
## ***Core java syllabus***

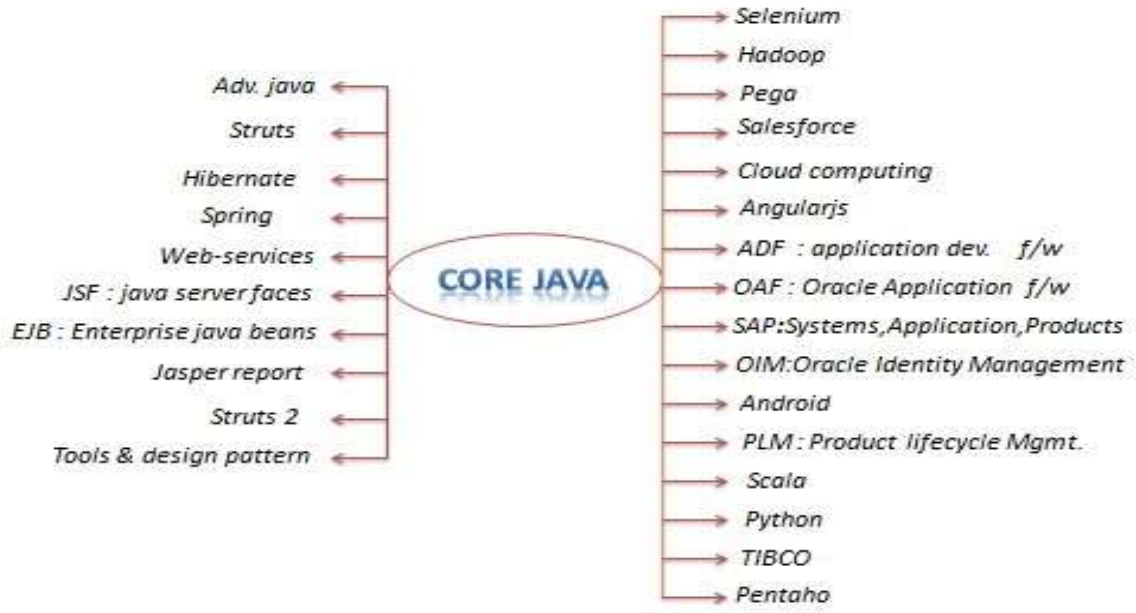
***JAVA introduction:-*** ***{class-1}***

| | | |
|---|---|---|
| *Author* | : | *James Gosling* |
| *Vendor* | : | *Sun Micro System(**which has since merged into Oracle Corporation**)* |
| *Initial Name* | : | *OAK language* |
| *Present Name* | : | *java* |
| *Initial version* | : | *jdk 1.0 (java development kit)* |
| *Present version* | : | *java 8 2014* |
| *Type* | : | *open source & free software* |
| *Extensions* | : | *.java & .class & .jar* |
| *Operating System* | : | *multi Operating System* |
| *Implementation Lang* | : | *c, cpp……* |
| *Symbol* | : | *coffee cup with saucer* |
| *SUN* | : | *Stanford Universally Network* |
| *Slogan/Motto* | : | *WORA(write once run anywhere)* |
| *Compilation* | : | *java compiler* |
| *Execution* | : | *JVM(java virtual machine)* |

***Importance of core java: -*** *According to the SUN 3 billion devices run on the java language only.*

1) *Java is used to develop Desktop Applications such as MediaPlayer,Antivirus etc.*
2) *Java is Used to Develop Web Applications such as sravyajobs.com, irctc.co.in etc.*
3) *Java is Used to Develop Enterprise Application such as Banking applications.*
4) *Java is Used to Develop Mobile Applications.*
5) *Java is Used to Develop Embedded System.*
6) *Java is Used to Develop SmartCards.*
7) *Java is Used to Develop Robotics.*
8) *Java is used to Develop Games ……..**etc.***

***Importance of java****:Technologies& frame works Depends on Core java*



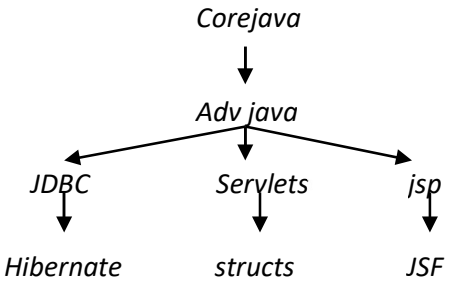

***<u>Important overview of technologies & frameworks& servers & database:-</u>***


***Parts of the java:-***

*Core java & adv. java not official words, As per the **sun micro system** standard the java language is divided into three parts.*

1) *J2SE/JSE(java 2 standard edition)*
2) *J2EE/JEE(java 2 enterprise edition)*
3) *J2ME/JME(java 2 micro edition)*

***Learning process of java:-***



*Web services*

*Spring*

| ***Version Name*** | ***Code Name*** | ***Release Date*** |
|---|---|---|
| *JDK 1.0* | *Oak* | *23 January 1996* |
| *JDK 1.1* | *(none)* | *19 February 1997* |
| *J2SE 1.2* | *Playground* | *4 December 1998* |
| *J2SE 1.3* | *Kestrel* | *8 May 2000* |
| *J2SE 1.4* | *Merlin* | *13 February 2002* |
| *J2SE 5.0* | *Tiger* | *29 September 2004* |
| *Java SE 6* | *Mustangs* | *11 December 2006* |
| *Java SE 7* | *Dolphins* | *28 July 2011* |
| *Java SE 8* | *(Not available)* | *18 March 2014* |

***Different types of languages:-***

1) *Scripting languages : JS,php,python, perl ,vbscript*
2) *Procedural languages : BCPL , Pascal. Fortran , COBOL*
3) *Structured Programming languages : ALGOL, Pascal, Pl/I, C , Ada*
4) *Object-oriented programming : java*
5) *Object based language : Java Script*
6) *Functional programming language. : Scala ,python*

*else switch case default break for while do continue*

***Java keywords :-( 50)*** **Data Types** *byte short int long float double char boolean* **(8)** ***(10)***

***Predefined constants***

***Reserved words (53):-***

***Flow-Control:-***

*if*

***Separators in java:-*** ***Symbol*** ***name*** *Keywords (50) + constants (3) = Reserved Words (53)* ***{Class-2}***

***method-level:-***
void return
***(2)***

***Object-level:-*** *new*
*this super*
*instanceof*
**(4)**

***source-file:***
*class extends*
*interface*
*implements*
*package*
*import*
***(6)***

*True, false, null (3)*

**Exception *handling:-***
*try*
*catch*
*finally throw*
*throws*
***(5)***

***1.5 version:-***
*enum*
*assert*
***(2)***

***unused:-***
*goto*
*const*
***(2)***

***usage***

***Modifiers:-***
public
private
protected
abstract
final static
strictfp
native
transient
volatile
synchronized
**(*11*)**

| | | |
|---|---|---|
| ***( )*** | *parentheses* | *used to contains list of parameters & contains expression.* |
| ***{ }*** | *braces* | *block of code for class, method, constructors & local scopes.* |
| ***[ ]*** | *brackets* | *used for array declaration.* |
| ***;*** | *semicolon* | *terminates statements.* |
| ***,*** | *comma* | *separate the variables declaration & chain statements in for.* |
| *.* | *period* | *used to separate package names from sub packages. And also used for separate a variable,method from a reference type.* |

***Java Comments:-***

- ✓ *Comments are used to write the detailed description about application logics to understand the logics easily.*
- ✓ *When we write the comments the application maintenance will become easy.*
- ✓ *Comments are non-executable code these are ignored during compilation.*

*There are 3 types of comments.*

1) ***Single line Comments: -****Possible to write the description in single line.*

*Syntax:- //description*

2) ***Multi line Comments****: -To write the description in more than one line. Starts **/*** ends ***/***

*Syntax: - /* satement-1*
*Statement-2*
*;;;;;;;;;;;;;;*
*Statement-n*
**/*

***3)Documentation Comments: -*** ***U****sed to prepare API documents.*

*Syntax: -*
```
/*
*statement-1
*statement-2
*/
```

***API****(Application programming interface) :*

- ✓ *It contains detailed description about how to use java product.*
- ✓ *It is a interface between end-user & product.*

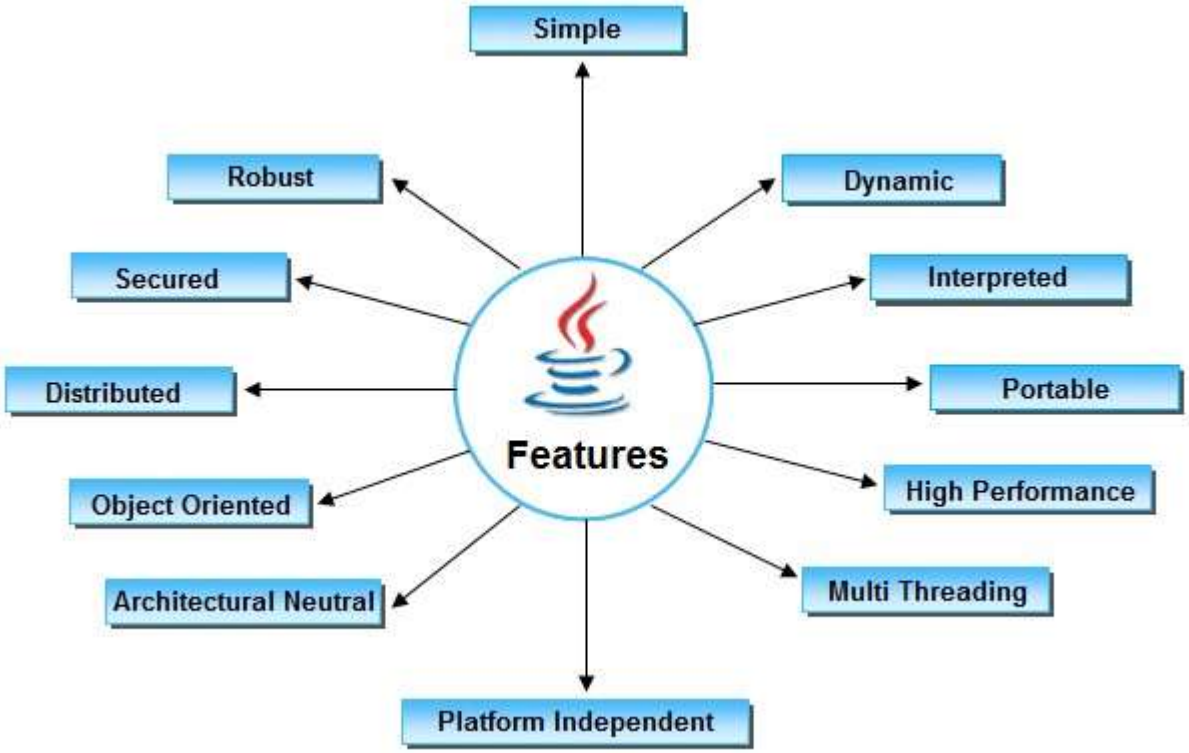
***Downloading Api document:-*** *it contains detailed description about how to use java product.*

*To download java api document use fallowing link [http://docs.oracle.com/javase/8/docs/](http://docs.oracle.com/javase/8/docs/)*


*click on JDK 8 documentation then you will get below page.*

*Accept the license agreement and download the file.*

***JAVA Features :-( Buzz words)***




*1.* ***Simple***

- ✓ *Java technology has eliminated all the difficult and confusion oriented concepts like pointers, multiple inheritance in the java language.*
- ✓ *Java uses c,cpp syntaxes mainly hence who knows C,CPP for that java is simple language.*

***2. Object Oriented***

- ✓ *Java is object oriented because it is representing total data of the class in the form of object.*
- ✓ *The languages which are support object,class,Inheritance,Polymorphism, Encapsulation,Abstraction those languages are called Object oriented.*

***3. Robust***

*Any technology good at two main areas that technology is robust technology.*

a. *Exception Handling*
b. *Memory Allocation*

*Java is providing predefined support to handle the exceptions.*

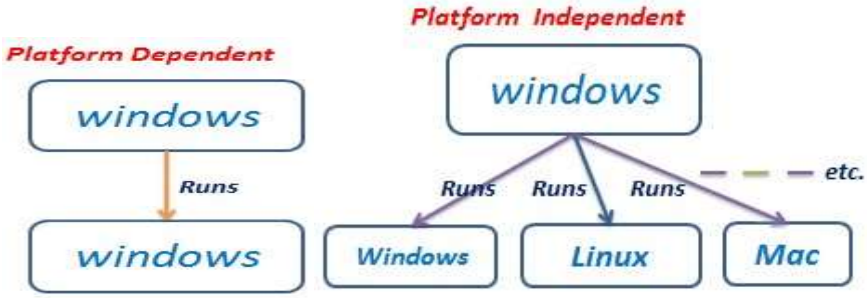
*Java provides Garbage collector to support memory management.*

**4. *Architectural Neutral***

*Java applications are compiled in one Architecture/hardware (RAM, Hard Disk) and that Compiled program runs on any architecture(hardware) is called Architectural Neutral.*

**5. *Platform Independent***

- ✓ *Once we develop the application by using any one operating system(windows) that application runs only on same operating system is called platform dependency. Ex :- C,CPP*
- ✓ *Once we develop the application by using any one operating system(windows) that application runs on in all operating system is called platform independency. Ex :- java*




**6. *Portable***

*In Java the applications are compiled and executed in any OS(operating system) and any Architecture(hardware) hence we can say java is a portable language.*

**7. *Secure***

*To provide security Java provides one component inside JVM called Security Manager. To provide security for the Java applications **java.security** package.*

**8. *Dynamic***

*Java is dynamic language at runtime the memory is allocated.*

**9. *Distributed***

*By using java it is possible to develop distributed applications by using RMI,EJB…etc*

**10. *Multithreaded***

*Executing more than one thread simultaneously is called multithreading.*

*Main advantage of multithreading is used to develop multimedia, gaming, web application.*

***11. High Performance***

*Java support features like Robust, Security, Platform Independent, Dynamic and so on then that technology is high performance.*

***12. Interpreted*** *JVM mostly uses interpreter to convert byte code to machine dependentcode.*

***{Class-3}***

***<u>Java coding conventions:-</u>***

***Classes***

- ✓ *Class name start with upper case letter and every inner word starts with upper case letter.*
- ✓ *This convention is also known as **camel case** convention.*
- ✓ *The class name should be nouns.*

Example:- **S**tring **S**tring**B**uffer **I**nput**S**tream**R**eader ……etc

***Interfaces*** *starts with upper case and every inner word starts with upper case letter.*

*Example: **S**erializable **C**loneable **R**andom**A**ccess…etc*

***Enum*** *starts with upper case letter and every inner word are starts with capital letter. Example :*

***E**num, **E**lement**T**ype, **R**etention**P**olicy…etc*

***Annotation*** *starts with upper case letter and every inner word are starts with capital letter. Example :**O**verride, **F**unctional**I**nterface …etc*

***Methods***

- ✓ *Method name starts with lower case letter and every inner word starts with upper case letter.* ✓ *This convention is also known as mixed case convention* ✓ *Method name should be verbs.*

*Example:- post() char**A**t() to**U**pper**C**ase()*

***Variables*** *start with lower case letter and every inner word starts with upper case letter. Example :- out in page**C**ontext …etc*

***Package*** *Package name is always must written in lower case letters. Example :- java.lang java.util java.io …etc*

***Keywords*** *While declaring keywords every character should be lower case. Example: try,if,break,continue…..etc*

***Constants:-****While declaring constants all the words are uppercase letters . Example:*

*MAX_PRIORITY MIN_PRIORITY NORM_PRIORITY*

*NOTE:- The coding standards are mandatory for predefined library & optional for user defined library but as a java developer it is recommended to fallow the coding standards for user defined library also.*

***ASCII VS Unicode:-***

| ***ASCII*** | ***Unicode*** |
|---|---|
| *Data representation in C-language* | *Data representation in java language. support only English Support all languages present in the world* |
| *a-z A-Z 0-9 special symbols* | *All languages characters* |
| *char size : 1-byte* | *char size 2-bytes* |
| *a=97 A=65* | *a=97 A=65* |

## Differences between C & CPP & JAVA:-

***C-lang***

```
#include<stdio.h>
Void main()
{   Printf("hi balu");
}
```

*Author:* ***Dennis Ritchie***

*Implementation languages:*
*COBOL,FORTRAN,*BCPL, B…

*In c-lang the predefined support is available in the form of header files.*
***Ex:- stdio.h , conio.h***

*The header files contain predefined functions.*
***Ex:- printf,scanf…..***

*C-lang* *C*

*Header files* *stdio.h*

*Functions* *printf,scanf..etc*

*In above first example we are using* ***printf*** *predefined function that is present in stdio.h header file hence must include that header file by using #include statement.*
***Ex:#include<stdio.h>***

*In C lang program execution starts from main method called by* ***Operating system.***

*To print data use* ***printf()***

***Cpp-lang***

```
#include<iostream.h>
Void main()
{   Cout<<"hello balu";
}
```

Author : ***Bjarne Stroustrup***

*implementation languages:*
c ,ada,ALGOL68…..

*cpp language the predefined is maintained in the form of header files.*
***Ex:- iostream.h***

*The header files contains predefined functions.*
***Ex:- cout,cin….***

*CPP-lang* *CPP*

*Header files* *iostream.h*

*Functions* *cout,cin..etc*

*In above first example we are using* ***cout*** *predefined function that is present in stdio.h header file hence*

*must include that header file by using #include statement.*
***Ex:#include<stdio.h>***

*In C lang program execution starts from main method called by* ***Operating system.***

*To print data use* ***cout***

***Java –lang***

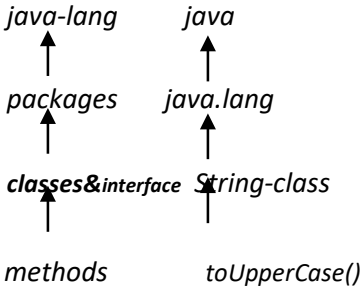
```
Import  java.lang.System;
Import  java.lang.String;
Class Test
{ Public static void main (String [] args)
    {System.out.println ("hi java");
     }
}
```

*Author :* ***James Gosling***

*implementation languages*
C,CPP,ObjectiveC…

*In java predefined support is available in the form of packages.*
***Ex: java.lang, java.io,java.awt***

*The packages contains predefined classes&interfaces and these class & interfaces contains predefined methods.*
***Ex:- String,System***




*in aboveexampe we are using two classes(String,System) these classes are present in* ***java.lang*** *package must import it by using import keyword.*
***a) Import java.lang.****; all lasses*
***b)Import java.lang.System; required***
***Import java.lang.String; classes***
***In above two approachaes 2nd good***

*In java execution starts from main called by JVM*
*To print data use* ***System.out.println()***

***Types of java applications:-***

1. ***Standalone applications:***
   - ✓ *It is also known as window based applications or desktop applications.*
   - ✓ *This type of applications must install in every machine like media player, antivirus …etc ✓ By using AWT & Swings we are developing these type of applications.*
   - ✓ *This type of application does notrequired client-server architecture.*

2. ***Web applications:***
   a. *The applications which are executed at server side those applications are called web applications like Gmail, facebook ,yahoo…etc .*
   b. *All applications present in internet those are called web-applications.*
   c. *The web applications required client-server architecture.*
      i. *Client : who sends the request.*
      ii. *Server : it contains application & it process the app & it will generate response.*
      iii. *Database : used to store the data.*
   d. *To develop the web applications we are using servlets,structs,spring…etc*

3. ***Enterprise applications:-***
   - ➢ *It is a business application &most of the people use the term it I big business application.*
   - ➢ *Enterprise applications are used to satisfy the needs of an organizationrather than individual users. Such organizations are business, schools, government …etc ➢ An application designed for corporate use is called enterprise application.*
   - ➢ *An application in distributed in nature such as banking applications.*
   - ➢ *All j2ee &EJB is used to create enterprise application.*

4. ***Mobile applications:-***

- ✓ *The applications which are design for mobile platform are called mobile applications.*
- ✓ *We are developing mobile applications by sing android,IOS,j2me…etc*
- ✓ *There are three types of mobile applications ○ Web-application (gmai l ,online shopping,oracle …etc)*
  - *Native (run on device without internet or browser)ex:phonecall,calculator,alaram,games These are install from application store& to run these apps internet not required.*
  - *Hybrid (required internet data to launch) ex:whats up,facebook,LinkedIn…etc*
    *These are installed form app store but to run this application internet data required.*

5. ***Distributed applications:-***

   *Software that executes on two or more computers in a network. In a client-server environment.Application logic is divided into components according to function.*
   *Ex : aircraft control systems,industrial control systems,network applications…etc*

***Install the software and set the path:***

1) *Download the software.*
2) *Install the java software in your machine. 3) Set the environmental variable.*

***Step 1: Download the software:-***

- ✓ *Download the software based on your operating system& processor because the software is different from operating system to operating system & processor to processor. Open the Google site type the jdk8 download as shown below.*


*After clicking above link we will get below window then accept license agreement by clinking radio button then choose the software based on your operating system and processor to download.*

**Java SE Development Kit 8u60**

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement  Decline License Agreement

| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM v6/v7 Hard Float ABI | 77.69 MB | jdk-8u60-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM v8 Hard Float ABI | 74.64 MB | jdk-8u60-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 154.66 MB | jdk-8u60-linux-i586.rpm |
| Linux x86 | 174.83 MB | jdk-8u60-linux-i586.tar.gz |
| Linux x64 | 152.67 MB | jdk-8u60-linux-x64.rpm |
| Linux x64 | 172.84 MB | jdk-8u60-linux-x64.tar.gz |
| Mac OS X x64 | 227.07 MB | jdk-8u60-macosx-x64.dmg |
| Solaris SPARC 64-bit (SVR4 package) | 139.67 MB | jdk-8u60-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 99.02 MB | jdk-8u60-solaris-sparcv9.tar.gz |
| Solaris x64 (SVR4 package) | 140.18 MB | jdk-8u60-solaris-x64.tar.Z |
| Solaris x64 | 96.71 MB | jdk-8u60-solaris-x64.tar.gz |
| Windows x86 | 180.82 MB | jdk-8u60-windows-i586.exe |
| Windows x64 | 186.16 MB | jdk-8u60-windows-x64.exe |

*For 32-bit operating system please click on Windows x86*
*For 64-bit operating system please click on Windows x64*

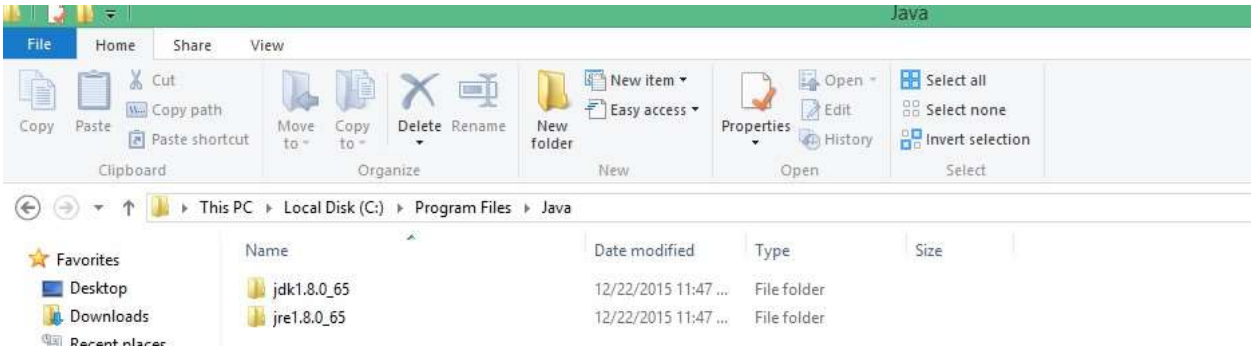
***Step 2: Install the java software***

- ✓ *Install the java software just like media players in your machine.*
- ✓ *After installing the software the installation java folder is available in the fallowing location by default. (But it possible to change the location at the time of installation).*

  ***Local Disk c: --->program Files--->java***

  ***Jdk :*** *java development kit*

***Jre:*** *java runtime environment*



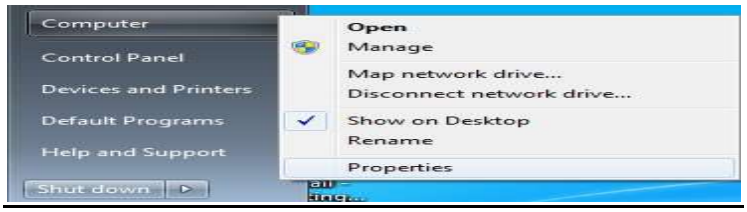

***Step 3: set the path***

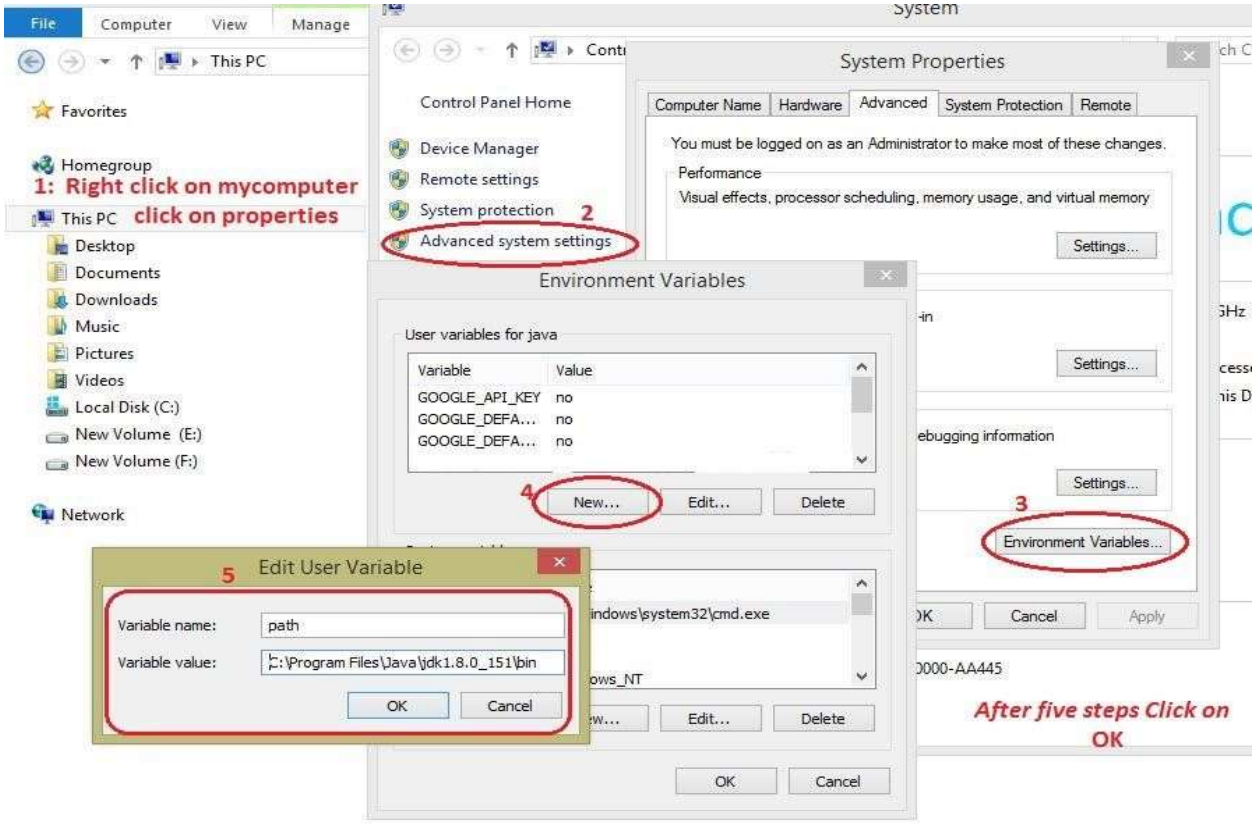
*In C language we can run the application only form turbo-C folder but in java it is possible to run the files in any local disk (D: , E: , F: …etc.)when we set the path.*

***Right click on mycomputer --->properties----->Advanced system setting--->Environment Variables --User variables--->new----->***

***variable name : path***

***Variable value : C:\programfiles\java\jdk1.8.0_11\bin;***




*After path setting open the command prompt type javac command then you will get list of commands then decide java is working properly your system.*

```
C:\Users\Welcome>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
```

## *Steps to Design a First Application:*

*Step-1:* ***Select the Editor.***

*Step-2:* ***Write the application.***

*Step-3:* ***Save the application.***

*Step-4:* ***Compilation Process.***

*Step-5:* ***Execution process.***

***Step1: Select an Editor***

*Editor is software it will provide very good environment to develop application.*
*ex :- Notepad, Notepad++,edit Plus…..etc*

***IDE:*** *( Integrated development Environment )*
*IDE is providing very good environment to develop the application.*
*ex : Eclipse,MyEclipse,Netbeans,JDeveloper….etc*

*In IDE the code is automatically generated like,* ✓ *Automatic compilation.*

- ✓ *Automatic package import.*
- ✓ *It shows all the predefined methods of classes.*
- ✓ *Automatically generate try catch blocks and throws (Exception handling)……..etc*

***Note: do the practical's of corejava by using edit plus editor.***

***Step 2: Write a program in edit plus.***

*Open editplus --->file -->new --->click on java (it display java application)*

```
import java.lang.System;
import java.lang.String;
class Test
{       public static void main(String[] args)
        {       System.out.println("Balu world");
        }
}
class A{
}
class B{
}
```

*In above example we are using two predefined classes(**String , System) &** these are present in java.lang package hence must import that package by using import statement.*

*There are two approaches to import the classes in java, 1)*
*Import all classes of particular package.*
*a. Import java.lang.*;*

*2) Importing application required classes.*

a. *Import java.lang.System;*
b. *Import java.lang.String;*

***Step3: save the application.***

- ✓ *Save the java application by using **(.java)** extension.*
- ✓ *While saving the application must fallow two rules ○ If the source file contains public class then must save the application by using public class-name **(publicClassName.java)**. Otherwise compiler generate error message.*

  ○ *if the source file does not contain public class then save the application by using any name**(anyName.java)** like A.java , Balu.java, Anu.java …….etc.*

***Note: -*** *The source file is allows to declare only one public class, if we are trying to declare more than one public class then compiler generate error message.*

***Case 1: valid : Test.java***

```
public class Test
{   };
class A
{   };
```

***Case 2:- valid***

***Test.java*** *public*

```
class Test
{   };
class A
{};
```

***Case 3:- invalidTest.java***

```
public class Test
{};
public class A
{};
```

***Application location:-***

*D: (any disk)*

*|-->balu (any folder)*
*|-->Sravya.java (your file name)*

***Step-4:- Compilation process.*** *Compile the java application by using **javac**command. **Syntax:-***

*Javac filename*

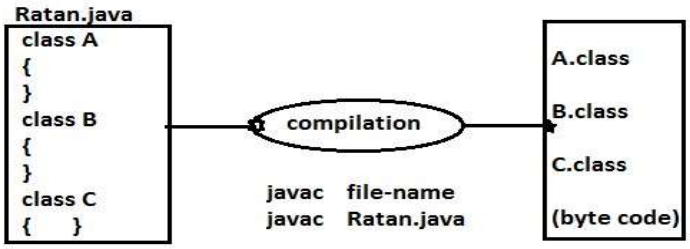
*Javac Test.java*

*Open the command prompt then move to your application location:-*

***C:\Users\hp>*** *initial cursor location*
***C:\Users\hp>d:*** *move to local disk D*
***D:\>cd balu*** *changing directory to balu*
***D:\balu>javac Sravya.java*** *compilation process*

***Compiler responsibilities:-***

- *compiler check the syntax errors , If the application contains syntax errors then compiler will generate error message in the form of compilation error.*
- *If the application does not contain syntax errors then compiler translate **.java** to **.class** file. **Note:** - In java compiler generate .class files based on number of classes present in source file.*
  *If the source file contains 100 classes after compilation compiler generates 100 .class files*




***Step-5:- Execution process.***

*Run /execute the java application by usingjavacommand.*

***Syntax:-*** *Java class-name*

*Java Test*

***JVM responsibilities:-***

- ✓ *JVM wills loads corresponding .class file byte code into memory.*
- ✓ *After loading .class file JVM calls main method to start the execution process.*

***D:\balu>java Test***

*Hi Balu*

***D:\balu>java A***

*Error: Main method not found in class A, please define the main method as:*

***D:\balu>java B***

*Error: Main method not found in class B, please define the main method as:*

***D:\balu>java XXX***

*Error: Could not find or load main class XXX*

***Note points:-***

- ✓ *compiler is a translator it is translating **.java** file to **.class**whereas JVM is also a translator it is translating **.class** file to **machine code.***
- ✓ *Compiler understandable file format is **.java** file but JVM understandable file format is**.class** file.*
- ✓ *It is possible to compile multiple files at a time but it is possible to execute one .class file at a time.*
- ✓ *The .java file contains high level language but .class file contains byte code instructions it is a platform independent code.*
- ✓ *Java is a platform independent language but JVM is platform dependent.*

***Process of compiling multiple files:-***

*D:*

*|-->balu*

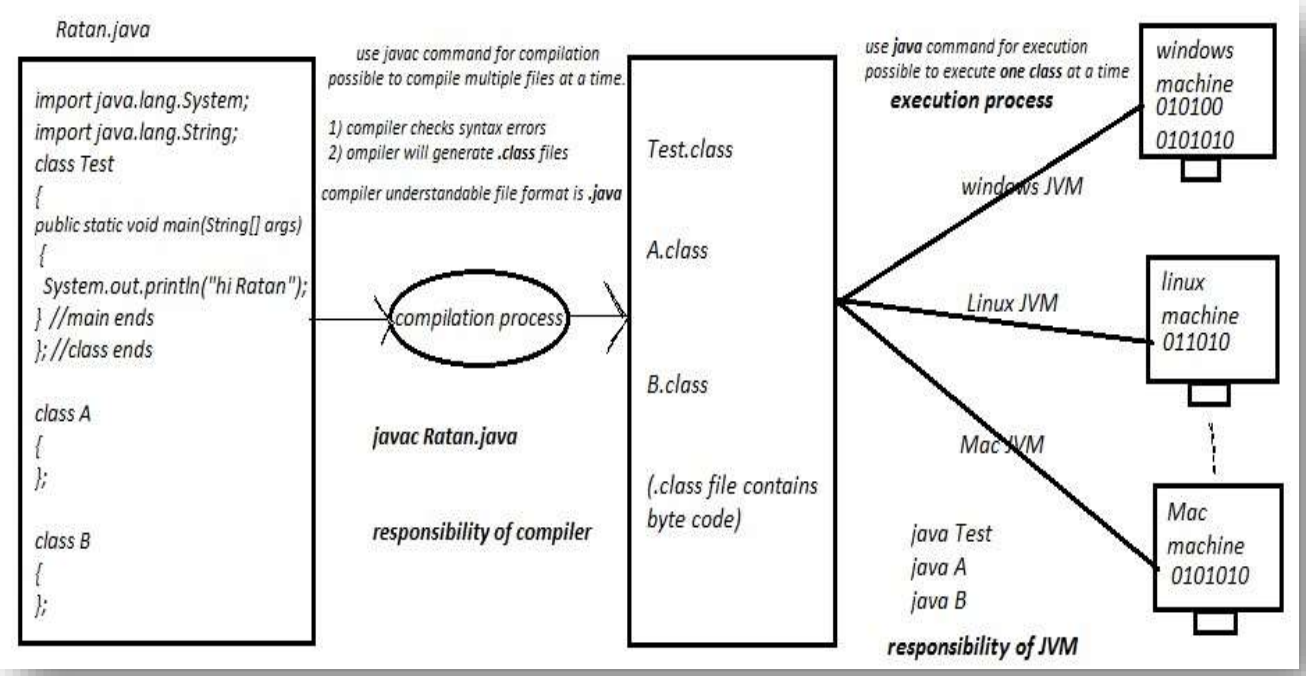
*|-->A.java*

*|-->B.java |-->C.java javac A.java one file is compiled(A.java)*

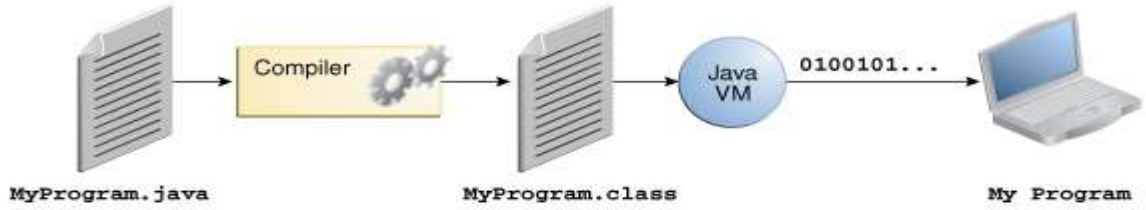
*javac B.java C.java two files are compiled javac *.java all files are compiled*

*javac Emp*.java files prefix with emp compiled (EmpId.java EmpName.java…) javac *Emp.java files suffix with emp compiled (XEmp.java YEmp.java…)*

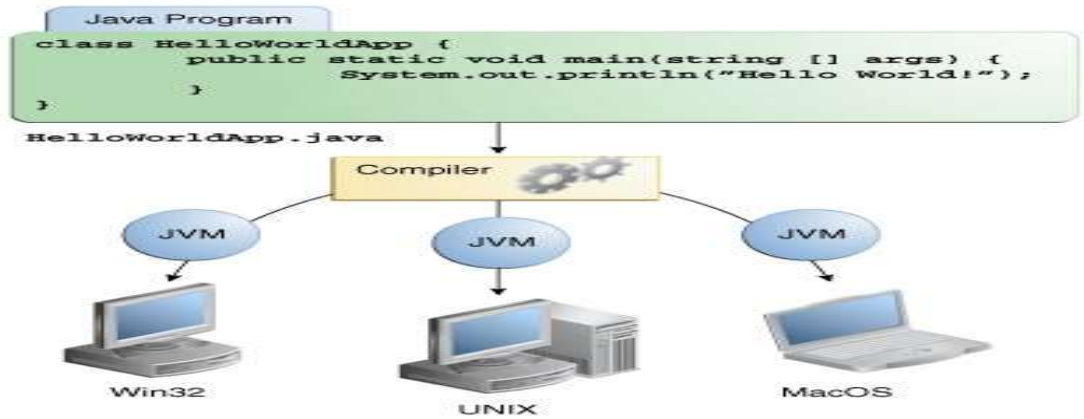
***Overview of first application***




***Environment of the java programming development:-***




***First program development :-***



***First application conclusions :-***

***Conclusion 1:-****Java contains 14 predefined packages but the default package in java is* ***java.lang***

***Conclusion -2:-****The source file is allows declaring only one public class, if you are declaring more than one public class compiler generating error message.*

```
public class Test
{
}
public class A
{
}
```

***error:*** *class A is public, should be declared in a file named A.java*

***Conclusion-3***

- ✓ *The coding conventions are mandatory for predefined library & optional for user defined but as a java developer must fallow the coding conventions for use defined library also.*
- ✓ *The below example compiled & executed but it is not recommended because the class name starting with lower case letters. class test*

```
{       public static void main(String[] args)
        {       System.out.println("Balu World!");
        }
}
G:\>java test Balu
World!
```

***Conclusion -4:-****The class contains main method is called* ***Mainclass*** *and java allows declaring multiple main class in a single source file.*

```
class Test
{       public static void main(String[] args)
        {       System.out.println("Test1 World!");
        }
}
class A
{       public static void main(String[] args)
        {       System.out.println("A World!");
        }
}
class B
```

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

***D:\morn11>java Test1*** ***D:\morn11>java Test2*** ***D:\morn11>java Test3***

***Test1 World!*** ***Test2 World!*** ***Test3 World!***

***{Class-5}***

***Print() vs Println ():-***

***Print():-****This method used to print the data after printing control present in same line.*

***Println():-****This method used to print the data after printing control present in new line.*

```
class Test
{       public static void main(String[] args)
        {       System.out.print("balu");
                System.out.print("anu");
                System.out.println("durga");
                System.out.println("sravya");
        }
}
```

*baluanudurga sravya*

***Escape Sequences:-*** *A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler. The following table shows the Java escape sequences*

**Escape Sequences**

| Escape Sequence | Description |
|---|---|
| \t | Insert a tab in the text at this point. |
| \b | Insert a backspace in the text at this point. |
| \n | Insert a newline in the text at this point. |
| \r | Insert a carriage return in the text at this point. |
| \f | Insert a formfeed in the text at this point. |
| \' | Insert a single quote character in the text at this point. |
| \" | Insert a double quote character in the text at this point. |
| \\ | Insert a backslash character in the text at this point. |

```
class Test
{       public static void main(String[] args)
        {       System.out.println("hi \"balu\" sir");
        System.out.println("hi \'balu\' sir");
                System.out.println("hi \\balu\\ sir");
                System.out.println("hi\tbalu\tsir");
                System.out.println("hi\nbalu\nsir");
```

```
    }
}
```

***Class elements:-****The java class contains 5-elements*

```
class Test
{       variables
        methods
        constructors
        instance blocks
        static blocks
}
```

***Java Tokens:-****Smallest individual part of a java program is called Token &It is possible to provide* ***n*** *number of spaces in between two tokens.*

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

***Tokens are*****----------→***class,test,{,",[ ……..etc*

***Java identifiers:-***

*Every name in java is called identifier such as,Class-name,Method-name,Variable-name…etc* ***Rules to declare identifier:***

1. *An identifier contains group of Uppercase & lower case characters, numbers, underscore& dollar sign characters but not start with number.*

   *int abc=10; --->valid  int _abc=30;--->valid  int $abc=40;--->valid*
   *int a-bc=50;--->not valid  int 2abc=20; ---> Invalid  int not/ok=100 --->invalid*

2. *Java identifiers are case sensitive of course java is case sensitive programming language. The below three declarations are different & valid.*

```
class Test
{       int a=10;
        Int A=20;
};
```

3. *Theidentifier must be unique.(duplicates are not allowed)*

```
Class   A{}
Class   A{}
```

4. *It is possible to use predefined class names & interfaces names as a identifier but it is not recommended.*

```
class Test
{       public static void main(String[] args)
        {       int String=10;
        intSerializable=20;
                System.out.println(String);
                System.out.println(Serializable);
        }
};
```

5. *It is not possible to use keywords as a identifiers.*

```
int if=10;          int try=20;    //Invalid
```

6. *There is no length limit for identifiers but is never recommended to take lengthy names because it reduces readability of the code.*

***Java primitive Data Types:-***

1. *Data types are used to represent type of the variable & expressions.*
2. *Representing how much memory is allocated for variable.*
3. *Specifies range value of the variable.*

*There are 8 primitive data types in java*

| ***Data Type*** | ***size(in bytes)*** | ***Range*** | ***default values*** |
|---|---|---|---|
| *byte* | *1* | *-128 to 127* | *0* |
| *short* | *2* | *-32768 to 32767* | *0* |
| *int* | *4* | *-2147483648 to 2147483647* | *0* |
| *long* | *8*–9,223,372,036,854,775,808 to 9 ,223,372,036,854,775,807 | *0* | |
| *float* | *4* | *-3.4e38 to 3.4e* | *0.0* |
| *double* | *8* | *-1.7e308 to 1.7e308* | *0.0* |
| *char* | *2* | *0 to 6553* | *single space* |
| *Boolean* | *no-size* | *no-range* | *false* |

***Byte :-***

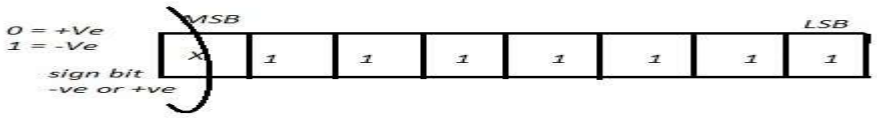
*Size : 1-byte*

*MAX_VALUE : 127*

*MIN_VALUE : -128*

*Range : -128 to 127*

*Formula : $-2^n$ to $2^{n-1}$ $-2^8$ to $2^{8-1}$*




***Note :-***

- *To represent numeric values (10,20,30…etc) use **byte,short,int,long.***
- *To represent decimal values(floating point values 10.5,30.6…etc) use **float,double.***
- *To represent character use **char** and take the character within single quotes.*
- *To represent true ,false use **Boolean.***

*Except Boolean and char remaining all data types consider as a signed data types because we can represent both +ve & -ve values.*

***Float vs double:-***

*Float will give 5 to 6 decimal places of accuracy but double gives 14 to 15 places of accuracy.*
*Float will fallow single precision but double will fallow double precision.*

***Syntax:- data-type name-of-variable=value/literal;***

*Ex:- int a=10;*

*Int ------→ Data Type*
*a ------→ variable name*
*= ------→ assignment*
*10 ------→ constant value*
*; ------→ statement terminator*

***printing variables :-***

*int a=10;*

*System.out.println(a); //valid*
*System.out.println("a"); //invalid*
*System.out.println('a'); //invalid*

***User provided values are printed*** *int a = 10;System.out.println(a);//10 double d=10.5;System.out.println(d);//10.5 char ch='a'; System.out.println(ch);//a boolean b=true;System.out.println(b);//true*

```
                System.out.println(f);
                double d=20.5;
                System.out.println(d);
        }
}
```

*D:\balu>javac Test.java*
*Test.java:3: error: possible loss of precision required: float found: double*

***Example :-***

```
class Test
{       public static void main(String[] args)
          {       float f=10.5;
```

***Default values(assigned by JVM)*** int *a;*
*System.out.println(a);//0*
*double d; System.out.println(d);//0.0*
*char ch; System.out.println(ch);//single space boolean b;*
*System.out.println(b);//false*

- ✓ *In java the decimal values are by default double values hence to represent float value use **f** constant or perform type casting.*

```
float f =10.5f;    //using f constant (valid) float f
=(float)10.5;      //using type casting (valid)
```

***String :-***

- ✓ *String is not a data type & it is a class present in java.lang package to represent group of characters or character array enclosed with in double quotes.* ✓ *The default value of the String is null* ✓ *For any class type the default value Is null.*

```
String ename="balu";
System.out.println(ename);

String s;
System.out.println(s); //null
```

## *Binary literals:*

*Java 7 allows you to express integral types (byte, short, int, and long) in binary number system. To specify a binary literal, add the prefix 0b or 0B to the integral value.*

```
        public
class Test {
   public static void main(String[] args) {        byte b1 = 0b101;    // Using b0,
        The b can be lower or upper case        short s1 = 0b111;
              int i1 = 0b100;
              long l1 = 0b0000011111100001;
System.out.println("b1 = "+b1);
                    System.out.println("s1 = "+s1);
                   System.out.println("i1 = "+i1);
                   System.out.println("l1 = "+l1);
   }
}
```

*Ex:*

```
public class
Test {
   public static void main(String[] args) {
      byte b1 = 5;                              // a decimal value
      byte b2 = 0b101;              // using b0, The b can be lower or upper case
byte b3 = -0b101;          byte b4 = 0b101_0;
      System.out.println("b1 = "+b1);
      System.out.println("b2 = "+b2);
      System.out.println("b3 = "+b3);
      System.out.println("b4 = "+b4);
      System.out.println("is b1 and b2 equal: "+(b1==b2));
      System.out.println("b2 + 1 = "+(b2+1));
      System.out.println("b3 + 1 = "+(b3+1));
      System.out.println("b4 x 2 = "+(b4*2));
   }
}
```

## *Underscores in numeric literals:*

*With Java 7, you can include underscores in numeric literals to make them more readable. The underscore is only present in the representation of the literal in Java code, and will not show up when you print the value.*

```
public class Test
{       public static void main(String[] args)
        {       int val1 = 10000000;                            //10 Million
                System.out.println("Amount is :" +val1);

                int val2 = 10_000_000;                  //10 Million  : more readable format
                System.out.println("Amount is :"+val2);
        }
}
```

*1. Consecutive underscores is legal.*

```
int n = 1000_____000;
```

*2. Underscores can be included in other numeric types as well.*

```
double d = 1000_000.0;
long l = 1000_000l;
int hex = 0xdead_c0de;
int bytes = 0x1000_0000;
```

*3. Underscore can be included after the decimal point.*

```
double example8 = 1000_000.000_000d;
```

1. *You cannot use underscore at the beginning or end of a number.*

```
int a = _10;       // Error, this is an identifier, not a numeric literal   int a
= 10_;   // Error, cannot put underscores at the end of a number
```

2. *It is also illegal to have underscore be just before or after a decimal point.*

```
float a = 10._0; // Error, cannot put underscores adjacent to a decimal point
float a = 10_.0; // Error, cannot put underscores adjacent to a decimal point
```

***Java basics interview questions***

1) *What is importance of java?*
2) *Why all the technologies & frame works depends on java?*
3) *What is the difference between open sources & licensed give some examples?*
4) *What do you mean by platform dependent & platform independent?*
5) *How many parts of java?*
6) *What is present version of java & initial version of java & symbol of java?*
7) *What is initial name of java and present name of java?*
8) *How many reserved words in java & how many keywords & how many constants?*
9) *Null is keyword or not?*
10) *What is the purpose of the commands & commands are executable code or not?*
11) *How many types of commands in java?*
12) *What do you mean by API document?*
13) *What is he difference between header files & packages?*
14) *Author of C,CPP,java ?*
15) *What is the difference between #include & import statement?*
16) *What is difference between ASCII & UNICODE?*
17) *What is the difference between editor & IDE?*
18) *What the difference between path & class-path?*
19) *Give some points about jdk & jre?*
20) *What is the difference between path and class path?*
21) *What is the purpose of environmental variables setup?*
22) *What are operations done at compilation time and execution time?*
23) *What is the purpose of JVM?*
24) *JVM platform dependent or independent?*
25) *Can we have multiple public classes in single source file?*
26) *What is main class & is it possible to declare multiple main classes in single source file or not?*
27) *What do you mean by token and literal?*
28) *What do you mean by identifier?*
29) *Is it possible to declare multiple public classes in single source file?*

30) *What are the coding conventions of classes and interfaces?*
31) *What are the coding conventions of methods and variables?*
32) *In java program starts form which method and who is calling that method?*
33) *What are the commands required for compilation and execution?*
34) *Possible to execute more than one class at time or not?*
35) *The compiler understandable file format & JVM understandable file format?*
36) *Is it possible to provide multiple spaces in between two tokens?*
37) *Class contains how many elements what are those?*
38) *Who is generating .class file and .class files generation is based on?*
39) *What is .class & .java contains (type of code)?*
40) *What is the purpose of data types and how many data types are present in java?*
41) *Who is assigning default values to variables?*
42) *What is the default value of int, char, Boolean, double?*
43) *For the class types what are the default values?*
44) *What is the default package in java?* ********Thank you***********

## *Java flow control Statements* *{2-classes}*

*There are three types of flow control statements in java*

a. *Selection Statements*
b. *Iteration statements*
c. *Transfer statements*

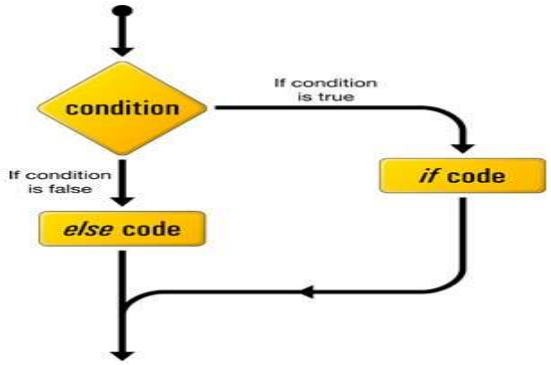
### *1. Selection Statements*

a. *If*
b. *If-else*
c. *Switch*
d. *else-if*

### *If syntax:-*

```
if (condition)                    }
{       true  body;               else
```

***{ false body;***
***}***




***Case-1:*** *Normal example*

```
class Test
{       public static void main(String[] args)
        {       int age=20;
                if (age>22)
                {        System.out.println("if body / true body");
                }
                else
                {       System.out.println("else body/false body ");
                }
        }
}
```

***case -2:For the if the condition it is possible to provide Boolean constants directly.***

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

***Case -3: in c-language 0-false & 1-true but these conventions are not allowed in java.***

```
                if (0)
                {        System.out.println("true body");
                }
                error: incompatible types: int cannot be converted to boolean
```

***case 4:*** *if (a=10)*

```
{       System.out.println("true body");
}
error: incompatible types: int cannot be converted to Boolean
```

***case 5:*** *The curly brasses are options but without curly braces it is possible to take only one statement that should not be a initialization. class Test*

```
{       public static void main(String[] args)
        {       if (true)
                 System.out.println("true body");
                else
                System.out.println("false body");
        }
}
```

***A****: valid*

```
If(true)
SOP("hi balu");
```

***C*** *:Invalid*

```
If(true)
Int x=100;
```

***B*** *: valid*

```
If(true);
```

***D****: valid*

```
If(true)
{Int x=100;}
```

*Note : In java semicolon is empty statement*

***Switch statement:-***

- ✓ *Switch statement is used to declare multiple options.*
- ✓ *Switch is taking the argument, the allowed arguments are* ***byte,short,int,char (primitive data types) Byte, Short,Integer,Character(wrapper calsses) enum(1.5 v) String(1.7 v)***
- ✓ *Inside the switch it is possible to declare more than one case but it is possible to declare only one default.*
- ✓ *Based on the provided argument the matched case will be executed if the cases are not matched default will be executed.*
- ✓ *Float,double,long is not allowed as a switch argument because these are having too large values.*

***Syntax:-***

```
switch(argument)
{  case   label1 :      statements;
                        break;
```

```
    case   label2  :        statements;
                            break;
          |                         |
          |                         |
    default       :         statements;
                            break;
}
```

***Case 1:***

```
class Test {
          publi
c static void
main(String[
] args)
          {         int a=10;
                    switch (a)
                                        {         case
                              10:System.out.println("anushka");         break;
                                        case
                              20:System.out.println("nazriya");
                                        break;
                    default:System.out.println("durga");
                              break;
                    }
          }
}
```

***Case 2 :*** *Inside the switch statement beak is optional. If we are not declaring break statement then from the matched case onwards up to the next break statement will be executed, if there is no break statement then end of the switch will be executed. This situation is called as fall though inside the switch.*

```
class Test
{         public static void main(String[] args)
          {         int a=10;
                    switch (a)
                    {         case 10:System.out.println("10");
                              case 20:System.out.println("40");
                                        break;
                              default: System.out.println("default");    break;
                    }
```

```
        }
};
```

***Case 3 : inside the switch the default is optional.***

```
int a=10;
switch (a)
{       case 10:System.out.println("balu");
}
```

***Case 4: Inside the switch cases are optional part.***

```
int a=10;
switch (a)
{       default: System.out.println("default");
}
```

***Case 5:- inside the switch both cases and default optional.***

```
int a=10;
    switch(a)
    {       }
```

***Case 6: Invalid*** *: Inside the switch independent statements are not allowed. If we are declaring the statements that statement must be inside the case or default.*

```
int x=10;  switch(x)
    {       System.out.println("Hello World");
    }
```

***Case 7 :****In switch it is possible to declare the default at starting or middle or end of the switch.*

```
int a=100;
switch (a)
{       default: System.out.println("default");
        case 10:System.out.println("10");
}
```

***Case 8:*** *Inside the switch the case labels must be unique; if we are declaring duplicate case labels the compiler will raise compilation error "duplicate case label".*

```
int a=10;   switch (a)
```

```
{       case 10:System.out.println("balu");
        case 10:System.out.println("anushka");
}
```

***Case 9 :****Inside the switch for the case labels & switch argument it is possible to provide expressions*

```
int a=100;
switch (a+10)
{       case 110        :System.out.println("balu");
        case 10*5       :System.out.println("durga");
}
```

***Case 10 :*** *The advantage of fall through is used to executed common actions for multiple cases.*

```
int a=2;
switch (a)
{       case 1:
        case 2:
        case 3:System.out.println("Q-1");         break;
        case 4:
        case 5:
        case 6:System.out.println("Q-2");         break;
}
```

***Case 11: Invalid :*** *Inside the switch the case label must be constant values. If we are declaring variables as a case labels the compiler will show error "constant expression required".* *int a=10,b=20;*

```
switch (a)
{       case a:System.out.println("anushka");
        case b:System.out.println("nazriya");
}
```

***Case 12 : Valid***

- ✓ ***I****t is possible to declare final variables as a case label. Because the final variables are constants.*
- ✓ *The final variables are replaced with constants during compilation. final int a=10;*

```
switch (a)
{       case a:System.out.println("anushka");
}
```

***Case 13:***

- ✓ *inside the switch the case label must match with provided argument data type otherwise compiler will raise compilation error "incompatible types".*
- ✓ *In below example we are passing String is a switch argument hence the case labels must be Strings constants.*

```
String str="aaa";
switch (str)
{       case "aaa"      :System.out.println("balu");
        case 'a'        :System.out.println("durga");
}
```

***Case 14 :***

- ✓ ***I****nside switch the case labels must be within the range of provided argument data type otherwise compiler will raise compilation error "possible loss of precision".*
- ✓ *In below example we are passing byte as a switch argument hence the case labels must be within the range of byte.*

```
byte b=127;
switch (b)
{       case 127:System.out.println("balu");
        case 128:System.out.println("anu");
}
```

***Case 15 :internal conversion for unicode.  Unicode values a-97 A-65***

***Case 1*** *: int i=65;*

```
switch (i)
{       case 'A':System.out.println("balu");
}
```

***Case 2*** *: char ch='d';*

```
switch (ch)
{       case 100:System.out.println("durga");
```

```
                }
```

***else-if:-***

```
class Test
{       public static void main(String[] args)
        {       int a=20;
                if (a==10)
                {       System.out.println("ten");
                }
                else if (a==20)
                {       System.out.println("twenty");
                }
                else if (a==30)
                {       System.out.println("thirty");
                }
                else
                {       System.out.println("default condition");
                }
        }
};
```

***<u>Iteration Statements:-</u>***

*By using iteration statements we are able to execute group of statements repeatedly or more number of times.*

1) *For*
2) *For-each*
3) *while*
4) *do-while*

***<u>Flow of execution in for loop:-</u>***

```
          (1)     (2)  (5)      (4)  (7)
for (initialization ;condition  ;increment/decrement )
                                                    {
                  (3)  (6)                            }
        Body;
```

```
                                                    System.out.println("balu");
                                                }
                                            };
```

***Case 2: Inside the for loop initialization part is optional but semicolon is mandatory.***

```
int i=0;
for (;i<10;i++)
{       System.out.println("durga");
}
```

***Case 3 :*** *Initialization part it is possible to take n number of System.out.println("ratna") statements and each and every statement is separated by comma(,) . int i=0;*

```
for (System.out.println("Aruna"),System.out.println("Balu");i<10;i++)
{       System.out.println("anu");
}
```

***Case 4 :*** *Initialization part it is not possible to declare the data type two times whether it is same or different types. But it is possible to declare the single data type with more than one v*

| ***Invalid*** | ***Valid*** |
|---|---|
| for (**int i=0,int j=0** ;i<10;i++) | for (**int i=0,j=0**;i<10;i++) { |
| { System.out.println("balu"); | System.out.println("balu"); |
| } | } |

***<u>Conditional part :</u>***

***Case 1:*** *Inside for loop conditional part is optional, if we are not providing condition at t of compilation compiler will generate true constant.*

```
for (int i=0;;i++)
{       System.out.println("balu");
}
```

***Case 2:*** *error : Unreachable statement*

- ✓ *If the control unable to enter in particular area is called unreachable statement.*
- ✓ *We will get the unreachable code when we declare only Boolean constants(true,false).*
- ✓ *When we will give **true** condition the remaining code is unreachable when we will give **false** condition the body is unreachable.*

***Rest of the code unreachable***

***<u>Initialization part :-</u>***

***Case 1:***

***With out for loop***

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

***loop body is unreachable***

***By using for loop*** *class*

```
Test
{   public static void main(String[] args)
      {   for (int i=0;i<5;i++)
            {       System.out.println("Balu");
            }
      }
}
```

```
for (int i=1;true;i++)          {
System.out.println("balu");
}
System.out.println("rest of the code");
```

***Case 3*** *:- Valid*

```
for (int i=1;false;i++)
{       System.out.println("balu");
}
System.out.println("rest of the code");
```

- ✓ *When you provide the condition even though that condition is represent infinite loop compiler is unable to find unreachable statements,because there may be chance of condition fail.*
- ✓ *When you provide Boolean constants as a condition then compiler is identifying unreachable statement because compiler knows that condition never change.*

```
for (int i=1;i>0;i++ )
{       System.out.println("durga");
}
System.out.println("rest of the app");
```

***Case 4****: method calling return Boolean condition.*

```
class Test
{       static boolean m1()
        {       return true;
        }
        public static void main(String[] args)
        {       for (i=0 ;Test.m1();i++ )
                {       System.out.println("durga");
                }
        }
}
```

***increment/decrement :- case 1 :*** *Inside the for loop increment/decrement part is optional. for (int i=0;i<10;)*

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

***Case 2 :*** *In the increment/decrement it is possible to take the n number of SOP() statements and each and every statement is separated by comma(,). for (int i=0;i<10;System.out.println("aruna"),System.out.println("sravya"))*

```
{       System.out.println("sravya");
```

```
                    i++;
                }
```

***Case 3 :*** *Inside the for loop each and every part is optional but semicolon is mandatory.*
*for(;; ) represent infinite loop because the condition is always true.*

***Case 4: error : unreachable statement***

```
for ( ;  ; )
{       System.out.println("balu");
}
System.out.println("anu");  //error: unreachable statement
```

***For-each loop:-****(introduced in 1.5 version)*

```
class Test
{       public static void main(String[] args)
        {       int[] a={10,20};
                System.out.println(a[0]);
                System.out.println(a[1]);
                //printing data by using for-loop
                for (int i=0;i<a.length;i++)
                {       System.out.println(a[i]);
                }

                //printing data by using for-each loop
        for (int aa: a)
                {       System.out.println(aa);
                }
        }
}
```

***for  loop vs for-each loop :-***

- ✓ *For loop is used to print the data & it is possible to apply conditions.*
- ✓ *For-each loop is used to print the data from starting element to ending element it is not possible to apply the conditions.*

***<u>While loop</u>:-*** ***while (Boolean-expression)*** *//condition must be Boolean & mandatory.*

```
{       body;
}
```

***Case 1:***

```
class Test
```

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

***Case 3***:

```
        while (true)
        {       System.out.println("balu");
        }
        System.out.println("anu");  //error: unreachable statement
```

***Case 4 :*** *Just to check the data & print the data use while loop.*

```
    while (itr.hasNext())
    {   System.out.println(itr.next());
    }
```

***for loop vs while loop :-***

- ✓ *To print the data from specific value to specific value with increment value use for loop.*
- ✓ *Just to check the data & print the data use while loop.*

***Do-While:-***

- ✓ *If we want to execute the loop body at least one time them we should go for do-while statement.*
- ✓ *In do-while first body will be executed then only condition will be checked.*

***Syntax:-***

```
do
{   //body of loop
} while(Boolean-condition);
```

***Case 1:*** *class Test*

```
        public static void main(String[] args)
        {       int i=0;
                do
                {       System.out.println("balu");
                        i++;
                }while (i<10);
        }
}
```

***Case 2:-***

```
        int i=0;
        do
        {       System.out.println("durga");
        }while (false);
        System.out.println("durgasoft");
```

***Transfer statements:- (jump statements)***

*By using transfer statements we are able to transfer the flow of execution from one position to another position.*

- ✓ ***Break***
- ✓ ***continue***
- ✓ ***return***
- ✓ ***try***
- ✓ ***goto***

***break:-*** *Break is used to stop the execution. And is possible touse the break statement only two areas.*

- ✓ ***Inside the switch statement.***
- ✓ ***Inside the loops.***

**Case 1 :** *break is used to stop the execution come out of loop.*

```
class Test
{       public static void main(String[] args)
        {       for (int i=0;i<10;i++)
                {       if (i==5)
                        break;
                        System.out.println(i);
                }
        }
}
```

***Case 2****:* *if we are using break outside switch or loops the compiler will raise compilation error* ***"break outside switch or loop"***

```
if (true)
{       System.out.println("balu");
        break;
        System.out.println("nandu");
}
```

***Continue:*** *it is used skip the current iteration and it is continue the rest of the iterations normally.*

```
class Test
{       public static void main(String[] args)
        {       for (int i=0;i<10;i++)
                {       if (i==5)
                continue;
                        System.out.println(i);
```

```
                }
        }
}
```

***Flow control statement interview Questions***

1) *How many types flow control statements in java?*
2) *What is the purpose of conditional statements?*
3) *What is the purpose of looping statements?*
4) *What are the allowed arguments of switch?*
5) *When we will get compilation error like "possible loss of precision"?*
6) *Inside the switch case vs. default vs. Break is optional or mandatory?*
7) *Is it possible to use variables as case labels or not?*
8) *While declaring if , if-else , switch curly braces are optional or mandatory?*
9) *Switch is allowed String argument or not?*
10) *Switch allows float,double,long arguments or not?*
11) *Inside switch how many cases possible & how many default declarations are possible?*
12) *What are the advantages of fall through inside the switch?*
13) *What is difference between if & if-else & switch?*
14) *What is the default condition of for loop & who will provide the default condition?*
15) *What is the difference between for loop & for-each loop?*
16) *When we will get compilation error like "incompatible types"?*
17) *for (; ;) representing?*
18) *When we will get compilation error like "unreachable statement "?*
19) *What is the difference between for loop & while loop?*
20) *Is it possible to declare while without condition?*
21) *What is the difference between while and do-while?*
22) *What do you mean by transfer statements and what are transfer statements in java?*
23) *We are able to use break statements how many places and what are the places?*
24) *What is the difference between break& continue?*
25) *Is it possible to use break inside the if statement?*

********** ***Thank you*** **********

## ***Java class concept***

*Class Test*
*{ 1) variables*
*2) methods*
*3) constructors*
*4) instance blocks*
*5) static blocks*
*}*

### ***Java Variables:***

- ✓ *Variables are used to store the constant values by using these values we are achieving project requirements.*
- ✓ *Generally project variables are :*

*Employee project : int eid; String ename; double esal;*
*Student project : int sid; String sname; double smarks;* ✓ *Variables are also known as **fields** of a class or **properties** of a class.*

- ✓ *All variables must have a type. That type it may be*

○ *Primitive type ( int, float,…)*
○ ○ *Int a=10;*
○ *Array type*
*Int[ ] a;*
*class type*
*Test t;*
*Enum type*
*Week k;*
○ ○ *Interface type.*
○ *It1 I;*
*Variable argument type.*
*Int… a;*
*Annotation type*
*Override e;*

- ✓ *Variable declaration is composed of three components in order,* ○ *Zero or more modifiers.*

- *The variable type.*
- *The variable name.*
- *Constant value/ literal*

***public final int x=100;***

***public int a=10;***

***public*** ----> *modifier (specify permission)*
***int*** ----> *data type (represent type of the variable)*
***a*** ----> *variable name*
*10* ----> *constant value or literal;*
*;* ----> *statement terminator*

*There are three types of variables in java*

1. *Local variables.*
2. *Instance variables.*
3. *Static variables.*

## *Local variables:*

- *The variables which are declare inside a method or constructor or blocks those variables are called local variables.*

```
class Test
{       public static void main(String[] args)
        {       //local variables
                int a=10;
                int b=20;
                System.out.println(a+b);
        }
}
```

- *It is possible to access local variables only inside the method or constructor or blocks only, it is not possible to access outside of method or constructor or blocks.*

```
 void add()
 {      int a=10;
        System.out.println(a);    //possible
 }
 void mul()
{         System.out.println(a);
}       //not-possible
```

- *Local variables memory allocated when method starts & memory released when method completed.*

- *The local variables are stored in stack memory.*

***There are five types of memory areas in java:***

1. *Heap memory*
2. *Stack memory*
3. *Method area*
4. *SCP(String constant pool) memory*
5. *Native method stacks*

***Areas of java:*** *There are two types' areas in java.*

***Instance Area:-***

```
void m1()              //instance method
{   Logics here        //instance area
}
```

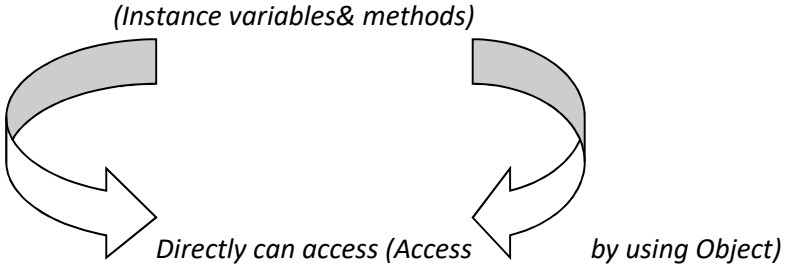
***Static Area:-***

```
Static void m1()       //static method
{    Logics here       //static area
}
```

***Instance variables (non-static variables):***

- ✓ *The variables which are declare inside a class but outside of methods are called instance variables.*
- ✓ *The scope (permission) of instance variable is inside the class having global visibility.*
- ✓ *Instance variables memory allocated during object creation& memory released when object is destroyed.*
- ✓ *Instance variables are stored in heap memory.*

### *Instance variable accessing:*



***Ex : File Name : Balu.java***

```
class Test
{       //instance variables
        int a=10;       int
b=20;
        //static method         public
static void main(String[] args)
        {       //Static Area
                Test t=new Test();
                System.out.println(t.a);
                System.out.println(t.b);
                t.m1();                         //instance method calling
        }
        //instance method
        void m1()       {
        //instance area
                System.out.println(a);
                System.out.println(b);
        }//main ends
};//class ends
```

- ✓ *In java program execution always starts from main method called by JVM.*
- ✓ *The JVM is responsible to execute only main method so must call the user defined method (m1)inside the main method then only user method will be executed.*

***Example :-operator overloading***

- ✓ *One operator with more than one behavior is called operator over loading.*
- ✓ *Java is not supporting operator overloading concept but only one implicit overloaded operator in java is **+** operator.*
  - o *If two operands are integers then **plus (+)** perform addition.*
  - o *If at least one operand is String then plus **(+)** perform concatenation.*

```
class Test
{       public static void main(String[] args)
        {       System.out.println(10+20);                      //addition
                System.out.println("balu"+"anushka"+2+2+"kids");     //concatenation
```

```
        int a=10;
        int b=20;
        int c=30;
        System.out.println(a);
        System.out.println(a+"---");
        System.out.println(a+"---"+b);
        System.out.println(a+"---"+b+"----");
        System.out.println(a+"---"+b+"----"+c);
    }
}
```

***Example: -Different ways to initialize the variables***

```
class Test
{       int p=10,q=20,r=30;             //10   20    30
        int i=10,j=20,k;                //10   20    0
        int x=10,y,z;                   //10   0     0
      static int d=10,e,f=30;           //10    0   30
        static int a,b,c;               //0    0    0
        public static void main(String[] args)
        {       Test t = new Test();
                System.out.println(t.p+"---"+t.q+"---"+t.r);
                System.out.println(t.i+"---"+t.j+"---"+t.k);
                System.out.println(t.x+"---"+t.y+"---"+t.z);
                System.out.printlnTest.d+"---"+Test.e+"---"+Test.f);
                System.out.println(Test.a+"---"+Test.b+"---"+Test.c);
        }
}
```

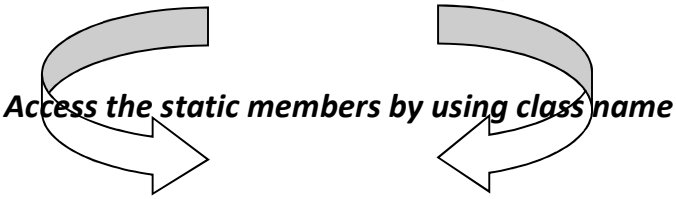
### *Static variables (class variables):*

- *The variables which are declared inside the class but outside of the methods with static modifier are called static variables.*
- *Scope of the static variables with in the class global visibility.*

- *Static variables memory allocated during .class file loading and memory released at .class file unloading time.*
- *Static variables are stored in method area.*

***Static variables & methods accessing:***

*(Static variables& static methods)*




***Static area*** ***instance area***

```
class Test
{       //static variables
        static int a=1000;
        static int b=2000;
        public static void main(String[] args)      //static  method
        {       System.out.println(Test.a);
                System.out.println(Test.b);
                Test t = new Test();
                t.m1();             //instance method calling
        }
        void m1()         //instance method
        {       System.out.println(Test.a);
                System.out.println(Test.b);
        }
}
```

***Static variables calling:*** *- There are three ways to access the static members*

*1) Direct accessing*

*2) By using reference variable.*

*3) By using class-name. (Project level use this approach)*

```
class Test
{       static int x=100;           //static variable
        public static void main(String[] args)

        {       System.out.println(a);              //1-way(directly possible)
                System.out.println(Test.a);         //2-way(By using class name)
                Test t=new Test();
```

```
        System.out.println(t.a);                    //3-way(By using reference variable)
}       }
```

***Example: -***

- ✓ *It is possible to create the objects inside the main method & inside the user defined methods also.*
- ✓ *When we create object inside method that object is destroyed when method completed, if any other method required object then create the object inside that method.*

```
class Test
{       int a=10;
        int b=20;
        static void m1()
        {       Test t = new Test();
                System.out.println(t.a);
                System.out.println(t.b);
        }
        static void m2()
        {       Test t = new Test();
                System.out.println(t.a);
                System.out.println(t.b);
        }
        public static void main(String[] args)
        {       Test.m1();      //static method calling
                Test.m2();      //static method calling
        }
}
```

***Variables VS default values:***

***Case 1:-****for the instance & static variables JVM will assign default values.*

```
        class Test
        {       int a;
                Static boolean b;
                public static void main(String[] args)
                {       Test t=new Test();
                        System.out.println(t.a);
                        System.out.println(Test.b);
                }
        }
```

***Case 2:-***

- ✓ *Local variables JVM does not provide default values.*

- ✓ *In java before using local variables must initialize some values to the variables otherwise compiler generates compilationerror "variable a might not have been initialized"***.**

```
class Test
{       public static void main(String[] args)
        {       int a;
                System.out.println(a);//error: variable a might not have been initialized
        }
};
```

***Class vs. Object:-***

- ✓ *Class is a logical entity it contains logics whereas object is physical entity it is representing memory.*
- ✓ *Class is blue print it decides object creation without class we are unable to create object.*
- ✓ *Based on single class it is possible to create multiple objects but every object occupies memory.*
- ✓ *We are declaring the class by using class keyword but we are creating object by using new keyword.*

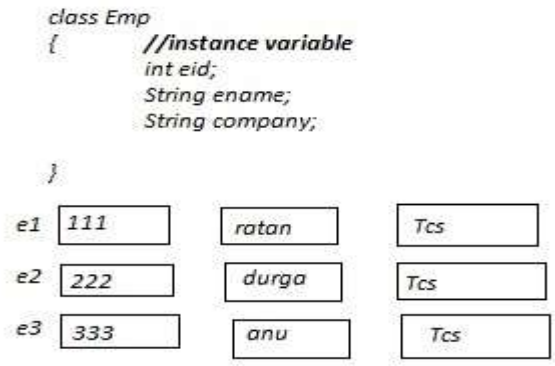
***Instance vs. Static variables:-***

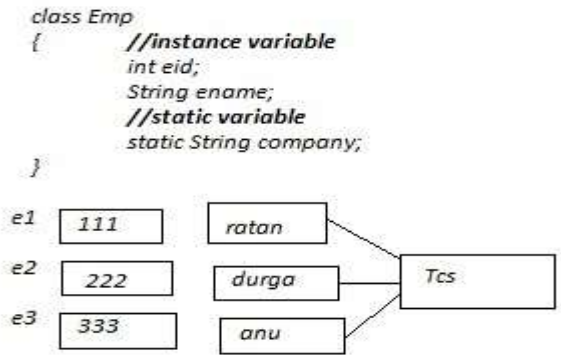
- ✓ *In case of instance variables the JVM will create separate copy of memory for each and every object.*
- ✓ *When we perform modifications on instance variables that will be reflected on same object only.*
- ✓ *In case of static variables irrespective of object creation per class single memory is allocated.*
- ✓ *When we performed modifications on static variable then will be reflected on next created all objects.*

```
class Test
{       int a=10;
        static int b=20;
        public static void main(String[] args)
        {       Test t = new Test();
                System.out.println(t.a);
                System.out.println(t.b);
                t.a=111;
                t.b=222;
                System.out.println(t.a);
                System.out.println(t.b);
                Test t1 = new Test();
                System.out.println(t1.a);
                System.out.println(t1.b);
        t1.b=444;
                Test t2 = new Test();
                System.out.println(t2.a);
```

```
            System.out.println(t2.b);
        }
}
```

***Instance variable vs. static variable :-***




***Java Methods*** ***{class-1}***

- ✓ *Inside the classes it is not possible to write the business logics directly, so inside the class declares the method inside that method writes the logics of the application.*
- ✓ *Methods are used to write the business **logics** of the application.*

- ✓ ***Coding convention**: method name starts with lower case letter and every inner word starts with uppercase letter (**mixed case**).*

  *ex: post() , char**A**t() , to**U**pper**C**ase() , compare**T**o**I**gnore**C**ase()……etc*

  *There are two types of methods in java,*

  1. *Instance method*
  2. *Static method*

- ❖ *Inside the class it is possible to declare **n** number of instance &static methods based on the developer requirement.*
- ❖ *It will improve the reusability of the code and we can optimize the code.*
- ❖ *Whether it is an instance method or static method the methods are used to provide business logics of the project.*

## *Instance method:*

```
void m1()       //instance method
{  //body      //instance area
}
```

*Note: Instance member's memory is allocated during object creation hence accesses the instance members by using object-name (reference-variable).*

*Objectname.instancemethod( );**//calling instance method***

*Test t = new Test(); t.m1( );*

## *Static method:*

*static void m1() **//static method***

*{ //body **//static area***

*}*

***Note:** Static member's memory allocated during **.class** file loading hence access the static members by using class-name.*

*Classname.staticmethod( ); **// call static method by using class name** Test.m2( );*

***Every method contains three parts.***

1. *Method declaration*
2. *Method implementation (logic)*
3. *Method calling*

*ex :*

```
void m1()                          ------>   method declaration
{       Body  (Business logic);    ----->    method implementation
}
Test t = new Test();
 t.m1();                           ------>   method calling
```

***Method Syntax:***

***[modifiers-list] return-Type Method-name (parameters list) throws Exception***

| | | |
|---|---|---|
| ***Method name*** | ---> | *functionality name* |
| ***Parameter-list*** | ---> | *input to functionality* |
| ***Modifiers-list*** | ---> | *represent access permissions*. |
| ***Return-type*** | ---> | *functionality return value* |
| ***Throws Exception*** | ---> | *representing exception handling* |

***ex:*** *Public void m1()*
*Private int m2(int a,int b)*
*Private String m2(char ch)throws Exception*

***Method Signature:*** *Method-name & parameters list is called method signature.*

*Syntax: Method-name(parameter-list)*
*ex: m1(int a)*
*m2(int a,int b)*

***ex 1 : instance & static methods without arguments.***

```
class Test
{       void m1()
        {       System.out.println("m1 instance method");
        }
        static void m2()
        {       System.out.println("m2 static method");
        }
        public static void main(String[] args)
        {       Test t = new Test();
                t.m1();
        Test.m2();
        }
}
```

***ex 2: Instance & static methods with parameters.***

- ✓ *If the method is expecting parameters (inputs to functionality), while calling that method must pass the values to parameters.*
- ✓ *If the method expecting three arguments must pass three arguments, not less than three not more than three.*
- ✓ *Method arguments are always local variables.*

```
class Test
{       void m1(int a,char ch)    //local variables
        {       System.out.println("m1 instance method");
                System.out.println(a);
                System.out.println(ch);
```

```
        }


        static void m2(String str,boolean b,double d)
        {       System.out.println("m2 static method");
                System.out.println(str);
    System.out.println(b);
                System.out.println(d);
        }
        void m3(int a, int b)
        {       System.out.println("m3 method");
    System.out.println(a);
                System.out.println(b);
        }
        public static void main(String[] args)
        {       Test t = new Test();
                t.m1(10,'a');
                 Test.m2("balu",true,10.5);

                int x=100;
                int y=200;
                t.m3(x,y);
        }
}
```

***Ex 3 : Instance & static methods with parameters.***

```
class Test
{       void m1(int a,char ch)
```

```
        }


        {       System.out.println("m1 method");
                System.out.println(a+"--"+ch);
        }
        static void m2(String str1,boolean b,double d)
        {       System.out.println("m2 method");
                System.out.println(str1+"   "+b+"   "+d);
        }
        void add(int a,int b)
        {       System.out.println(a+b);

        static void login(String uname,String upwd)
        {        if (uname.equals("balu"))
                {       System.out.println("Login Success");
                }
                else
                {       System.out.println("login fail");
                }
        }
        public static void main(String[] args)
        {       Test t = new Test();
        t.m1(10,'a');

                 Test.m2("balu",true,34.6);

                int x=100;
                int y=200;
                t.add(x,y);

                 Test.login("balu","durga");
        }
}
```

```
    }
```

***Ex 7 : java methods it is possible to provide Objects as a parameters(in real time project).***

```
class Emp{ }
class Student{ }
class Dog{ }
class Animal{ }
class Test
{       void m1(Emp e,Student s)
        {       System.out.println("m1 method");
                System.out.println(e+"  "+s);
        }
        static void m2(int a,Animal a1,Dog d)
        {       System.out.println("m2 method");
                System.out.println(a+" "+a1+" "+d);

        public static void main(String[] args)
        {       Test t = new Test();
                Emp e = new Emp();
                Student s = new Student();
                t.m1(e,s);

                Animal x = new Animal();
                Dog d = new Dog();
                Test.m2(10,x,d);
        }
}
```

- ✓ *In above example when we print reference variables hash code is printed (we will discuss later)*
- ✓ *The extra classes (Dog,Animal…) these classes it may belongs to same module classes or diffrernet module classes.*

***Ex: assignment***

```
class Test
```

```
    }

{       2-ins var
        2-static var
        1-ins method(Animal a,Dog d,Puupy p)
        1-static method(any 2-args)
        1-ins method(boolean b,Product p,Emp e)
        1-static method(float f,char ch,Bank b)

        public static void main(String[] args)
        {   print 4-var
            call 4 methods by passing arguments
        }
}
```

***ex 4:*** *Java methods return type is mandatory, otherwise the compilation will generate error message* ***"invalid method declaration; return type required ".***

```
class Test
{   m1()
    {
    }
}
```

***error: invalid method declaration; return type required***

***ex 5:*** *Inside the class it is not possible to declare more than one method with same signature* ***(duplicate methods)*** *, if we are trying to declare with same signature compiler generates error :* ***"m1() is already defined in Test"***

```
class Test
{       void m1()
        {       System.out.println("m1 method");
        }
        void m1()
        {       System.out.println("m1  method");
        }
}
```

***error: m1() is already defined in Test***

***ex 6:***

- ✓ *Declaring the class inside another class is called inner classes, java supports inner classes.*
- ✓ *Declaring the methods inside other methods is called inner methods but java not supporting inner methods concept if we are trying to declare inner methods compiler generate error message****.***

```
class Test
{       void m1()
        {       void m2()    //inner method : error: illegal start of expression
                {
                }
        }
}
```

***error: illegal start of expression***

***ex -8 :***

*If the application continas both instnace & local variables with same name, in this case to represent instance variables we have two approaches. 1. Access by uisng this keyword 2. Access by using object.*

***'This' keyword is used to represent current class object.***

```
class Test
{       int a=100,b=200;
        void add(int a,int b)
        {       System.out.println(a+b);
                System.out.println(this.a+this.b);          //approach-1

                Test t = new Test();
                System.out.println(t.a+t.b);                //approach-2
        }
        public static void main(String[] args)
        {       Test t = new Test();
        t.add(10,20);
        }
}
```

***Case 1: Invalid : inside the static area this keyword not allowed***

```
Int a=100,b=200; static
void add(int a,int b)
{   System.out.println(this.a+this.b);
}
```

***Compilation error:-****non-static variable* ***this*** *cannot be referenced from a static context.*

***Case 2:*** *In almost all cases we are using this keyword to represent instacne variables but inside the static area this keyword is not allowed hence use object creation approach.*

```
Int a=100,b=200; static
void add(int a,int b)
{   Test t = new Test();
    System.out.println(t.a+t.b);
}
```

***Example 9 :- methods vs return type.***

- ✓ *Java methods return type is mandatory & void represents no return value.*

- ✓ *Methods can have any return type like primitive type such as byte,short,int,long,float &Arrays type , Class type , Interface type ,Enum type….etc*

- ✓ *If the method is having return type other than void then must return the value by using **return** keyword otherwise compiler will generate error message "**missing return statement**"*

***Invalid***

```
int m1()
{   System.out.println("Anushka");
}
```

***Valid*** *int m1()*

```
{   System.out.println("Anushka");
    return 100;
}
```

- ✓ *Inside the method it is possible to declare only one return statement, that statement must be last statement of the method otherwise compiler will generate error message.*

- ✓ *Method is able to returns the value, it is recommended to hold the return value check the status of the method but it is optional.*

```
class Test
{       int m1()
        {       System.out.println("m1 method");
                return 10;
        }
        static boolean m2()
        {       System.out.println("m2 method");
        return false;
        }
        String m3()
        {       System.out.println("m3 method");
                 return "balu";
        }
        public static void main(String[] args)
        {       Test t = new Test();
                int x = t.m1();
```

```
                System.out.println("m1 method return value="+x);

                boolean b = Test.m2();
                System.out.println("m2 method return value="+b);

                String str = t.m3();
                System.out.println("m3 method return value="+str);
 }      }
```

***ex:***

```
class Test
{       int m1()
        {       System.out.println("m1 method");
        return 10;
        }
        static String m2()
        {       System.out.println("m2 method");
                return "balu";
        }
        int add(int a,int b)
        {       return a+b;
        }
        static String login(String uname,String upwd)
        {       if (uname.equals("balu"))
                {       return "success";
                }
                else
                {       return "fail";
                }
        }
        public static void main(String[] args)
        {       Test t = new Test();
                int x = t.m1();
                System.out.println("return value="+x);

                String str = Test.m2();
                System.out.println("return value="+str);
```

```
                int addition = t.add(10,20);
                System.out.println("return value="+addition);

                 String status = Test.login("balu","anu");
                System.out.println("Logins status="+status);
        }
}
```

***ex 10:*** *It is possible to print return value of the method in two ways,*

1. *Hold the return value & print that value.*
2. *Directly print the value by calling method using System.out.println() class Test*

```
{       int m1()
        {       System.out.println("m1 method");
                return 10;
        }
        public static void main(String[] args)
        {       Test t =new Test();
                int x = t.m1();
                System.out.println("return value="+x);            //1-way  printing return value

                System.out.println("return value="+t.m1());      //2-way printing return value
        }
}
```

***Observation :*** *If the method is having return type is void but if we are trying to call method by using System.out.println() then compiler will generate error message. Static void m2()*

```
        {       System.out.println("m2 method");
        }
        System.out.println(Test.m2());
```

***Compilation error:'void' type not allowed here***

***Example 11:- method return type is object***

```
class Student{  }  class Emp{ }
class Test
{       Emp m1()
        {       System.out.println("m1 method");
                Emp e = new Emp();
                return e;
        }
        Student m2()
        {       System.out.println("m2 method");
        Student s = new Student();
                return s;
        }
        public static void main(String[] args)
        {       Test t = new Test();
                Emp  e = t.m1();
                System.out.println("return value="+e);

                Student s = t.m2();
                System.out.println("return value="+s);
        }
}
```

***Example 12 :-****Java method is able to return current class object in two ways.*

1) *Creating object & return reference variable.*
2) *Return* ***this*** *keyword.*

*In almost all cases we are using this keyword but inside the static area this keyword not allowed hence use object creation approach.*

```
class Test
{       Test m1()
        {       System.out.println("m2 method");
                return this;
        }
        static Test m2()
        {       System.out.println("m1 method");
                Test t = new Test();
        return t;
        }
        public static void main(String[] args)
```

```
        {       Test t = new Test();
                Test t1 = t.m1();
                System.out.println("return value="+t1);
                Test t2 = Test.m2();
                System.out.println("return value="+t2);
        }
}
```

***Example13**: method arguments & return value class*

```
Animal {
 }
class Dog{
 }
class Emp{
}
class Test
{       Emp m1(Animal a,Dog d,int i)
        {       System.out.println("m1 method");
                System.out.println(a+"  "+d+"  "+i);
                Emp e = new Emp();
                return e;
        }
        public static void main(String[] args)
        {       Test t = new Test();
                Animal a = new Animal();
                Dog d  = new Dog();
                Emp e = t.m1(a,d,10);
                System.out.println("Return value ="+e);
        }
}
```

***Example 14: method vs return variables***

***Case 1:** if the application contains both instance & local variables the return value will be local.*

```
class Test
{       int a=10;
        int m1(int a)
        {    System.out.println("m1() method");
             return a;  //return local variable
        }
        public static void main(String[] args)
```

```
        {       Test t = new Test();
                int x = t.m1(100);
                System.out.println("return value="+x);
        }
}
```

***D:\>java Test m1()***
***method***
***100***

***Case 2:*** *No local variables in application the return value will be instance value.*

```
class Test
{       int a=10;
        int m1()
        { System.out.println("m1() method");
        return a; //returns instance value
        }
        public static void main(String[] args)
        {       Test t = new Test();
                int x = t.m1();
                System.out.println("return value="+x);
        }
}
```

***Case 3:****If the application contains both local & instance variables with same name then first priority goes to local variables but to return instance value use* ***this*** *keyword.*

```
class Test
{       int a=10;
        int m1(int a)
        {       System.out.println("m1() method");             return this.a;
//return instance variable as a return value.
        }
        public static void main(String[] args)
        {       int x = new Test().m1(100);
                System.out.println("m1() return value is="+x);
        }
}
```

***Example 15 :-***

✓ *Scanner class present in **java.util** package and it is introduced in 1.5 versions & it is used to take dynamic input from the keyboard.*

*to get int value ----> s.nextInt()*
*to get float value ----> s.nextFloat()*
*to get String value ----> s.next()*
*to get single line ----> s.nextLine()*
*to close the input stream ----> s.close()*

```
import java.util.*; class
Test
{       public static void main(String[] args)
        {       Scanner s=new Scanner(System.in);
                System.out.println("enter emp no");
                int eno=s.nextInt();

                System.out.println("enter emp name");
                String ename=s.next();

                System.out.println("enter emp salary");
                float esal=s.nextFloat();

                System.out.println("enter emp hobbies");
                String ehobbies = s.nextLine();

                System.out.println("*****emp details*****");
        System.out.println("emp no----->"+eno);
                System.out.println("emp name---->"+ename);
                System.out.println("emp sal------>"+esal);
                System.out.println("emp hobbies------>"+ehobbies);
        s.close();      //used to close the stream
        }
}
```

***Example 16:-*** *The \s represents whitespace.*

```
import java.util.*;  public
class Test
{       public static void main(String args[])
        {       String input = "7 tea 12 coffee";
                Scanner s = new Scanner(input).useDelimiter("\\s");
```

```
                System.out.println(s.nextInt());
                System.out.println(s.next());
                System.out.println(s.nextInt());
                System.out.println(s.next());
        s.close();
         }
}
```

***{class-3}***

***Example-17 :-Conversion of local variables to instace variables to improve the scope of the variable.***

```
class Test
{       //instance variables
        int val1;       int
val2;
        void values(int val1,int val2)      //local variables
        {       System.out.println(val1);
                System.out.println(val2);
                //conversion of local to instance (passing local variables values to instance variables)
        this.val1=val1;
                this.val2=val2;
        }
        void add()
        {       System.out.println(val1+val2);
        }
        void mul()
        {       System.out.println(val1*val2);
        }
        public static void main(String[] args)
        {       Test t = new Test();
        t.values(10,20);
                t.add();
                t.mul();
        }
}
```

*Observation :- if the instance variable names and static variable names are different we can assign directly without using this keyword.*

```
        //instance variables
```

```
int a;
int b;
void values(int val1,int val2)//local variables
{       System.out.println(val1);
        System.out.println(val2);
//conversion of local to instance (passing local variables values to instance variables)
        a=val1;
b=val2;
}
```

***Example 18:- conversion of local values to instance***

***Case 1: Inside the method taking local value then converting local to instance***

```
import java.util.*; class Test
{       int sid;  void
details()
        {       Scanner s = new Scanner(System.in);
        System.out.println("enter student id");
        int sid = s.nextInt();
                this.sid=sid;
        }
        void disp()
        {       System.out.println("student is="+sid);
        }
        public static void main(String[] args)
        {       Test t = new Test();
        t.details();
                t.disp();
        }
}
```

***Case 2:- Directly assigning values to instance variables.*** *int sid;*

```
void details()
{       Scanner s = new Scanner(System.in);
System.out.println("enter student id");
        sid = s.nextInt();
}
void disp()
{       System.out.println("student is="+sid);
}
```

***Example 19:- Method recursion****A method is calling itself during execution is called recursion.*

***case 1:- (normal output)***

```
class RecursiveMethod
{       static void recursive(int a)
        {System.out.println("number is :- "+a);
                if (a==0)
        {return;}
                recursive(--a);
        }
        public static void main(String[] args)
        {       RecursiveMethod.recursive(10);
        }
};
```

***case 2:- (StackOverFlowError)*** *class RecursiveMethod*

```
{       static void recursive(int a)
        {System.out.println("number is :- "+a);
                if (a==0)
        {return;}
                recursive(++a);
        }
        public static void main(String[] args)
        {       RecursiveMethod.recursive(10);
        }
};
```

***Example-20:- methods vs. All data- types***

- ✓ *In java numeric values are by default integer values but to represent byte, short perform typecasting.*
- ✓ *In java decimal values are by default double values but to represent float value use "**f**"constant or perform typecasting.*

```
class Test
{       void m1(byte a)         {       System.out.println("Byte value-->"+a);   }       void
m2(short b )    {       System.out.println("short value-->"+b);  }       void m3(int c)
        {       System.out.println("int value-->"+c);           }       void m4(long d)
        {       System.out.println("long value is-->"+d);       }       void m5(float e)
        {       System.out.println("float value is-->"+e);      }       void m6(double f)
        {       System.out.println("double value is-->"+f);     }       void m7(char g)
```

```
        {       System.out.println("character value is-->"+g);    }       void m8(boolean h)
        {       System.out.println("Boolean value is-->"+h);      }       public static void
main(String[] args)
        {       Test t=new Test();
                t.m1((byte)10);
                t.m2((short)20);
                t.m3(30);
                t.m4(40);
                t.m5(10.6f);
                t.m6(20.5);
                t.m7('a');
                t.m8(true);
        }
}
```

***Example 21:-java method calling: one method is able to call more than one method*** *class Test*

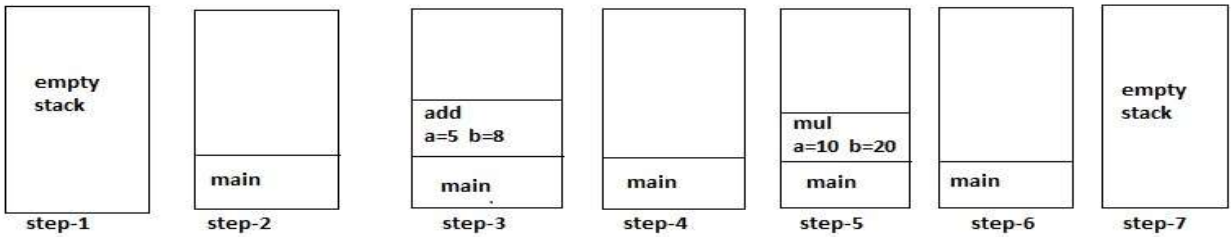
```
{       void m1()
        {       m2(100);
                System.out.println("m2 ");
                m2(200);
        }
        void m2(int a)
        {       System.out.println("m3 ");
        }
        public static void main(String[] args)
        {       Test t=new Test();
                t.m1();
        }
}
```

***Example 22 :- Stack Memory***

- ✓ *The jvm will create empty stack memory just before calling main method & jvm will destroyed empty stack memory after completion of main method.*

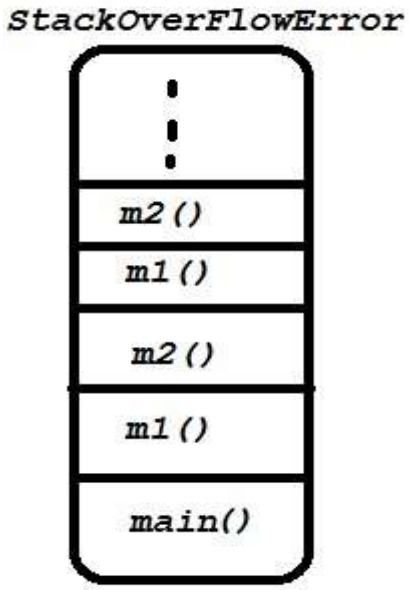
- ✓ *When JVM calls particular method then that method entry and local variables stored in stack memory&when the method completed, that particular method entry and local variables are destroyed from stack memory &that memorybecomes available to other methods.*

```
class Test
{       void add(int a,int b)
        {       System.out.println(a+b);
        }
        void mul(int a,int b)
        {       System.out.println(a+b);
        }
        public static void main(String[] args)
        {       Test t = new Test();
                t.add(5,8);
                t.mul(10,20);
        }
};
```




***Example 23:-when we call methods recursively then JVM will generate StackOverFlowError.***

```
class Test
{       void m1()
        {       System.out.println("rattaiah");
                m2();
        }
        void m2()
        {       System.out.println("Sravya");
                m1();
        }
        public static void main(String[] args)
        {       Test t=new Test();
                t.m1();
        }
}
```




***CONSTRUCTORS*** ***{class-1}***

***Class vs. Object:-***

- ✓ *Class is a logical entity it contains logics whereas object is physical entity it is representing memory.*
- ✓ *Class is blue print it decides object creation without class we are unable to create object.*
- ✓ *Based on single class it is possible to create more than one object but every object occupies memory.*
- ✓ *Declare the class by using class keyword & create the object by using new keyword.*

***Object creation syntax:-***

***Class-name reference-variable = new class-name ();*** *Testt =new Test();*

***Test*** ---> *class Name*
***t*** ---> *Reference variables*
***=*** ---> *assignment operator*
***new*** ---> *keyword used to create object*
***Test ()*** ---> *constructor*
***;*** ---> *statement terminator*

*When we create instance (Object) of a class using new keyword, a constructor for that class is called.*

***Different ways to createobject:-***

- ✓ *By using new operator*
- ✓ *By using clone() method ✓ By using new Instance() method ✓ By using instance factory method.*
- ✓ *By using static factory method*
- ✓ *By using pattern factory method*
- ✓ *By using deserialization process*

***New keyword:-***

- ✓ *New keyword is used to create object in java.*
- ✓ *When we create object by using new operator after new keyword that part is constructor then constructor execution will be done.*

***Rules to declare constructor:-***

1) *Constructor name class name must be same.*
2) *It is possible to provide parameters to constructors (just like methods). 3) Constructor not allowed explicit return typeeven **void**.*

***0-arg constructor :-***

```
Test()
{       //logics here
}
```

***1-arg constructor :-***

```
Test(int a)
{       //logics here
}
```

*There are two types of constructors,*

1) *Default Constructor (provided by compiler).*
2) *User defined Constructor (provided by user) or parameterized constructor.*

***Default Constructor:***

- ✓ *Inside the class if we are not declaring any constructor then compiler generates zero argument constructors with empty implementation at the time of compilation is called default constructor.*
- ✓ *The compiler generated constructor is called **default constructor.***
- ✓ *Inside the class default constructor is invisible mode.*
- ✓ *To check the default constructor provided by compiler open the .class file code by using java decompiler software.*

***Application before compilation:-*** *class*

```
Test
{       void m1()
        {       System.out.println("m1 method");
```

```
    }
    public static void main(String[] args)
    {       Test t = new Test();
            t.m1();
    }
}
```

*In above application when we create object by using new keyword "**Test t = new Test()**" then compiler is searching for **"Test()"** constructor inside the class since not available hence compiler generate default constructor at the time of compilation.*

***Application after compilation :-*** *class*

```
Test
{       void m1()
        {       System.out.println("m1 method");
        }
        /* default constructor generated by compiler
        Test()
        {empty implementation
        }
        */      public static void
main(String[] args)     {       Test t = new
Test();
                t.m1();
        }
}
```

- ✓ *Only the compiler generated 0-argument constructor is called default constructor.*
- ✓ *The user defined 0-argument constructor is not a default constructor.*

***User defined constructor:-***

- ✓ *Constructors are used to write the logics these logics are executed during object creation.*
- ✓ *Method also used to write the logics these logics are executed when we call the method.*

```
class Test
{       void m1()
```

```
        {       System.out.println("m1 method");
        }
        Test()
        {       System.out.println("0-arg constructor");
        }
        Test(int i)
        {       System.out.println("1-arg constructor");
        }
        public static void main(String[] args)
        {       Test t1=new Test();
        Test t2=new Test(10);
                t1.m1();
        t2.m1();
        }
}
```

- ✓ *The compiler generated 0-argument is called default constructor.*
- ✓ *The user defined 0-argument constructor is not a default constructor.*
- ✓ *The constructor arguments are local variables.*

***Example:-***

- ✓ *Inside the class if we are declaring at least one constructor (either 0-arg or parameterized) then compiler generate 0-arg constructor with empty implementation is called default constructor.*
- ✓ *Inside the class if we are declaring at least one constructor (either 0-arg or parameterized) then compiler not generating default constructor.*
- ✓ *If we are trying to compile below application the compiler will generate error message "Cannot find symbol " because compiler is unable to generate default constructor.*

```
class Test
{       Test(int i)
        {       System.out.println("1-arg constructor");
        }
        public static void main(String[] args)
        {       Test t1=new Test();     //error : inside the class no 0-arg constructor
                Test t2=new Test(10);
        }
}
```

***Object creation formats:-*** *2-formats of object creation.*

1) *Named object (having reference variable)* ***Test t = new Test();***
2) *Nameless object (without reference variable)* ***new Test();***

```
class Test
{       void m1()
        {       System.out.println("m1 method");
        }
        public static void main(String[] args)
        {       //named object
                Test t = new Test();
                t.m1();
                //nameless object
                new Test().m1();
        }
}
```

***ex :***

*void m1(){} void m2(){}*

***//named object : good*** *Test t = new Test(); new Test().m1(); t.m1();*

***//name less object :*** *new Test().m2(); t.m2();*

***ex: method arguments are nameless object***

***//named object*** *void m1(X x,Emp e){ } X x = new X(); Emp e = new Emp(); t.m1(x,e);*

***//nameless object : good*** *void m1(X x,Emp e){ } t.m1(new X(),new Emp());*

***ex: method return value is name less object.***

***//named object***

*Emp m1()*

```
{       Emp e = new Emp();
        return e;
}
```
***//name-less object  : good***
```
Emp m1()
{       return new Emp();
}
```
***Application with named object :-***
```
class X{} class Y{} class Z{}    class Emp{} class Animal{} class Dog{} class Puppy{} class Test
{       Test m1(X x,Y y,Z z)
        {       System.out.println("m1 method");
                return this;
        }
        static Emp m2(Animal a,Dog d,Puppy p)
 { System.out.println("m2 method");  Emp e = new
Emp();  return e;
        }
        public static void main(String[] args)
        {       Test t = new Test();
X x = new X();
Y y = new Y();
Z z = new Z();
                Test t1 = t.m1(x,y,z);
                System.out.println("return value="+t1);

                Animal a = new Animal();
                Dog d = new Dog();
                Puppy p = new Puppy();
                Emp e = Test.m2(a,d,p);
                System.out.println("return value="+e);
        }
}
```
***Application with nameless object :-***
```
class X{} class Y{} class Z{}    class Emp{} class Animal{} class Dog{} class Puppy{} class Test
{       Test m1(X x,Y y,Z z)
        {       System.out.println("m1 method");
                return this;
        }
```

```
        static Emp m2(Animal a,Dog d,Puppy p)
        {       System.out.println("m2 method");
                return new Emp();
        }
        public static void main(String[] args)
        {       Test t1 = new Test().m1(new X(),new Y(),new Z());
                System.out.println("return value="+t1);
                Emp e = Test.m2(new Animal(),new Dog(),new Puppy());
                System.out.println("return value="+e);
        }
}
```

***2- formats of object creation.*** ***{Class-2}***

1) Eager object creation.
2) Lazy object creation.

```
class Test
{       void m1()
        {       System.out.println("m1 method");
        }
        public static void main(String[] args)
        {       //Eager object creation approach
        Test t = new Test();
                t.m1();

                //lazy object creation  approach
                Test t1;
                ;;;;;;;;;;
                t1=new Test();
                t1.m1();
        }
}
```

ex: ***problem*** *: in below example in every method we are creating obj*

```
class A { void disp() {
}
}
class Test
{       void m1()
        {       A a =
new A();
        a.disp();
        }
        void m2()
        {       A a =
new A();
        a.disp();
        }
        void m3()
        {       A a =
new A();
        a.disp();
        }
}
```

*ex: **Solution** : only one time we are creating object. class Test*

```
{       A a;            //ins var
        void m1()       {       a = new A();
                        a.disp();
        }
        void m1()       {
        a.disp();
        }
        void m1()       {
        a.disp();
        }
}
```

***Constructor calling: - To*** *call Current class constructor use this keyword*

***this();*** *----> current class 0-arg constructor calling*
***this(10);*** *----> current class 1-arg constructor calling*
***this(10 , true);*** *----> current class 2-arg constructor calling*

***ex-1:-****Call the methods by using method name but call t contructor using this keyword.*

```
class Test
{       Test()
        {       this(100);      //current class 1-arg constructor calling
                System.out.println("0-arg constructor  logics");
        }
        Test(int a)
        {       this('g',10);       //current class 2-arg constructor calling
                System.out.println("1-arg constructor logics");
        }
        Test(char ch,int a)
        {       System.out.println("2-arg constructor logics");
        }
        public static void main(String[] args)
        {       new Test();
        }
}
```

***ex-2:-*** *Inside the constructor this keyword must be first statement.*

```
Test()
{       System.out.println("0 arg");
        this(10);
}
```

*Error : "**call to this must be first statement in constructor**".*

***ex -3:-*** *one method is able to call more than one method at a time but one constructor is able to call only one construtor at time.*

```
Test()  {
this(10);
```

```
        this(10,20);
        System.out.println("0-arg cons");
    }
    error: call to this must be first statement in constructor
```

***Advantages of constructors:-***

- ✓ *Constructors are used to write the logics these logics are executed during object creation.*
- ✓ *Constructors are used to initialize the values to instance variables during object creation.*
- ✓ *Constructor is a special method used to initialize the object.*

***Example :-***

```
class
Employee
{       //instance
variables       int eid;
        String ename;
        double esal;
        void display()
        {       System.out.println("****Employee details****");
        System.out.println("Emp eid :-->"+eid);
                System.out.println("Emp name :-->"+ename);
                System.out.println("Empesal :-->"+esal);
        }
        public static void main(String[]
args)    {       Employee e1 = new
Employee();
                e1.display();
        }
}
```

```
D:\morn11>java Employee
****Employee details****
Emp eid :-->0
Empname :-->null
```

*Emp esal :-->0.0*

***Problems in above example***

- ✓ *In above example Emp object is created but default values are printing.*
- ✓ *To overcome abovelimitation to initialize the values to instance variables during object creation use constructor.*

***Example 2:-****Constructor used to initialize the values to instance variables during object creation.*

```
class Employee
{       //instance
variables       int eid;
        String ename;
        double esal;
        //constructor used to initialize the values to instance variables during object creation.
        Employee()
        {       eid=111;
        ename="balu";
        esal =60000;
        }
        void display()
        {       System.out.println("****Employee details****");
                System.out.println("Empid :-->"+eid);
                System.out.println("Empname :-->"+ename);
                System.out.println("Empsal :-->"+esal);
        }
        public static void main(String[]
args)   {       Employee e1 = new
Employee();
                e1.display();
        }
}
```

```
D:\morn11>java Employee
****Employee details****
Empid :-->111
Emp name :-->balu
Empsal :-->60000.0
```

✓ *In above example when we are create only one object it is good process.*

***Problem :-***

```
public static void main(String[] args)
        {       Emp e1 = new Emp();
        e1.display();
        Emp e2 = new Emp();
        e2.display();
}
D:\morn11>java Employee
 Empname :-->balu                                   Emp name :-->balu
Emp  id :-->111                                     Emp id :-->111
Emp sal :-->60000.0                                 Empsal :-->60000.0
```

✓ *In above example when we create more than one object for every object same constructor executed it initializes the same values so to overcome this problem use parameterized constructor to initialize different value to different objects.*

***Example :-*** *To initialize different values to different objects use parameterized constructor.*

```
class Employee
{       //instance
variables       int eid;
        String ename;
        double esal;
        Employee(int eid,String ename,double esal)      //local variables
        {       //conversion (passing local values to instance
values) this.eid = eid;                 this.ename = ename;
                this.esal = esal;
        }
        void display()
        {       System.out.println("****Employee details****");
        System.out.println("Empid :-->"+eid);
                System.out.println("Emp  name :-->"+ename);
                System.out.println("Emp sal :-->"+esal);
        }
        public static void main(String[] args)
        {       Employee e1 = new Employee(111,"balu",60000);
                e1.display();
```

```
                new Employee(111,"balu",60000).disp();

                Employee e2 = new Employee(222,"anu",70000);
                e2.display();

                new Employee(222,"anu",70000).disp();
        }
}
```

***Object creation parts:-*** *Every object creation having three parts.*

1) ***Declaration:-***
   *Test t;*
   *Student s;*
   *Emp e;*
2) ***Instantiation:- (object creation)*** *new Test();*
   *new Student();*
   *new Emp();*
3) ***initialization:-(during object creation perform initialization)***
   *new Test(10,20);*
   *new Student("balu",111);*
   *new Emp(111,"balu",1000);*

***Example :- primitive variable vs reference variable***

- ***a*** *is variable of primitive type such as int,char,double,boolean…etc* - ***t****isreference variable &it is the memory address of object.*


```
class Test
{       public static void main(String[] args)
        {       int a=10;               //a is primitive variable
                System.out.println(a);

                Test t = new Test();    //t is reference variable
```

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

***Copy the values from one object to another object:-***

*There are three ways to copy the values from one object to another object.*

1) *Assigning values of one object to another object with out using constructor.*
2) *Copy the values of one object to another object by using constructor.*
3) *By using clone () method of Object class.*

***Example: Assigning** values of one object to another object with out using constructor.*

```
class Emp
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
        Emp(){ }
        void
disp()
        {       System.out.println("Emp id="+eid);
                System.out.println("Emp name="+ename);
        }
        public static void main(String[] args)
        {       Emp e1 = new
Emp(111,"balu");                Emp e2 =
new Emp();              e2.eid=e1.eid;
        e2.ename=e1.ename;
        e1.disp();              e2.disp();
        }
}
```

***Example 2:** Assigning values of one object to another object by using constructor.*

```
class Emp
{       int eid;
        String ename;
        Emp(int eid,String ename)
```

```
        {       this.eid=eid;
                this.ename=ename;
        }
        Emp(Emp e)
        {
        eid=e.eid;
                ename=e.ename;
        }
        void disp()
        {       System.out.println("Emp id="+eid);
                System.out.println("Emp name="+ename);
        }
        public static void main(String[] args)
        {       Emp e1 = new
Emp(111,"balu");                Emp e2 =
new Emp(e1);            e1.disp();
        e2.disp();
        }
}
```

***Difference between methods and constructors:-***

| ***Property*** | ***Methods*** | ***Constructors*** |
|---|---|---|
| ***1)Purpose*** | *methods are used to write logics used write but these logics will be executed project but the when we call that method. be executed during Object creation.* | *Constructor is logics of the logics will* |
| ***2)Variable initialization*** | *It is initializing variable when We call that method.* | *It is initializing variable during object creation.* |
| ***3)Return type*** | *Return type not allowed Even void.* | *It allows all valid return Types(void,int,Boolean…etc)* |
| ***4)Name*** | *Method name starts with lower Case & every inner word starts With upper case. Ex: charAt(),toUpperCase()….* | *Class name and constructor name must be matched.* |
| ***5)types*** | *a) instance method b)static method* | *a)default constructor b)user defined constructor* |
| ***6)inheritance*** | *methods are inherited* | *constructors are not inherited.* |
| ***7)how to call*** | *To call the methods use method Name.* | *to call the constructor use this keyword.* |
| ***8)Able to call how many Methods or constructors*** | *one method is able to call multiple methods at a time.* | *one constructors able to Call only one constructor at a time.* |
| ***9)this*** | *to call instance method use this Keyword but It is not possible to call static method.* | *To call constructor use this keyword but inside constructor use only one this statement.* |

***10)Super*** *used to call super class methods.* *Used to call super class constructor*

***11)Overloading*** *it is possible to overload methods* *it is possible to overload cons.*

***12)compiler generate Default cons or not*** *yes* *does not apply*

***13)compiler generate Super keyword.*** *yes* *does not apply.*

***Instance Blocks:-***

- *Instance blocks are used to write the logics of projects & these logics are executed during object creation just before constructor execution.*

  ***Syntax:***

  *{ **//logics here***
  *}*

- *Instance blocks execution depends on object creation it means if we are creating 10 objects 10 times instance blocks are executed.*
- *Inside the class it is possible to declare the more than one instance block the execution order is top to bottom.*

***Example:***

- ✓ ***During compilation the compiler will copy instance blocks code in every constructor.***
- ✓ ***The constructor logics are specific to object but instance block logics are common for all objects.***

***//Application before compilation (.java)***

```
class Test
{       {       System.out.println("instance block:logics-1");     }
        {       System.out.println("instance block:logics-2");     }
        Test()
        {       System.out.println("0-arg cons");
        }
        Test(int a)
        {       System.out.println("1-arg cons ");
        }
        public static void main(String[] args)
        {       new Test();
                new Test(10);
        }
}
```

***//Application after compilation (.class)***

```
class Test
{       Test()
        {       System.out.println("instance block:logics-1");
                System.out.println("instance block:logics-2");
                System.out.println("0-arg cons ");
        }
        Test(int a)
        {       System.out.println("instance block:logics-1");
                System.out.println("instance block:logics-2");
                System.out.println("1-arg cons ");
        }
        public static void main(String[] args)
        {       new Test();
                new Test(10);
        }
}
```

***Example:***

- ✓ *Instance block execution depends on object creation but not constructor exaction.*
- ✓ *In below example two constructors are executing but only one object is crating hence only one time instance block is executed.*
- ✓ *Inside the constructor when we declare this or super keyword in this instance block is not copied in constructor.*

```
class Test
{       {       System.out.println("instance block logics");
        }
        Test()
        {       this(10);
                System.out.println("0-arg constructor ");
        }
        Test(int a)
        {       System.out.println("1-arg constructor ");
        }
        public static void main(String[] args)
        {       new Test();
        }
```

```
}
```

***Example:-***

- ✓ *Instance blocks are used to initialize instance variables during object creation.*
- ✓ *In java it is possible to initialize the values in different ways ○ By using constructors ○ By using instance blocks ○ By setter methods*

```
class Emp
{       int eid;
        {       eid=111;}               //instance block

         Emp() {        eid=222;   }    //constructor

        void disp()
        {       System.out.println("emp id="+eid);
        }
        public static void main(String[] args)
        {       new Emp().disp();
        }
};
```

***Case 1:-*** *When we declare instance block & instance variable the execution order is top to bottom. In below example instance block is declared first so instance block is executed first.*

```
class Test
{       { System.out.println("instance block"); }       //instance block

        int a=m1();                                     //instance variables

        int m1()
        {       System.out.println("m1() method called by variable");
                return 100;
        }
        public static void main(String[] args)
```

```
        {       new Test();
        }
}
D:\morn11>java Test
instance block
m1() method called by variable
```

***case 2:-****When we declare instance block & instance variable the execution order is top to bottom.*
*In below example instance variable is declared first so instance block is executed first.*

```
class Test
{       int a=m1();             //instance variables
        int m1()
        {       System.out.println("m1() method called by variable");
                return 100;
        }
        { System.out.println("instance block"); }       //instance block
        public static void main(String[] args)
        {       new Test();
        }
}
D:\morn11>java Test
m1() method called by
variable instance block
```

***Static block:***

- ✓ *Static blocks are used to write the logics theselogics are executed during .class file loading time.*
- ✓ *In java .class file is loaded only one time hence static blocks are executed only once per class.*

```
static
        {       //logics here
        }
```

- ✓ *In class it is possible to write more than one static blocks but the execution order is top to bottom.*

***Note :*** *Instance blocks execution depends on object creation but static blocks execution depends on .class file loading.*

***Ex :***

```
class Test
{       static  {       System.out.println("static block-1");     }
        static  {       System.out.println("static block-2");     }
        public static void main(String[] args)
        {
        }
}
```

***Example :-***

```
class Test
{       static  {       System.out.println("static block-1");     }
        static  {       System.out.println("static block-2");     }
        {   System.out.println("instance block");          }
 Test() { System.out.println("0-arg cons"); }  Test(int a) {
System.out.println("1-arg cons"); }
        public static void main(String[] args)
        {       new Test();
                new Test(10);
        }
}
```

***Example :-*** *up to 1.6 version* ***Valid :*** *it is possible to execute static blocks without using main method.*

```
class Test
        {       static
                {   System.out.println("static block");
                }
        }
```

*From 1.7 version* ***Invalid :*** *it is not possible to execute static blocks without using main method.*

```
        class Test
        {       static
                {   System.out.println("static block");
                }
        }
```

***Example:-*** *The .class file loaded into memory in three different ways*

1. *When we execute the class by using java command the static blocks are executed but in this case main method is mandatory.*
2. *When we create the object the class loaded , the class static blocks are executed but in this case main method is mandatory.*
3. *When we load the file by using forName() method, the static blocks are executed but in this case main method is mandatory.* ***File 1:Demo.java*** *class Demo*

```
{       static
        {       System.out.println("Demo class static block");
        }
        void m1()
        {       System.out.println("Demo class m1 method");
        }
};
```

***File 2: Test.java***

```
class Test
{       public static void main(String[] args) throws Exception
        {       // new Demo().m1();
                Class c = Class.forName("Demo");
                Demo d = (Demo)c.newInstance();
                d.m1();
        }
}
```

***Example:-*** *Static blocks are used to initialize static variables.*

```
class Emp
{       static int eid;
        static  {
         eid=111;
        }
        static void disp()      {
        System.out.println(eid);
        }
        public static void main(String[] args)
        {       Emp.disp();
        }
}
```

## ***Java class concept interview questions***

1. *What is the purpose of the variables?*
2. *How many types of variables in java?*
3. *What are the memory areas in java?*
4. *What is the scope of local variables& instance & static?*
5. *Instance variables when memory allocated & when destroyed?*
6. *What are the areas in java?*
7. *What is the purpose of instance& static variables?*
8. *How to access the instance members & static members?*
9. *What do you mean by operator overloading & java supporting or not?*
10. *Is it possible to declare the instance variables inside the instance method or not?*
11. *Is it possible to declare the static variables inside the static method or not?*
12. *How many types of methods in java?*
13. *Define method signature?*
14. *For java methods return type mandatory or optional?*
15. *How to take the input from the keyboard?*
16. *What do you mean by method recursion & java supporting or not?*
17. *Java support inner methods or not?*
18. *When we will get the error message : StackOverFlowError*
19. *When we will get error message : missing return statement?*
20. *When we will get the error message : method already defined in class?*
21. *Who creates stack memory & who destroyed stack memory?*
22. *What are the rules to declare the constructor?*
23. *How many types of constructors in java?*
24. *What are the advantages of constructors?*
25. *What is the difference between method & constructor?*
26. *What do you mean by default constructor & who generate default constructor?*
27. *Inside the class I declared 1-arg constructor then default constructor generated or not?*
28. *What is the difference between named object & name less object?*
29. *How to call the current class constructor?*
30. *When we will get error message : this keyword must be first statement in constructor?*

31. *One constructor is able to call more than one constructor or not?*
32. *Define the words : declaration & instantiation & initialization?*
33. *User defined 0-arg constructor is default constructor or not?*
34. *What is the purpose of the instance block?*
35. *Executed 4-constructors by calling but I created only object how many times ins blocks executed?*
36. *What is the difference between instance block & constructor?*
37. *What is he purpose the static block?*
38. *What is the difference between instance block & static block?*
39. *How many ways are there to load the .class into memory?*
40. *To execute the static block inside the class main method mandatory or not?*
41. *Once I created 5 objects how many times instance blocks & static blocks executed?*

**************** ***Thank you*** ********************