialization.

53) How to run a JAR file through command prompt?

We can run a jar file using java command but it requires Main-Class entry in jar manifest file. Main-Class is the entry point of the jar and used by java command to execute the class.

54) What will be the output of following programs?

1. **static method in class**

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
public class Test {  
  
    public static String toString(){  
        System.out.println ("Test toString called");  
        return "";  
    }  
  
    public static void main(String [] args){  
        System.out.println (toString());  
    }  
}
```

Answer: The code won't compile because we can't have an Object class method with static keyword. Note that Object class has `toString()` method. You will get compile time error as "This static method cannot hide the instance method from Object". The reason is that static method belongs to class and since every class base is Object, we can't have same method in instance as well as in class. You won't get this error if you change the method name from `toString()` to something else that is not present in super class Object.

static method invocation

```
public class Test {  
  
    public static String display(){  
        System.out.println ("Test display called");  
        return "";  
    }  
  
    public static void main(String [] args){  
        Test obj = null;  
        System.out.println (obj. display () );  
    }  
}
```

Answer: Well this is a strange situation. We all have seen `NullPointerException` when we invoke a method on object that is NULL. But here this program will work and prints "Test foo called".

The reason for this is the java compiler code optimization. When the java code is compiled to produced byte code, it figures out that `foo()` is a static method and should be called using class. So it changes the method call `obj. display ()` to `Test. display ()` and hence no `NullPointerException`.

55) What is marker interface?

An interface that have no data member and method is known as a marker interface. For example `Serializable`, `Cloneable` etc.

56) What is difference between abstract class and interface?

Interface

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- | | |
|---|--|
| 1) An abstract class can have method body (non-abstract methods). | Interface have only abstract methods. |
| 2) An abstract class can have instance variables. | An interface cannot have instance variables. |
| 3) An abstract class can have constructor. | Interface cannot have constructor. |
| 4) An abstract class can have static methods. | Interface cannot have static methods. |
| 5) You can extends one abstract class. | You can implement multiple interfaces. |

57) What is package?

A package is a group of similar type of classes interfaces and sub-packages. It provides access protection and removes naming collision.

58) Do I need to import java.lang package any time? Why ?

No. It is by default loaded internally by the JVM.

59) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or same class multiple times. Neither compiler nor JVM complains about it. But the JVM will internally load the class only once no matter how many times you import the same class.

60) What is static import ?

By static import, we can access the static members of a class directly, there is no to qualify it with the class name.

61) What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors. It is mainly used to handle checked exceptions.

62) What is difference between Checked Exception and Unchecked Exception?

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException etc. Unchecked exceptions are not checked at compile-time.

63) What is the base class for Error and Exception?

Throwable.

64. How an object is serialized in java?

Ans: In java, to convert an object into byte stream by serialization, an interface with the name

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

Serializable is implemented by the class. All objects of a class implementing serializable interface get serialized and their state is saved in byte stream.

65. When we should use serialization?

Ans: Serialization is used when data needs to be transmitted over the network. Using serialization, object's state is saved and converted into byte stream .The byte stream is transferred over the network and the object is re-created at destination.

66) Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

67) What is finally block?

- finally block is a block that is always executed.

68) Can finally block be used without catch?

- Yes, by try block. finally must be followed by either try or catch.

69) Is there any case when finally will not be executed?

finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

70) What is difference between throw and throws?

throw keyword

1)throw is used to explicitly throw an exception. throws is used to declare an exception.

2)checked exceptions can not be propagated with throw only. checked exception can be propagated with throws.

3)throw is followed by an instance. throws is followed by class.

4)throw is used within the method. throws is used with the method signature.

5)You cannot throw multiple exception e.g. public void method()throws IOException,SQLException.

71) Can an exception be rethrown?

Yes.

72) Can subclass overriding method declare an exception if parent class method doesn't throw an exception ?

Yes but only unchecked exception not checked.

73) What is exception propagation ?

Forwarding the exception object to the invoking method is known as exception propagation.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

74) What is the meaning of immutable in terms of String?

The simple meaning of immutable is unmodifiable or unchangeable. Once string object has been created, its value can't be changed.

75) Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

76) How many ways we can create the string object?

There are two ways to create the string object, by string literal and by new keyword.

77) How many objects will be created in the following code?

1. String s1="Welcome";
2. String s2="Welcome";
3. String s3="Welcome";

Only one object.

78) Why java uses the concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

79) How many objects will be created in the following code?

1. String s = new String("Welcome");

Two objects, one in string constant pool and other in non-pool(heap)

80) What is the purpose of `toString()` method in java ?

The `toString()` method returns the string representation of any object. If you print any object, java compiler internally invokes the `toString()` method on the object. So overriding the `toString()` method, returns the desired output, it can be the state of an object etc. depends on your implementation.

81) What is nested class?

A class which is declared inside another class is known as nested class. There are 4 types of nested class member inner class, local inner class, anonymous inner class and static nested class.

82) Is there any difference between nested classes and inner classes?

Yes, inner classes are non-static nested classes i.e. inner classes are the part of nested classes.

83) Can we access the non-final local variable, inside the local inner class?

No, local variable must be constant if you want to access it in local inner class.

84) What is nested interface ?

Any interface i.e. declared inside the interface or class, is known as nested interface. It is static by default.

85) Can a class have an interface?

Yes, it is known as nested interface.

86) Can an Interface have a class?

Yes, they are static implicitly.

87) What is Garbage Collection?

Garbage collection is a process of reclaiming the runtime unused objects. It is performed for memory management.

88) What is gc()?

gc() is a daemon thread. gc() method is defined in System class that is used to send request to JVM to perform garbage collection.

89) What is the purpose of finalize() method?

finalize() method is invoked just before the object is garbage collected. It is used to perform cleanup processing.

90) Can an unrefrenced objects be refrenced again?

Yes.

91)What kind of thread is the Garbage collector thread?

Daemon thread.

92)What is difference between final, finally and finalize?

final: final is a keyword, final can be variable, method or class. You, can't change the value of final variable, can't override final method, can't inherit final class.

finally: finally block is used in exception handling. finally block is always executed.

finalize(): finalize() method is used in garbage collection. finalize() method is invoked just before the object is garbage collected. The finalize() method can be used to perform any cleanup processing.

93)What is the purpose of the Runtime class?

The purpose of the Runtime class is to provide access to the Java runtime system.

94)How will you invoke any external process in Java?

By Runtime.getRuntime().exec(?) method.

95)What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

96)What an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.

97) What is serialization?

Serialization is a process of writing the state of an object into a byte stream. It is mainly used to travel object's state on the network.

98) What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization.

99) What is transient keyword?

If you define any data member as transient, it will not be serialized.

100)What is Externalizable?

Externalizable interface is used to write the state of an object into a byte stream in compressed format. It is not a marker interface.

101)What is the difference between Serializable and Externalizable interface?

Serializable is a marker interface but Externalizable is not a marker interface. When you use Serializable interface, your class is serialized automatically by default. But you can override writeObject() and readObject() two methods to control more complex object serialization process. When you use Externalizable interface, you have a complete control over your class's serialization process.

102)How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?

By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address.

103) What is reflection?

Reflection is the process of examining or modifying the runtime behaviour of a class at runtime. It is used in:

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans.
- Debugger
- Test Tools etc.

104) Can you access the private method from outside the class?

Yes, by changing the runtime behaviour of a class if the class is not secured.

105)What are wrapper classes?

Wrapper classes are classes that allow primitive types to be accessed as objects.

106)What is a native method?

A native method is a method that is implemented in a language other than Java.

107)What is the purpose of the System class?

The purpose of the System class is to provide access to system resources.

108)What comes to mind when someone mentions a shallow copy in Java?

Object cloning.

109)What is singleton class?

Singleton class means that any given time only one instance of the class is present, in one JVM.

110)What is a lightweight component?

Lightweight components are the one which doesn't go with the native call to obtain the graphical units. They share their parent component graphical units to render them. For example, Swing components.

111)What is a heavyweight component?

For every paint call, there will be a native call to get the graphical units. For Example, AWT.

161)What is Locale?

A Locale object represents a specific geographical, political, or cultural region.

162)How will you load a specific locale?

By ResourceBundle.getBundle(?) method.

Java Versions, Features and History

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- Released on 23 January 1996, JDK 1.0 version

- Released on 19 February 1997 JDK 1.1 version.

New features in JDK 1.1

1. JDBC (Java Database Connectivity)
2. [Inner Classes](#)
3. Java Beans
4. RMI (Remote Method Invocation)
5. Reflection (introspection only)

- Released on 8 December 1998 J2SE 1.2 version.

New features in J2SE 1.2

1. [Collections framework.](#)
2. Java String memory map for constants.
3. Just In Time (JIT) compiler.
4. Jar Signer for signing Java ARchive (JAR) files.
5. Policy Tool for granting access to system resources.
6. Java Foundation Classes (JFC) which consists of Swing 1.0, Drag and Drop, and Java 2D class libraries.
7. Java Plug-in
8. Scrollable result sets, BLOB, CLOB, batch update, user-defined types in JDBC.
9. Audio support in Applets.

- Released on 8 May 2000 J2SE 1.3 version.

New features in J2SE 1.3

1. Java Sound
2. Jar Indexing
3. A huge list of enhancements in almost all the java area.

- Released on 6 February 2002 J2SE 1.4 version.

New features in J2SE 1.4

1. XML Processing
2. Java Print Service
3. Logging API
4. Java Web Start
5. JDBC 3.0 API
6. Assertions
7. Preferences API
8. Chained Exception
9. IPv6 Support
10. Regular Expressions
11. Image I/O API

- **Released on 30 September 2004 J2SE 1.5 version.**

New features in J2SE 1.5

1. Generics
2. [Enhanced for Loop](#)
3. [Autoboxing/Unboxing](#)
4. [Enum](#)
5. [Varargs](#)
6. [Static Import](#)
7. Metadata (Annotations)
8. Instrumentation

- **Released on 11 December 2006 J2SE 1.6 version.**

New features in J2SE 1.6

1. Scripting Language Support
2. JDBC 4.0 API
3. Java Compiler API
4. Pluggable Annotations
5. Native PKI, Java GSS, Kerberos and LDAP support.
6. Integrated Web Services.
7. Lot more enhancements.

- **Released on 28 July 2011 J2SE 1.7 version.**

New features in J2SE 1.7

1. Strings in switch Statement
2. Type Inference for Generic Instance Creation
3. Multiple Exception Handling
4. Support for Dynamic Languages
5. Try with Resources
6. Java nio Package
7. Binary Literals, underscore in literals
8. Diamond Syntax
9. Automatic null Handling

- **Released on 18th march 2014 JDK 1.8 version.**

New features in JDK 1.8

1. [Default and Static methods in Interface](#)
2. Lambda Expressions
3. Optional
4. Streams
5. Method References
6. Data Time API
7. Nashorn Javascript Engine
8. Parallel Arrays

Some of the important java 9 features are;

- 1) [Java 9 REPL \(JShell\)](#)
- 2) [Factory Methods for Immutable List, Set, Map and Map.Entry](#)
- 3) [Private methods in Interfaces](#)
- 4) [Java 9 Module System](#)
- 5) [Process API Improvements](#)
- 6) [Try With Resources Improvement](#)
- 7) [CompletableFuture API Improvements](#)
- 8) [Reactive Streams](#)
- 9) [Diamond Operator for Anonymous Inner Class](#)
- 10) [Optional Class Improvements](#)
- 11) [Stream API Improvements](#)
- 12) [Enhanced @Deprecated annotation](#)
- 13) [HTTP 2 Client](#)
- 14) [Multi-Resolution Image API](#)
- 15) [Miscellaneous Java 9 Features](#)

1.Java 9 REPL (JShell)

Oracle Corp has introduced a new tool called “jshell”. It stands for Java Shell and also known as REPL (Read Evaluate Print Loop). It is used to execute and test any Java Constructs like class, interface, enum, object, statements etc. very easily.

```
G:\>jshell
| Welcome to JShell -- Version 9-ea
| For an introduction type: /help intro
```

```
jshell> int a = 10
a ==> 10
```

```
jshell> System.out.println ("a value = " + a )
a value = 10
```

2.Factory Methods for Immutable List, Set, Map and Map.Entry

Oracle Corp has introduced some convenient factory methods to create Immutable List, Set, Map and Map.Entry objects. These utility methods are used to create empty or non-empty Collection objects.

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

In Java SE 8 and earlier versions, We can use Collections class utility methods like unmodifiableXXX to create Immutable Collection objects. For instance, if we want to create an Immutable List, then we can use Collections.unmodifiableList method.

However these Collections.unmodifiableXXX methods are very tedious and verbose approach. To overcome those shortcomings, Oracle corp has added couple of utility methods to List, Set and Map interfaces.

List and Set interfaces have “of()” methods to create an empty or no-empty Immutable List or Set objects as shown below:

Empty List Example

```
List immutableList = List.of();
```

Non-Empty List Example

```
List immutableList = List.of("one", "two", "three");
```

Map has two set of methods: of() methods and ofEntries() methods to create an Immutable Map object and an Immutable Map.Entry object respectively.

Empty Map Example

```
jshell> Map emptyImmutableMap = Map.of()  
emptyImmutableMap ==> {}
```

Non-Empty Map Example

```
jshell> Map nonemptyImmutableMap = Map.of(1, "one", 2, "two", 3, "three")  
nonemptyImmutableMap ==> {2=two, 3=three, 1=one}
```

3.Private methods in Interfaces

In Java 8, we can provide method implementation in Interfaces using Default and Static methods. However we cannot create private methods in Interfaces.

To avoid redundant code and more re-usability, Oracle Corp is going to introduce private methods in Java SE 9 Interfaces. From Java SE 9 on-wards, we can write private and private static methods too in an interface using ‘private’ keyword.

These private methods are like other class private methods only, there is no difference between them.

```
public interface Card{
```

```
    private Long createCardID(){  
        // Method implementation goes here.  
    }
```

```
private static void displayCardDetails(){  
    // Method implementation goes here.  
}  
}
```

4.Java 9 Module System

One of the big changes or java 9 feature is the Module System. Oracle Corp is going to introduce the following features as part of **Jigsaw Project**.

- Modular JDK
- Modular Java Source Code
- Modular Run-time Images
- Encapsulate Java Internal APIs
- Java Platform Module System

Before Java SE 9 versions, we are using Monolithic Jars to develop Java-Based applications. This architecture has lot of limitations and drawbacks. To avoid all these shortcomings, Java SE 9 is coming with Module System.

JDK 9 is coming with 92 modules (may change in final release). We can use JDK Modules and also we can create our own modules as shown below:

Simple Module Example

Here We are using ‘module’ to create a simple module. Each module has a name, related code and other resources.

5.Process API Improvements

Java SE 9 is coming with some improvements in Process API. They have added couple new classes and methods to ease the controlling and managing of OS processes.

Two new interface in Process API:

- `java.lang.ProcessHandle`
- `java.lang.ProcessHandle.Info`

Process API example

```
ProcessHandle currentProcess = ProcessHandle.current();  
System.out.println ("Current Process Id: = " + currentProcess.pid());
```

6.Try With Resources Improvement

We know, Java SE 7 has introduced a new exception handling construct: Try-With-Resources to manage resources automatically. The main goal of this new statement is “Automatic Better Resource Management”.

Java SE 9 is going to provide some improvements to this statement to avoid some more verbosity and improve some Readability.

Java SE 7 example

```
void testARM_Before_Java9() throws IOException{
    BufferedReader reader1 = new BufferedReader(new FileReader("journaldev.txt"));
    try (BufferedReader reader2 = reader1) {
        System.out.println (reader2.readLine());
    }
}
```

Java 9 example

```
void testARM_Java9() throws IOException{
    BufferedReader reader1 = new BufferedReader(new FileReader("journaldev.txt"));
    try (reader1) {
        System.out.println (reader1.readLine());
    }
}
```

7.CompletableFuture API Improvements

In Java SE 9, Oracle Corp is going to improve CompletableFuture API to solve some problems raised in Java SE 8. They are going add to support some delays and timeouts, some utility methods and better sub-classing.

```
Executor exe = CompletableFuture.delayedExecutor(50L, TimeUnit.SECONDS);
```

Here delayedExecutor() is static utility method used to return a new Executor that submits a task to the default executor after the given delay.

8.Reactive Streams

Now-a-days, Reactive Programming has become very popular in developing applications to get some beautiful benefits. Scala, Play, Akka etc. Frameworks has already integrated Reactive Streams and getting many benefits. Oracle Corps is also introducing new Reactive Streams API in Java SE 9.

Java SE 9 Reactive Streams API is a Publish/Subscribe Framework to implement Asynchronous, Scalable and Parallel applications very easily using Java language.

Java SE 9 has introduced the following API to develop Reactive Streams in Java-based applications.

- java.util.concurrent.Flow
- java.util.concurrent.Flow.Publisher
- java.util.concurrent.Flow.Subscriber
- java.util.concurrent.Flow.Processor

9.Diamond Operator for Anonymous Inner Class

We know, Java SE 7 has introduced one new feature: Diamond Operator to avoid redundant code and verbosity, to improve readability. However in Java SE 8, Oracle Corp (Java Library Developer) has found that some limitation in the use of Diamond operator with Anonymous Inner Class. They have fixed that issues and going to release as part of Java 9.

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
public List getEmployee(String empid){  
    // Code to get Employee details from Data Store  
    return new List(emp){ };  
}
```

Here we are using just “List” without specifying the type parameter.

10.Optional Class Improvements

In Java SE 9, Oracle Corp has added some useful new methods to `java.util.Optional` class. Here I’m going to discuss about one of those methods with some simple example: stream method

If a value present in the given Optional object, this `stream()` method returns a sequential Stream with that value. Otherwise, it returns an Empty Stream.

They have added “`stream()`” method to work on Optional objects lazily as shown below:

```
Stream<Optional> emp = getEmployee(id)  
Stream empStream = emp.flatMap(Optional::stream)
```

Here `Optional.stream()` method is used convert a Stream of Optional of Employee object into a Stream of Employee so that we can work on this result lazily in the result code.

11.Stream API Improvements

In Java SE 9, Oracle Corp has added four useful new methods to `java.util.Stream` interface. As Stream is an interface, all those new implemented methods are default methods. Two of them are very important: `dropWhile` and `takeWhile` methods

If you are familiar with Scala Language or any Functions programming language, you will definitely know about these methods. These are very useful methods in writing some functional style code. Let us discuss about `takeWhile` utility method here.

This `takeWhile()` takes a predicate as an argument and returns a Stream of subset of the given Stream values until that Predicate returns false for first time. If first value does NOT satisfy that Predicate, it just returns an empty Stream.

```
jshell> Stream.of(1,2,3,4,5,6,7,8,9,10).takeWhile(i -> i < 5 )  
      .forEach(System.out::println);  
1  
2  
3  
4
```

12 Enhanced @Deprecated annotation

In Java SE 8 and earlier versions, `@Deprecated` annotation is just a Marker interface without any methods. It is used to mark a Java API that is a class, field, method, interface, constructor, enum etc.

In Java SE 9, Oracle Corp has enhanced `@Deprecated` annotation to provide more information about deprecated API and also provide a **Tool** to analyse an application’s static usage of deprecated APIs.

They have add two methods to this Deprecated interface: **forRemoval** and **since** to serve this information.

13.HTTP 2 Client

In Java SE 9, Oracle Corp is going to release New HTTP 2 Client API to support HTTP/2 protocol and WebSocket features. As existing or Legacy HTTP Client API has numerous issues (like supports HTTP/1.1 protocol and does not support HTTP/2 protocol and WebSocket, works only in Blocking mode and lot of performance issues.), they are replacing this HttpURLConnection API with new HTTP client.

They are going to introduce new HTTP 2 Client API under “java.net.http” package. It supports both HTTP/1.1 and HTTP/2 protocols. It supports both Synchronous (Blocking Mode) and Asynchronous Modes. It supports Asynchronous Mode using WebSocket API.

HTTP 2 Client Example

```
jshell> import java.net.http.*  
jshell> import static java.net.http.HttpRequest.*  
jshell> import static java.net.http.HttpResponse.*  
  
jshell> URI uri = new URI("http://rams4java.blogspot.co.uk/2016/05/java-news.html")  
uri ==> http://rams4java.blogspot.co.uk/2016/05/java-news.html  
  
jshell> HttpResponse response = HttpRequest.create(uri).body(noBody()).GET().response()  
response ==> java.net.http.HttpResponseImpl@79efed2d  
  
jshell> System.out.println ("Response was " + response.body(asString()))
```

14.Multi-Resolution Image API

In Java SE 9, Oracle Corp is going to introduce a new Multi-Resolution Image API. Important interface in this API is MultiResolutionImage . It is available in java.awt.image package.

MultiResolutionImage encapsulates a set of images with different Height and Widths (that is different resolutions) and allows us to query them with our requirements.

15.Miscellaneous Java 9 Features

In this section, I will just list out some miscellaneous Java SE 9 New Features. I'm NOT saying these are less important features. They are also important and useful to understand them very well with some useful examples.

As of now, I did not get enough information about these features. That's why I am going list them here for brief understanding. I will pickup these Features one by one and add to above section with a brief discussion and example. And final write a separate tutorial later.

- GC (Garbage Collector) Improvements
- Stack-Walking API

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- Filter Incoming Serialization Data
 - Deprecate the Applet API
 - Indify String Concatenation
 - Enhanced Method Handles
 - Java Platform Logging API and Service
 - Compact Strings
 - Parser API for Nashorn
 - Javadoc Search
-
- HTML5 Javadoc

Java Inner Class

Java inner class is defined inside the body of another class. Java inner class can be declared private, public, protected, or with default access whereas an outer class can have only public or default access.

static nested class

If the nested class is static, then it's called static nested class. Static nested classes can access only static members of the outer class. Static nested class is same as any other top-level class and is nested for only packaging convenience.

Static class object can be created with following statement.

```
OuterClass.StaticNestedClass nestedObject =  
    new OuterClass.StaticNestedClass();
```

Any non-static nested class is known as inner class in java. Java inner class is associated with the object of the class and they can access all the variables and methods of the outer class.

Since inner classes are associated with instance, we can't have any static variables in them.

Object of java inner class are part of the outer class object and to create an instance of inner class, we first need to create instance of outer class.

Java inner class can be instantiated like this;

```
OuterClass outerObject = new OuterClass();  
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
```

There are two special kinds of java inner classes.

1. local inner class.

If a class is defined in a method body, it's known as local inner class.

Since local inner class is not associated with Object, we can't use private, public or protected access modifiers with it. The only allowed modifiers are abstract or final.

A local inner class can access all the members of the enclosing class and local final variables in the scope it's defined.

Local inner class can be defined as:

```
public void print() {  
    //local inner class inside the method  
    class Logger {  
        String name;  
    }  
    //instantiate local inner class in the method to use  
    Logger logger = new Logger();
```

2. anonymous inner class

A local inner class without name is known as anonymous inner class. An anonymous class is defined and instantiated in a single statement.

Anonymous inner class always extend a class or implement an interface. Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class.

Anonymous inner classes are accessible only at the point where it is defined.

It's a bit hard to define how to create anonymous inner class, we will see it's real time usage in test program below.

Here is a java class showing how to define java inner class, static nested class, local inner class and anonymous inner class.

```
import java.io.File;  
import java.io.FilenameFilter;  
  
public class OuterClass {  
  
    private static String name = "OuterClass";  
    private int i;  
    protected int j;  
    int k;  
    public int l;  
  
    //OuterClass constructor  
    public OuterClass(int i, int j, int k, int l) {  
        this.i = i;  
        this.j = j;  
        this.k = k;  
        this.l = l;  
    }  
  
    public int getI() {  
        return this.i;  
    }
```

}

//static nested class, can access OuterClass static variables/methods
static class StaticNestedClass {

 private int a;
 protected int b;
 int c;
 public int d;

 public int getA() {
 return this.a;
 }

 public String getName() {
 return name;
 }

}

//inner class, non static and can access all the variables/methods of outer class
class InnerClass {

 private int w;
 protected int x;
 int y;
 public int z;

 public int getW() {
 return this.w;
 }

 public void setValues() {
 this.w = i;
 this.x = j;
 this.y = k;
 this.z = l;
 }

 @Override
 public String toString() {
 return "w=" + w + ":x=" + x + ":y=" + y + ":z=" + z;
 }

 public String getName() {
 return name;
 }

}

//local inner class

 public void print(String initial) {
 //local inner class inside the method
 class Logger {
 String name;

```
public Logger(String name) {
    this.name = name;
}

public void log(String str) {
    System.out.println(this.name + ": " + str);
}

Logger logger = new Logger(initial);
logger.log(name);
logger.log("'" + this.i);
logger.log("'" + this.j);
logger.log("'" + this.k);
logger.log("'" + this.l);
}

//anonymous inner class
public String[] getFilesInDir(String dir, final String ext) {
    File file = new File(dir);
    //anonymous inner class implementing FilenameFilter interface
    String[] filesList = file.list(new FilenameFilter() {

        @Override
        public boolean accept(File dir, String name) {
            return name.endsWith(ext);
        }

    });
    return filesList;
}
}
```

Here is the test program showing how to instantiate and use inner class in java.

```
import java.util.Arrays;
//nested classes can be used in import for easy instantiation
import com.journaldev.nested.OuterClass.InnerClass;
import com.journaldev.nested.OuterClass.StaticNestedClass;

public class InnerClassTest {

    public static void main(String[] args) {
        OuterClass outer = new OuterClass(1,2,3,4);

        //static nested classes example
        StaticNestedClass staticNestedClass = new StaticNestedClass();
        StaticNestedClass staticNestedClass1 = new StaticNestedClass();

        System.out.println(staticNestedClass.getName());
        staticNestedClass.d=10;
        System.out.println(staticNestedClass.d);
    }
}
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
System.out.println(staticNestedClass1.d);

//inner class example
InnerClass innerClass = outer.new InnerClass();
System.out.println(innerClass.getName());
System.out.println(innerClass);
innerClass.setValues();
System.out.println(innerClass);

//calling method using local inner class
outer.print("Outer");

//calling method using anonymous inner class
System.out.println(Arrays.toString(outer.GetFilesInDir("src/com/journaldev/nested", ".java")));

System.out.println(Arrays.toString(outer.GetFilesInDir("bin/com/journaldev/nested", ".class")));
}

}
```

Here is the output of above java inner class example program.

```
OuterClass
10
0
OuterClass
w=0:x=0:y=0:z=0
w=1:x=2:y=3:z=4
Outer: OuterClass
Outer: 1
Outer: 2
Outer: 3
Outer: 4
[NestedClassTest.java, OuterClass.java]
[NestedClassTest.class, OuterClass$1.class, OuterClass$1Logger.class, OuterClass$InnerClass.class,
OuterClass$StaticNestedClass.class, OuterClass.class]
```

Notice that when OuterClass is compiled, separate class files are created for inner class, local inner class and static nested class.

Benefits of Java Inner Class

1. If a class is useful to only one class, it makes sense to keep it nested and together. It helps in packaging of the classes.
2. Java inner classes implements encapsulation. Note that inner classes can access outer class private members and at the same time we can hide inner class from outer world.
3. Keeping the small class within top-level classes places the code closer to where it is used and makes code more readable and maintainable.

Java varargs

varargs in java enables a method to accept variable number of arguments. We use three dots (...) also known as ellipsis in the method signature to make it accept variable arguments. For example;

```
public static int sum(int i, int...js ){  
//do something  
}
```

Important points about varargs in java

Few points to know about varargs in java are;

1. We can have only one varargs in the method.
2. Only the last argument of a method can be varargs.
3. According to java documentation, we should not overload a varargs method. We will see why it's not a good idea.

How java varargs work?

When we invoke a method with variable arguments, java compiler matches the arguments from left to right. Once it reaches to the last varargs parameter, it creates an [array](#) of the remaining arguments and pass it to the method. In fact varargs parameter behaves like an array of the specified type.

```
//method with variable arguments  
public static int sum(int i, int...js ){  
    int sum = i;  
    for(int x : js){  
        sum+=x;  
    }  
    return sum;  
}  
  
//method with same implementation as sum with array as argument  
public static int sumArray(int i, int[] js ){  
    int sum = i;  
    for(int x : js){  
        sum+=x;  
    }  
    return sum;  
}
```

If you will look at both sum and sumArray methods, you will see that the implementation body is exactly same. So we should use varargs when API offers them, for example `java.io.PrintStream.printf()` method but we should not take it as a replacement for array.

```
public class VarargsExample {  
  
    public static void main(String[] args) {  
        System.out.println(sum(1));  
        System.out.println(sum(1,2)); //compiler error, ambiguous method  
    }  
}
```

```
public static int sum(int i, int...js ){
    System.out.println("sum1 called");
    int sum = i;
    for(int x : js){
        sum+=x;
    }
    return sum;
}

public static int sum(int i, int k, Object...js ){
    System.out.println("sum2 called");
    int sum = i+k;
    for(Object x : js){
        sum+=1;
    }
    return sum;
}

}
```

In above example, you will notice that compiler will not complain when we overload methods with varargs. But when we try to use it, compiler get's confused which method to use when mapping the second argument.

If there is only one argument, compiler is smart to use first method because it can work with minimum one argument but second method needs at least two arguments. Below image from Eclipse shows the error message as The method sum(int, int[]) is ambiguous for the type VarargsExample.

Interview Questions

1) A result of expression involving byte, int, and literal numbers is promoted to which of these?

- [A. int](#)
- [B. long](#)
- [C. byte](#)
- [D. float](#)

Answer: Option A

Explanation:

An expression involving bytes, ints, shorts, literal numbers, the result of the expression is promoted

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

to next data type after calculation is done.

2. What is the numerical range of a char in Java?

- A. -128 to 127
- B. 0 to 256
- C. 0 to 32767
- D. 0 to 65535

Answer: Option D

Explanation:

In java Char is a Unicode char, which occupies 16-bit in memory, it supports ranging from 0 to 65535.

3.Which of these values can a boolean variable contain?

- A. True & False
- B. 0 & 1
- C. Any integer value.
- D. Both a & b

Answer: Option A

Explanation:

Boolean variable can contain only one of two possible values, true and false.

4. Which one is a valid declaration of a boolean?

- A. boolean b1 = 1;
- B. boolean b2 = ‘false’;
- C. boolean b3 = false;
- D. boolean b4 = ‘true’

Answer: Option C

Explanation:

Boolean can only be assigned true or false literals.

5) What is the output of this program?

```
class Test {  
    public static void main(String args[])  
    {  
        boolean var1 = true;  
        boolean var2 = false;  
        if(var1)  
            System.out.println(var1);  
        else
```

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

```
System.out.println(var2);
}
}
```

- A. 0
- B. 1
- C. true
- D. false

Answer: Option C

Explanation:

true

6) Modulus operator, %, can be applied to which of these?

- A. Integers
- B. Floating – point numbers
- C. Both Integers and floating – point numbers.
- D. None of the mentioned

Answer: Option C

Explanation:

Modulus operator can be applied to both integers and floating point numbers.

7) With x = 0, which of the following are valid Java code for changing the value of x to 1?

1. x++;
2. x = x + 1;
3. x += 1;
4. x =+ 1;

- A. 1, 2 & 3
- B. 1 & 4
- C. 1, 2, 3 & 4
- D. 3 & 2

Answer: Option A

Explanation:

Operator ++ increases value of variable by 1. x = x + 1 can also be written in shorthand form as x += 1. Also x =+ 1 will set the value of x to 1.

8) What is the output of this program?

```
class Test {
```

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

```
public static void main(String args[])
{
int a = 3;
System.out.print(++a * 8);
}
```

- A. 25
- B. 24
- C. 32
- D. 33

Answer: Option C

Explanation:

Operator ++ is a pre-increment, thus a becomes 4 and when multiplied by 8 gives 32.

9) What is the output of this program?

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

- A.

11 11

- B.

10 10

- C.

11 10

- D.

10 11

Answer: Option C

Explanation:

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

x is initialized to 10 then increased by 1 by ++ operator making it 11. x is again decreased by — operator making it 10, next x is incremented by post increment and initialized to y, here the value of x obtained before increment operator is executed, so value of y is 10 and value of x is 11.

11) Which operator is used to invert all the digits in binary representation of a number?

- A. ~
- B. >>>
- C. ^
- D. !

Answer: Option A

Explanation:

Unary not operator, ~, inverts all of the bits of its operand in binary representation.

12) What is the return value of relational operators?

- A. Integer
- B. Boolean
- C. Characters
- D. Double

Answer: Option B

Explanation:

Boolean

13) Which of the following operators can operate on a boolean variable?

1. &&
2. ==
3. ?:
4. +=

- A. 3 & 2
- B. 1 & 4
- C. 1, 2 & 4
- D. 1, 2 & 3

Answer: Option D

Explanation:

Operator &&, is equal to(==) , ternary if-then-else, ?:, are boolean logical operators. += is an arithmetic operator it can operate only on numeric values.

14) What is the output of the following.

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

- A. 10
- B. 11
- C. 12
- D. 13

Answer: Option C

15) Which of these is an incorrect array declaration?

A. int arr[] = new int[5] ;

B. int [] arr = new int[5] ;

C. int arr[];

arr = new int[5];

D. int arr[5] = new int[5]

Answer: Option D

Explanation:

Size of array is not valid at compile time.

16) What is the out put of the following

```
public class Test {  
  
    public static void main(String[] args) {  
        int [] arr = new int[5];  
        System.out.println(arr);  
    }  
}
```

}

- A) 0 0 0 0
- B) 5 (length of array)
- C) 5 garbage values
- D) Address of array

Answer : Option D

17. Which of the following are Java reserved words?

- 1) run
- 2) import
- 3) default
- 4) implements

- A) 2 and 3
- B) 2 and 4
- C) 2 , 3 and 4
- D) 1 and 2**

Answer : Option B

18) Which class or interface defines the wait(), notify(),and notifyAll() methods?

- A) Object
- B) Runnable
- C) Class
- D) Thread

Answer : Option A

19) Which of the following statements is preferred to create a string "Welcome to Java Programming"?

Options

- A) String str = "Welcome to Java Programming";
- B) String str = new String("Welcome to Java Programming");
- C) String str;
 str = "Welcome to Java Programming";
- D) String str;
 str = new String ("Welcome to Java Programming");

Answer : A

Explanation: value of string is known. Compile time is the best.

20) What is the difference between this() and super()

- A) super() constructor is invoked within a method of a class while this() constructor is used within the constructor of the sub class

- B) this() constructor is invoked outside a method of a class while super() constructor is used within the constructor of the sub class
- C) this() constructor is invoked within a method of a class while super() constructor is used within the constructor of the sub class
- D) this() constructor is invoked within a method of a class while super() constructor is used outside the constructor of the sub class

Answer : Option C

21) Which of the following is TRUE for serialization in JAVA?

- A) Process of converting classes' instance into stream of bytes
- B) Doesn't help in persisting data
- C) Both a and d
- D) An operation in which an object's internal state is converted into a stream of bytes

Answer : Option C

22) The object of DataInputStream is used to

- A) To convert binary stream into character stream
- B) To convert character stream into binary stream
- C) To write data onto output object
- D) All of the above

Answer : Option A

23) Which of the following is synchronized?

- A) Set
- B) LinkedList
- C) Vector
- D) ArrayList

Answer : Option C

24) Which of the following statements is false ?

- A) An instance of a class is an object
- B) Objects can access both static and instance data
- C) Object is the super class of all other classes
- D) Objects do not permit encapsulation

Answer : Option D

25) What is an aggregate object?

- A) An object with only primitive attributes

**Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123**

- B) An instance of a class which has only static methods
- C) An instance which has other objects
- D) None of the above

Answer : Option C

26) All the wrapper classes (Integer, Boolean, Float, Short, Long, Double and Character) in java

- A) are private
- B) are serializable
- C) are immutable
- D) are final

Answer : Option B, C and D

27) Select all the true statements from the following.

- A) AbstractSet extends AbstractCollection
- B) AbstractList extends AbstractCollection
- C) Vector extends AbstractList
- D) All of the above

Answer : Option : D

28) To execute the threads one after another

- A) the keyword synchronize is used
- B) the keyword synchronizable is used
- C) the keyword synchronized is used
- D) None of the above

CORRECT ANSWER : Option C

29) When a thread terminates its processing, into what state that thread enters?

- A) Running state
- B) Waiting state
- C) Dead state
- C) Beginning state

ANSWER : Option C

30) If result = 2 + 3 * 5, what is the value and type of 'result' variable?

- A) 17, byte
- B) 25, byte
- C) 17, int
- D) 25, int

ANSWER : Option C

31) Which of the following statements is true?

Nageswar Rao Mandru, World class Java Trainer, DurgaSoft Pvt Ltd,
Hyderabad- what app No. 8 179 189 123

- A) The default char data type is a space(' ') character.
- B) The default integer data type is 'int' and real data type is 'float'
- C) The default integer data type is 'long' and real data type is 'float'
- D) The default integer data type is 'int' and real data type is 'double'

ANSWER : Option D

32)What is the value of salaries [3]?

```
public class Test {  
    public static void main(String[] args) {  
        int salaries[];  
        int index = 0;  
        salaries = new int[4];  
        while (index < 4) {  
            salaries[index] = salaries[index] + 10000;  
            index++;  
        }  
  
        System.out.println(salaries[3]);  
    }  
}  
A) 10000  
B) 20000  
C) 30000  
D) 40000
```

Answer : Option A

33) Which of the following is not a method of the Thread class.

- A) public void run()
- B) public void start()
- C) public void exit()
- D) public final int getPriority()

34) What is the output of the following

```
public class Test {  
    public static void main(String[] args) {  
        int i=1;  
        while(i++<=12);  
        System.out.println(i);  
    }  
}  
A) 12  
B) 13  
C) 14  
D) Infinite loop
```

Answer : Option C

35) After the following code fragment, what is the value in fname?

```
String str;  
int fname;  
str = "Foolish boy.";  
fname = str.indexOf("fool");
```

- A) 0
- B) 1
- C) -1
- D) 2

Answer : Option C

36) Consider the following code snippet. What will be assigned to the variable fourthChar, if the code is executed?

```
String str = new String("Java");  
char fourthChar = str.charAt(4);
```

Options

- A) 'a'
- B) 'v'
- C) throws StringIndexOutOfBoundsException
- D) null character

ANSWER : Option C

37) Which statement is true?

- A) HashTable is a sub class of Dictionary
- B) ArrayList is a sub class of Vector
- C) LinkedList is a subclass of ArrayList
- D) Vector is a subclass of Stack

ANSWER : Option A

38)What kind of thread is the Garbage collector thread is?

- A) Non daemon thread
- B) Daemon thread
- C) Thread with dead state
- D) None of the above

ANSWER : Option B

39) Which of the following is used to obtain length of String object?

A.length

B. size()

[C. capacity\(\)](#)

[D. length\(\)](#)

Answer : Option D

40) What is the return type of split() in the string class

A) String

B) int

C) String []

D) boolean

Answer : Option C

41) What is the final vale of s1

```
String s1 = "one";
```

```
s1.concat("two");
```

```
System.out.println(s1);
```

A) onetwo

B) one

C) twoone

D) two

Answer : Option B

41) What is the final vale of s1

```
String s1 = "hyderabad";
```

```
String s2 = s1.replace("bad1", "abc");
```

```
System.out.println(s2);
```

A) hyderaabc

B) hyderaabc1

C) hyderabad1

D) hyderabad

Answer : Option D

42) In the following what are the final values of s1 and s2

```
StringBuffer s1=new StringBuffer("one");
```

```
StringBuffer s2 = s1.append("two");
```

A) s1 value is "one" and s2 value is "two"

B) both contains same

C) s1 value is "one" and s2 value is "two"

D) s1 value is "one" and s2 value is "onetwo"

Answer : Option B

43) Which of the following are incorrect form of StringBuffer class constructor?

- A. StringBuffer()
- B. StringBuffer(int size)
- C. StringBuffer(String str)
- D. StringBuffer(int size , String str)

Answer: Option D

44) . Which of these process occur automatically by java run time system?

- A. Serialization
- B. Garbage collection
- C. File Filtering
- D. All of the mentioned

Answer: Option A

Explanation:

Serialization and deserialization occur automatically by java run time system, Garbage collection also occur automatically but is done by CPU or the operating system not by the java run time system.

45) Which of these is a method of ObjectOutput interface used to finalize the output state so that any buffers are cleared?

- A. clear()
- B. flush()
- C. fflush()
- D. close()

Answer: Option B

Explanation:

flush()

46) Which of these is method of ObjectOutput interface used to write the object to input or output stream as required?

- A. write()
- B. Write()
- C. StreamWrite()
- D. writeObject()

Answer: Option D

Explanation:

writeObject() is used to write an object into invoking stream, it can be input stream or output stream.

47) Which of these is a process of extracting/removing the state of an object from a stream?

- [**A.** Serialization](#)
- [**B.** Externalization](#)
- [**C.** File Filtering](#)
- [**D.** Deserialization](#)

Answer: Option D

Explanation:

Deserialization is a process by which the data written in the stream can be extracted out from the stream.

48) Which of these is a method of ObjectInput interface used to deserialize an object from a stream?

- [**A.** int read\(\)](#)
- [**B.** void close\(\)](#)
- [**C.** Object readObject\(\)](#)
- [**D.** Object WriteObject\(\)](#)

Answer: Option C

Explanation:

Object readObject()

49) When does Exceptions in Java arises in code sequence?

- [**A.** Run Time](#)
- [**B.** Compilation Time](#)
- [**C.** Can Occur Any Time](#)
- [**D.** None of the mentioned](#)

Answer: Option A

Explanation:

Exceptions in java are run-time errors.

50) Which of these keywords is not a part of exception handling?

- A. try
- B. final
- C. throws
- D. catch

Answer: Option B