Java means DURGA SOFT..

CORE JAVA

Material

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

DURGASOFT, # 202,2nd Floor,HUDA Maitrivanam,Ameerpet, Hyderabad - 500038, ☎ 040 – 64 51 27 86,
80 96 96 96 96, 9246212143 | www.durgasoft.com

Core Java

JAVA introduction:-

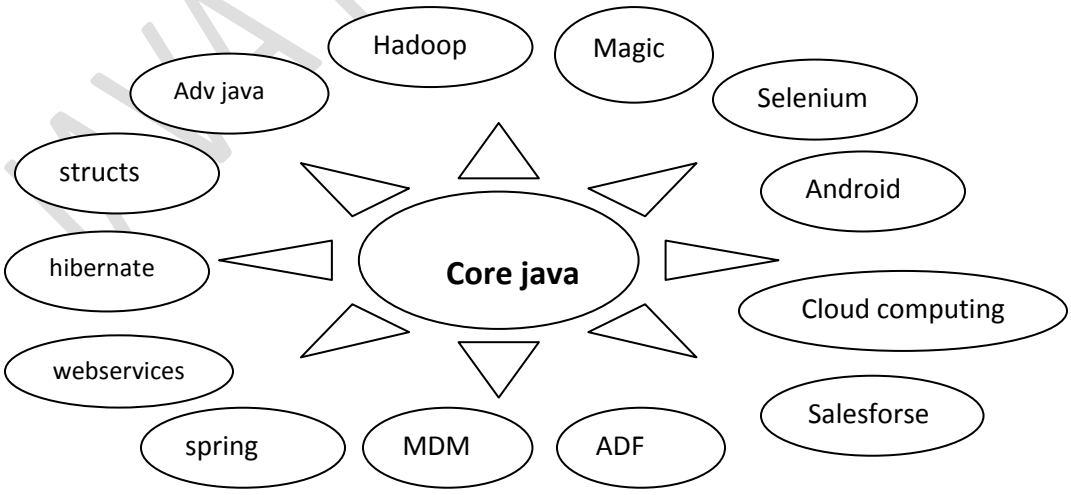
Author	:	***James Gosling***
Vendor	:	***Sun Micro System(****which has since merged into Oracle Corporation****)***
Project name	:	***Green Project***
Type	:	***open source & free software***
Initial Name	:	*OAK language*
Present Name	:	*java*
Extensions	:	*.java & .class & .jar*
Initial version	:	*jdk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

Importance of core java:-

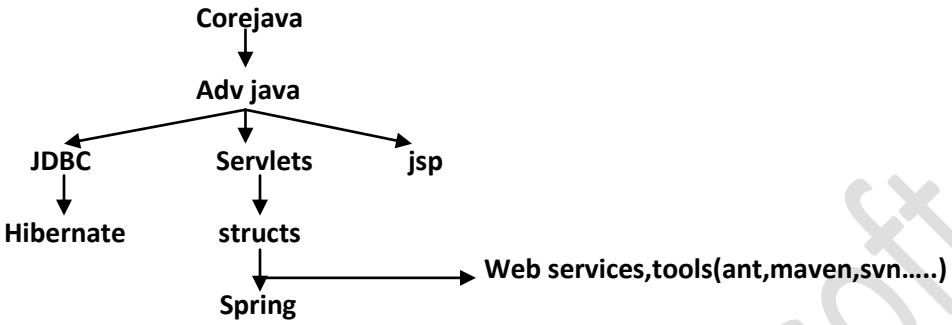
According to the SUN 3 billion devices run on the java language only.

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.***

Technologies Depends on Core java:-



Learning process:-



Parts of the java:-

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)

Java keywords:-

Data Types
byte
short
int
long
float
double
char
boolean
(8)

Flow-Control:-
if
else
switch
case
default
break
for
while
do
continue
(10)

method-level:-
void
return
(2)

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

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

Exception *handling:-*
try
catch
finally

throw
throws
*(**5**)*

1.5 version:-
enum
assert
(2)

unused:-
goto
const
(2)

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

JAVA VERSIONS:-

VERSION		***YEAR***
Java Alpha & beta	:	1995
JDK 1.0	:	1996
JDK1.1	:	1997
J2SE 1.2	:	1998
J2SE 1.3	:	2000
J2SE 1.4	:	2002
J2SE 1.5	:	2004
JAVA SE 6	:	2006
JAVA SE 7	:	2011
JAVA SE 8	:	2014

Differences:-

C-lang

1) Author: **Dennis Ritchie**

2) Implementation languages: BCPL, B…

3) In C lang program executionstarts frommain methodcalled by **Operating system.**

4) In c-lang the predefined support is available in the form of header files

Ex:- Stdio.h, Conio.h

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

6) To make available predefined support into our applications use #include statement.
Ex:- #include<stdio.h>
7) To print some statements into output console use "printf" function.
Printf("hi ratan ");

8)extensions used :-**.c ,.obj , .h**

Cpp-lang

1)Author : Bjarne Stroustrup

2) implementation languages are c ,ada,ALGOL68…..

3) program execution starts from main method called by **operating system.**

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

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

6) To make available predefined support into our application use #include statement.
Ex:- #include<iostream>

7) To print the statements use "cout" function.
Cout<<"hi ratan";

8)extensions used :-**.cpp ,.h**

Java -lang

1) Author : James Gosling

2) implementation languages are C,CPP,ObjectiveC…….

3) program execution starts from main method called by **JVM(java virtual machile**4)

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

5) The packages contains predefined classes and class contains predefined funtions.
Ex:- String,System

6) To make available predefined support into our application use import statement.
Ex:- import java.lang.*;
[*] mean all

7)To print the statements we have to use **System.out.println("hi ratan");**

8)extensions used : -**.java**, **.class**

C –sample application:-

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

Java sample application:-

```
        Import  java.lang.System;
        Import  java.lang.String;
        Class Test
```

CPP –sample application:-

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

```
{       Public static void main (String [] args)
        {       System.out.println ("welcome to java language");
        }
}
```

c-language:-

c-language → Dennis Ritchie → **void main()** (execution startsfrom main)

headerfiles → stdio.h,conio.h → **printf("ratan");** (used to print the output)

functions → printf,scanf…..

cpp-language:-

cpp-language → Bjarne Stroustrup → **void main()** (execution startsfrom main)

headerfiles → iostream.h → **cout<<"ratan";** (used to print the output)

functions → cout,cin….

java-language:-

java-language → james gosling → **public static void main(String[] args)** (execution startsfrom main)

packages → java.lang → **System.out.println("ratan");** (used to print the output)

classes & interfaces → System,String……..

methods & variables → length(),charAt(),concat()…

JAVA Features:-

1. Simple
2. Object Oriented
3. Platform Independent
4. Architectural Neutral
5. Portable
6. Robust
7. Secure
8. Dynamic
9. Distributed
10. Multithread
11. Interpretive
12. High Performance

1. Simple:-

Java is a simple programming language because:

- Java technology has eliminated all the difficult and confusion oriented concepts like pointers, multiple inheritance in the java language.
- The c,cpp syntaxes easy to understand and easy to write. Java maintains C and CPP syntax mainly hence java is simple language.
- Java tech takes less time to compile and execute the program.

2. ***Object Oriented:-***

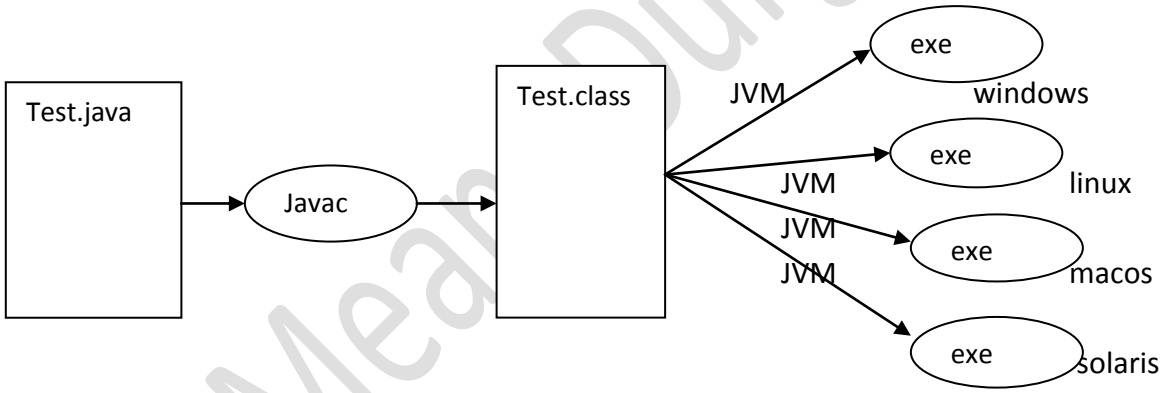
Java is object oriented technology because to represent total data in the form of object. By using object reference we are calling all the methods, variables which is present in that class.

Class Test

{logics } ***Test t=new Test();***

3. ***Platform Independent :-***

- Compile the Java program on one OS (operating system)that compiled file can execute in any OS(operating system) is called Platform Independent Nature.
- The java is platform independent language. The java applications allow its applications compilation one operating system that compiled (.class) files can be executed in any operating system.



source file compilation JVM understandable file output console

4. ***Architectural Neutral:-***

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

5. ***Portable:-***

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

6. ***Robust:-***

Any technology if it is good at two main areas it is said to be ROBUST

1. Exception Handling
2. Memory Allocation

JAVA is Robust because

- JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.
- JAVA is having very good memory management system that is Dynamic Memory (at runtime the memory is allocated) Allocation which allocates and deallocates memory for objects at runtime.

7. ***Secure:-***
   - To provide implicit security Java provide one component inside JVM called Security Manager.
   - To provide explicit security for the Java applications we are having very good predefined library in the form of java.Security.package.
8. ***Dynamic:-***

   Java is dynamic technology it follows dynamic memory allocation(at runtime the memory is allocated) and dynamic loading to perform the operations.
9. ***Distributed:-***

   By using JAVA technology we are preparing standalone applications and Distributed applications.

   Standalone applications are java applications it doesn't need client server architecture.
   web applications are java applications it need client server architecture.
   Distributed applications are the applications the project code is distributed in multiple number of jvm's.

10. ***Multithreaded: -***
    - Thread is a light weight process and a small task in large program.
    - If any tech allows executing single thread at a time such type of technologies is called single threaded technology.
    - If any technology allows creating and executing more than one thread called as multithreaded technology called JAVA.
11. ***Interpretive:-***

    JAVA tech is both Interpretive and Completive by using Interpretator we are converting source code into byte code and the interpretator is a part of JVM.
12. ***High Performance:-***

If any technology having features like Robust, Security, Platform Independent, Dynamic and so on then that technology is high performance.

Install the software and set the path :-

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

Download the software from internet based on your operating system. The software is different from 32-bit operating and 64-bit operating system.

To download the software open the fallowing web site.
http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

for 32-bit operating system please click on
Windows x86 :- 32- bit operating system

for 64-bit operating system please click on
Windows x64 :- 64-bit operating system

After installing the software the java folder is available in the fallowing location
Local Disk c: ------→program Files--------→java----→jdk(java development kit),jre(java runtime environment)

To check whether the java is installed in your system or not go to the command prompt. To open the command prompt
Start ----------→run------→open: cmd-----→ok

Command prompt is opened.

In the command prompt type :- javac
'javac' is not recognized is an internal or external command, operable program or batch file.

Whenever we are getting above information at that moment the java is installed but the java is not working properly.

C:/>javac

Whenever we are typing javac command on the command prompt

1) Operating system will pickup javac command search it in the internal operating system calls. The javac not available in the internal command list .
2) Then operating system goes to environmental variables and check is there any path is sets or not. up to now we are not setting any path. So operating system don't know anything about javac command Because of this reason we are getting error message.

Hence we have to environmental variables. The main aim of the setting environmental variable is to make available the fallowing commands javac,java,javap (softwares) to the operating system.

To set the environmental variable:-

My Computer (right click on that) ---->properties----->Advanced--->Environment Variables---->

User variables--→new---->variable name : Path
Variable value: C:\programfiles\java\jdk1.6.0_11\bin;.;
-------→ok------→ok

Now the java is working good in your system. open the command prompt to check once
C:>javac---------→now list of commands will be displayed

Steps to Design a First Application:-

Step-1:- Select Editor.
Step-2:- Write the application.
Step-3:- save the application.
Step-4:- Compilation Process.
Step-5:- Execution process.

Step1:- Select Editor

Editor is a tool or software it will provide very good environment to develop java application.

Ex :- Notepad, Notepad++,edit Plus…..etc

Note :- Do thepractical's of core java only by using*Edit Plus* software.

DE:- (Integrated development Environment)

IDE is providing very good environment to develop the application and it is real-timestandard but don't use IDE to develop core java applications.

Editor vs. IDE:-

If we are using IDE to develop core java application then 75% work is done by IDE like

1) Automatic compilation.
2) Automatic import.
3) It shows all the methods of classes.
4) Automatically generate try catch blocks and throws (Exception handling)
5) It is showing the information about how to fix the bug……etc

And remaining 25% work is down by developer

If we are using EditPlus software to develop application then 100% work done by user only.

Step 2:- Write a program.

- Write the java program based on the java API(Application Programming Interface) rule and regulations .

Open editplus --->file ----->new ------>click on java (it display sample java pplication)

- Java is a case Sensitive Language so while writing the program you must take care about the case (Alphabet symbols).

Example application:-

```
Import java.lang.System;
Import java.lang.String;
class Test          //class declaration
{           //class starts
        public static void main(String[] args)      //program starting point
        {           //main starts
                System.out.println("hi Ratan");  //printing statement
        }           //main ends
};          //class ends
class A
{       };
class B
{       };
```

*In above example **String & System c**lasses are present predefined java.lang package hence must import that package by using import statement.*

To import the classes into our application we are having two approaches,

1) Import all class of particular package.
    a. ***Import java.lang.*; //it is importing all classes of java.lang package.***
2) Import required classes

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

In above two approaches second approach is best approach because we are importing application required classes.

Step3:- save the application.

- After writing the application must save the application by using **(.java)** extension.
- While saving the application must fallow two rules
  - If the source file contains public class then public class and the name and Source file must be same ***(publicClassName.java)***. Otherwise compiler generate error message.
  - if the source file does not contain public class then save the source file with any name***(anyName.java)*** like A.java , Rtan.java, Anu.java …….etc.

Note: - The source file allowed only one public class, if we are trying to declare multiple public classes then compiler generate error message.

example 1:- invalid

```
//Ratan.java
public class Test
{};
class A
{};
```

example 2:- valid

```
//Test.java
public class Test
{};
class A
{};
```

example 3:- invalid

```
//Test.java
public class Test
{};
public class A
{};
```

Application location:-

```
D:
|-->ratan
        |-->Sravya.java
```

Step-4:- Compilation process.

*Compile the java application by using **javac**command.*

Syntax:- *Javac filename*

Javac Test.java

Process of moveing application saveing location:-

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

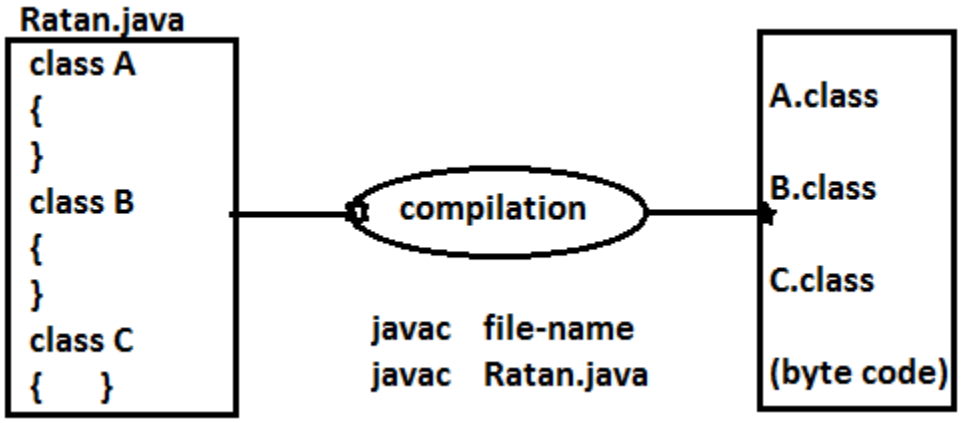
Whenever we are trying to perform compilation compiler perform fallowing actions.

- Compiler checks the syntax error, if syntax error are there compiler generate error message.
- If syntax errors are not present then compiler generate .***class*** *files.*

Note:- *in java* ***.class*** *file files generated by compiler at compilation time and .class file generation based on number of classes present in source file.*

If the source file contains 100 classes after compilation compiler generates 100 .class files

The compiler generate .class file and .class file contains byte code instructions it is intermediate code.



Process of compiling different files:-

D:

|-->ratan

|-->Sravya.java

|-->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*

Step-5:- Execution process.

Run /execute the java application by using ***java*** *command.*

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

Java Test

Whenever you are executing particular class file then JVM perform fallowing actions.

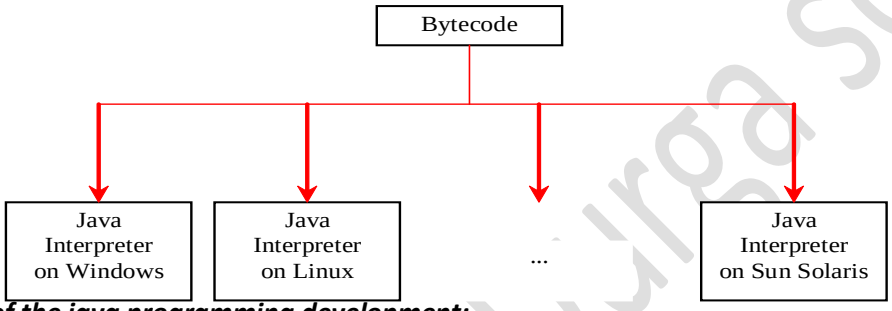
- It wills loads corresponding .class file byte code into memory. If the .class is not available JVM generate error message like **"Could not find main class**".
- After loading .class file byte into memory JVM calling main method to start the execution process. If the main method is not available compiler generate error message like ***"Main method not found in class A, please define the main method".***

Note 1:- compiler is translator it is translating ***.java*** *file to* ***.class*** *where as JVM is also a translator it istranslating* ***.class*** *file to* ***machine code.***

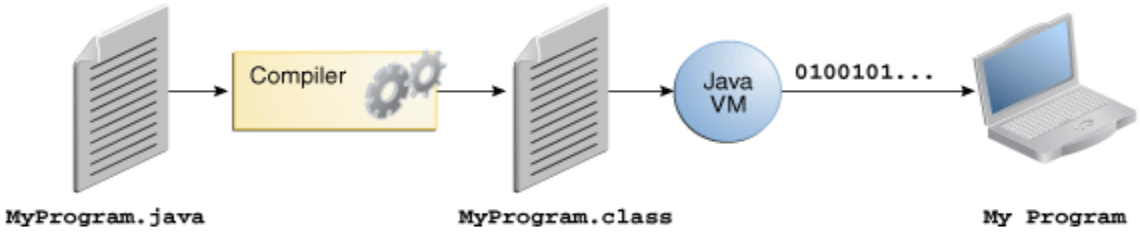
Note 2:- compiler understandable file format is ***.java*** *file but JVM understandable file format is* ***.class***

Executing all generated ***.class*** *files:-*

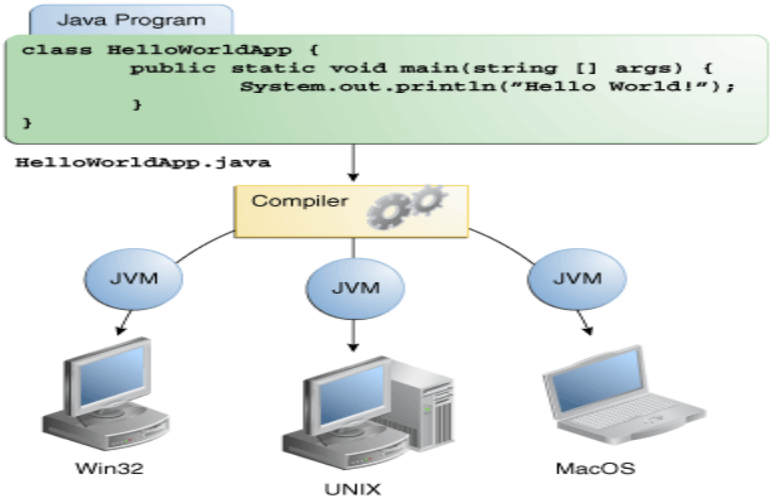
D:\ratan>java Test
Hi Ratan
D:\ratan>java A
Error: Main method not found in class A, please define the main method as:
public static void main(String[] args)
D:\ratan>java B
Error: Main method not found in class B, please define the main method as:
public static void main(String[] args)
D:\ratan>java XXX
Error: Could not find or load main class XXX

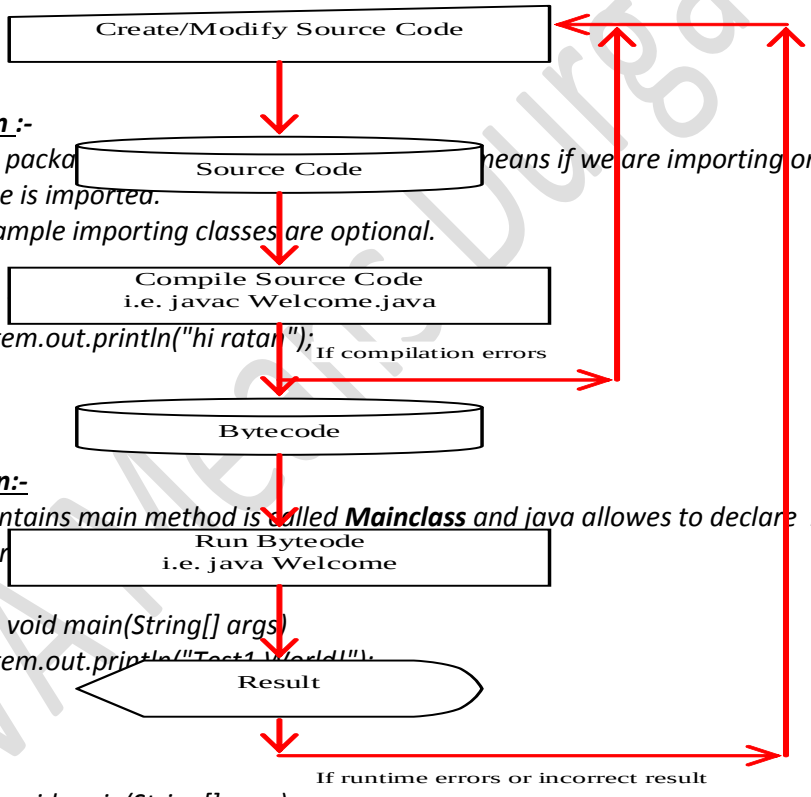


Environment of the java programming development:-



First program development :-





Example application :-

- *The default packa eans if we are importing or not by default that package is importea.*
- *In below example importing classes are optional.*

```
class Test
{       public static
        {       System.out.println("hi ratan");
        }
}
```

Example Application:-

*The class contains main method is called **Mainclass** and java allowes to declare multiple main class in a single sour*

```
class Test1
{       public static void main(String[] args)
        {       System.out.println("Test1 World!");
        }
}
class Test2
{       public static void main(String[] args)
        {       System.out.println("Test2 World!");
        }
}
class Test3
{       public static void main(String[] args)
        {       System.out.println("Test3 World!");
        }
}
```

D:\morn11>java Test1	D:\morn11>java Test2	D:\morn11>java Test3
Test1 World!	Test2 World!	Test3 World!

Class Elements:-

Class Test
{

1. *variables* — ***int a = 10;***
2. *methods* — ***void add() {business logic }***
3. *constructors* — ***Test() {business logic }***
4. *instance blocks* — ***{business logic }***
5. *static blocks* — ***static {business logic }***

}

Java codeing conventions :-

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** convension.*
- ✓ *The class name should be nouns.*

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

Interfaces :-

- ❖ Interface name starts with upper case and every inner word starts with upper case letter.
- ❖ *This convension is also known as **camel case** convension.*
- ❖ *The class name should be nouns.*

*Ex: **S**erializable **C**loneable **R**andom**A**ccess*

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.*

*Ex:- post() char**A**t() to**U**pper**C**ase() compare**T**o**I**gnore**C**ase()*

Variables:-

- ❖ Variable name starts with lower case letter and every inner word starts with upper case letter.
- ❖ This convention is also known as mixed case convention.

Ex :- out in page**C**ontext

Package :-

- ✓ *Package name is always must written in lower case letters.*

Ex :- java.lang java.util java.io …etc

Constants:-

- ❖ *While declaring constants all the words are uppercase letters .*

Ex: MAX_PRIORITY MIN_PRIORITY NORM_PRIORITY

NOTE:-The coding standards are applicable for predefined library not for user defined library .But it is recommended to fallow the coding standards for user defined library also.

Java Tokens:-

Smallest individual part of a java program is called Token. It is possible to provide any number of spaces in between two tokens.

Example:-

```
Class                   Test
        {
    Public              static      void        main
(       String[]            args    )
    {   int             a   =       10          ;
        System .        out         .           println     (
                "java  tokens");
        }
}
```

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

Java Comments :-

- Comments are used to provide detailed description about application.
- comments are non executable code.
-

There are 3 types of comments.

1) **Single line Comments:-**

   By using single line comments we are providing description about our program within a single line.

   Starts with……..>// (double slash)

   Syntax:- //description

2) **Multi line Comments:-**

   This comment is used to provide description about our program in more than one line.

   Syntax: - /*…….line-1
   ……..line-2
   */

3) **Documentation Comments:-**

   al we are using document comment to prepare API(Application programming interface) documents.. We will discuss later chapter.

   Syntax: - /**…….line-1
   *……line-2
   */

Example:-

```
/*project name:-green project
team size:-     6
team lead:- ratan
*/
class  Test//class declaration
{       //class starts
        public static void main(String[] args)// execution  starting point
        {       //main starts
                System.out.println("ratan");      //printing statement
        }//main ends
};//class ends
```

Print() vs Println ():-

Print():-used to print the statement in console and the control is present in the same line.

Example:- *System.out.print("Sravyainfotech");*
System.out.print("core java");
Output:-SravyaInfotechcorejava

Println():-used to print the statements in console but the control is there in next line.

Example:- *System.out.println("Sravyainfotech");*
System.out.println("core java");
Output: - ***Sravyainfotech***
Core java

Java Identifiers:-

any name in the java program like variable name, classname, methodname, interface name is called identifier.

```
class Test                      Test-------→identifier
{    void add()                 add-------→identifier
     {      int a=10;           a----------→identifiers
            int b=20;           b----------→identifiers
     }
};
```

Rules to declare identifiers:-

1. the java identifiers should not start with numbers, it may start with alphabet symbol and underscore symbol and dollar symbol.
   a. Int abc=10;-----→valid
   b. Int 2abc=20;----→not valid
   c. Int _abc=30;----→valid
   d. Int $abc=40;----→valid
   e. Int @abc=50;---→not valid

2. The identifier will not contains symbols like
   + , - , . , @ , # , *…….

3. The identifier should not duplicated.

```
class Test
{       void add()
        {       int a=10;
                int a=20;       } the identifier should not be duplicated.
        }
};
```

4. In the java applications it is possible to declare all the predefined class names and predefined interfaces names as a identifier. But it is not recommended to use.

```
class Test
{       public static void main(String[] args)
        {       int String=10;          //predefind String class
                int Serializable=20;    //predified Seriaiable Interface
                float Exception=10.2f;  //predefined Exception class
                System.out.println(String);
                System.out.println(Serializable);
                System.out.println(Exception);
        }
};
```

java flow control Statements:-

There are three types of flow control statements in java

1) *Selection Statements*
2) *Iteration statements*
3) *Transfer statements*

1. Selection Statements

a. If b. If-else c. switch

If syntax:-

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

- If is taking condition that condition must be Boolean condition otherwise compiler will raise compilation error.
- The curly brasses are optional whenever we are taking single statements and the curly brasses are mandatory whenever we are taking multiple statements.

Ex-1:-

```
class Test
{       public static void main(String[] args)
        {       int a=10;       int b=20;
                if (a<b)
                {       System.out.println("if body / true body");       }
                else
                {       System.out.println("else body/false body ");     }
        }
}
```

Ex -2:- For the if the condition it is possible to provide Boolean values.

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

Ex-3:-in c-language 0-false & 1-true but these convensions are not allowed in java.

```
class Test
{       public static void main(String[] args)
        {       if (0)
                {       System.out.println("true body");       }
                else
                {       System.out.println("false body");       }
        }
}
```

Switch statement:-

1) Switch statement is used to declare multiple selections.
2) Inside the switch It is possible to declare any number of cases but is possible to declare only one default.
3) Switch is taking the argument the allowed arguments are
   a. **Byte b. Short c. Int d.Char e.String(allowed in 1.7 version)**
4) Float and double and long is not allowed for a switch argument because these are having more number of possibilities (float and double is having infinity number of possibilities) hence inside the switch statement it is not possible to provide float and double and long as a argument.
5) Based on the provided argument the matched case will be executed if the cases are not matched default will be executed.

Syntax:-

```
switch(argument)
{
   case   label1 :        sop(" ");break;
    case  label2 :        sop(" ");break;
          |                       |
    default       :        sop("  ");       break;
}
```

Ex-1:Normal input and normal output.

```
class Test
{       public static void main(String[] args)
        {       int a=10;
                switch (a)
                {
                case 10:System.out.println("anushka");          break;
                case 20:System.out.println("nazriya");          break;
                case 30:System.out.println("samantha");         break;
                default:System.out.println("ubanu");            break;
                }
        }
}
```

Ex-2:-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".

```
class Test
{       public static void main(String[] args)
        {       int a=10;
                switch (a)
                {
                case 10:System.out.println("anushka");          break;
                case 10:System.out.println("nazriya");          break;
                case 30:System.out.println("samantha");         break;
                default:System.out.println("ubanu");            break;
                }       }}
```

Ex-3:Inside the switch for the case labels it is possible to provide expressions(10+10+20 , 10*4 , 10/2).

```
class Test
{       public static void main(String[] args)
        {       int a=100;
                switch (a)
                {
                case 10+20+70:System.out.println("anushka");          break;
                case 10+5:System.out.println("nazriya");              break;
                case 30/6:System.out.println("samantha");             break;
                default:System.out.println("ubanu");                  break;
                }
        }
}
```

Eg-4:- Inside the switch the case label must be constant values. If we are declaring variables as a case labels the compiler will show compilation error "constant expression required".

```
class Test
{       public static void main(String[] args)
        {       int a=10;       int b=20;       int c=30;
                switch (a)
                {
                case a:System.out.println("anushka");             break;
                case b:System.out.println("nazriya");             break;
                case c:System.out.println("samantha");            break;
                default:System.out.println("ubanu");              break;
                }
        }
}
```

Ex-5:-inside the switch the default is optional.

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

Ex 6:-Inside the switch cases are optional part.

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

Ex 7:-inside the switch both cases and default Is optional.

```
public class Test
{    public static void main(String[] args)
    {   int a=10;
       switch(a)
        {        }
    }
}
```

Ex -8:-inside the switch independent statements are not allowed. If we are declaring the statements that statement must be inside the case or default.

```
public class Test
{    public static void main(String[] args)
    {       int x=10;
       switch(x)
        {           System.out.println("Hello World");
         }
    }
}
```

Ex-9:-internal conversion of char to integer.
Unicode values a-97 A-65

```
class Test
{       public static void main(String[] args)
        {       int a=65;
                switch (a)
                {
                case 66:System.out.println("10");          break;
                case 'A':System.out.println("20");         break;
                case 30:System.out.println("30");          break;
                default: System.out.println("default");    break;
                }
        }
};
```

Ex -10: internal conversion of integer to character.

```
class Test
{       public static void main(String[] args)
        {       char ch='d';
                switch (ch)
                {
                case 100:System.out.println("10");       break;
                case 'A':System.out.println("20");       break;
                case 30:System.out.println("30");        break;
                default: System.out.println("default");  break;
                }
        }
};
```

Ex-11:-Inside the switch statement beak is optional. If we are not providing break statement at that situation from the matched case onwards up to break statement is executed if no beak is available up to the end of the switch is executed. This situation is called as fall though inside the switch case.

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

Ex-12:- inside the switch the case label must match with provided argument data type otherwise compiler will raise compilation error "incompatible types".

```
class Test
{       public static void main(String[] args)
        {       char ch='a';
                switch (ch)
                {
                case "aaa"      :System.out.println("samantha");        break;
                case  65        :System.out.println("anu");             break;
                case 'a'        :System.out.println("ubanu");           break;
                default         :System.out.println("default")         break;
                }
        }
}
```

Ex-13 :-inside the switch we are able to declare the default statement starting or middle or end of the switch.

```
class Test
{       public static void main(String[] args)
        {       int a=100;
                switch (a)
                {
                default: System.out.println("default");
                case 10:System.out.println("10");
                case 20:System.out.println("20");
                }
        }
};
```

Ex -14:-The below example compiled and executed only in above 1.7 version because switch is taking String argument from 1.7 version.

```
class Sravya
{    public static void main(String[] args)
   {  String str = “aaa”;
      switch (str)
       {      case "aaa"  : System.out.println("Hai");          break;
              case "bbb"  : System.out.println("Hello");    break;
              case "ccc"  : System.out.println("how");         break;
               default     : System.out.println("what");        break;
        }
   }
}
```

Ex-15:-inside switch the case labels must be within the range of provided argument data type otherwise compiler will raise compilation error “possible loss of precision”.

```
class Test
{       public static void main(String[] args)
        {       byte b=125;
                switch (b)
                {
                case 125:System.out.println("10");
                case 126:System.out.println("20");
                case 127:System.out.println("30");
                case 128:System.out.println("40");
                default:System.out.println("default");
                }
        }
};
```

Iteration Statements:-

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

1) For 2) while 3) do-while

for syntax:-

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

Flow of execution in for loop:-

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

The above process is repeated until the condition is false. If the condition is false the loop is stopped.

Initialization part:-

1) Initialization part it is possible to take the singe initialization it is not possible to take the more than one initialization.

With out for loop

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

By using for loop

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

Initialization:-

Ex1: Inside the for loop initialization part is optional.

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

Ex 2:- Instead of initialization it is possible to take any number of System.out.println("ratna") statements and each and every statement is separated by camas(,) .

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

Ex 3:- compilation error more than one initialization not possible.

```
class Test
{
        public static void main(String[] args)
        {
        for (int i=0,double j=10.8;i<10;i++)
                {
                System.out.println("Rattaiah");
                }
        }
}
```

Ex :-declaring two variables possible.

```
class Test
{
        public static void main(String[] args)
        {
                for (int i=0,j=0;i<10;i++)
                {

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

Conditional part:-

Ex 1:-inside for loop conditional part is optional if we are not providing condition at the time of compilation compiler will provide true value.

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

Increment/decrement:-

Ex1:- Inside the for loop increment/decrement part is optional.

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

Ex 2:- Instead of increment/decrement it is possible to take the any number of SOP() that and each and every statement is separated by camas(,).

```
class Test
{       public static void main(String[] args)
        {       for (int i=0;i<10;System.out.println("aruna"),System.out.println("nagalakshmi"))
                {       System.out.println("Rattaiah");
                        i++;
                }
        }
}
```

Note : ***Inside the for loop each and every part is optional.***
for(;;)----------→represent infinite loop because the condition is always true.

Example :-

```
class Test
{       static boolean foo(char ch)
        {       System.out.println(ch);
                return true;
        }
        public static void main(String[] args)
        {       int i=0;
                for (foo('A');foo('B')&&(i<2);foo('C'))
                {       i++;
                        foo('D');
                }
        }
};
```

Ex:- compiler is unable to identify the unreachable statement.

```
class Test
{
        public static void main(String[] args)
        {
                for (int i=1;i>0;i++)
                {
System.out.println("infinite times ratan");
                }
        System.out.println("rest of the code");
        }
}
```

ex:- compiler able to identify the unreachable Statement.

```
class Test
{
        public static void main(String[] args)
        {
                for (int i=1;true;i++)
                {

        System.out.println("ratan");
                }
        System.out.println("rest of the code");
        }
}
```

While loop:-

Syntax:- ***while (condition) //condition must be Boolean & mandatory.***
{ body;
}

Ex :-

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

Ex :- compilation error unreachable statement

```
class Test
{       public static void main(String[] args)
        {       int i=0;
                while (false)
                {       //unreachable statement
                        System.out.println("rattaiah");
                        i++;
                }
        }
}
```

Do-While:-

1) If we want to execute the loop body at least one time them we should go for do-while statement.
2) In the do-while first body will be executed then only condition will be checked.
3) In the do-while the while must be ends with semicolon otherwise we are getting compilation error.
4) do is taking the body and while is taking the condition and the condition must be Boolean condition.

Syntax:-do

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

Example :-

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

Example :- unreachable statement

```
class Test
{       public static void main(String[] args)
        {       int i=0;
                do
                {       System.out.println("rattaiah");
                }while (true);
                System.out.println("Sravyainfotech");//unreachable statement
        }
}
```

Example :-

```
class Test
{       public static void main(String[] args)
        {       int i=0;
                do
                {       System.out.println("rattaiah");
                }while (false);
                System.out.println("Sravyainfotech");
        }
}
```

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

1. **break 2. Continue 3. Return 4. Try**

break:- Break is used to stop the execution.

We are able to use the break statement only two places.

a. Inside the switch statement. b. Inside the loops.

if we are using any other place the compiler will generate compilation error message **" break outside switch or loop".**

Example :-*break means 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);
                }
        }}
```

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

```
class Test
{       public static void main(String[] args)
        {       if (true)
                {       System.out.println("ratan");
                        break;
                        System.out.println("nandu");
                }
        }};
```

Continue:-(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);
                }}}
```

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***		***0.0***
char	***2***	***0 to 6553***	***single space***
Boolean	***no-size***	***no-range***	***false***

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
System.out.println(10);         //invalid
```

Note :-

- *To represent numeric values (10,20,30…etc) use **byte,short,int,long.***
- *To represent point 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.***

User provided values are printed

```
int a = 10;
System.out.println(a);//10
boolean b=true;
System.out.println(b);//true
char ch='a';
System.out.println(ch);//a
double d=10.5;
System.out.println(d);//10.5
```

Default values(JVM assigned values)

```
int a;
System.out.println(a);//0
boolean b;
System.out.println(b);//false
char ch;
System.out.println(ch);//single space
double d;
System.out.println(d);//0.0
```

variable declarations:

int a=10; ----> integer variable
double d=10.5; ----> double variable
char ch='a'; ----> char variable
boolean b=true; ----> boolean variable
float f=10.5f; ----> float variable

Example :-//Test.java

```
class Test
{       public static void main(String[] args)
        {       float f=10.5;
                System.out.println(f);
                double d=20.5;
                System.out.println(d);
        }
}
```

D:\ratan>javac Test.java
Test.java:3: error: possible loss of precision
float f=10.5;
required: float
found: double

*in above example decimal value(10.5) by default double value hence compiler generating error message so to represent float value use **f** constant.*

To overcome above problem use "f" constant to represent float value.

```
float f =10.5f;  //valid
System.out.println(f);
```

Example :-

```
class Test
{       public static void main(String[] args)
        {       System.out.println("information about byte data type");
                System.out.println("byte size="+Byte.SIZE);
                System.out.println("byte min value="+Byte.MIN_VALUE);
                System.out.println("byte max value="+Byte.MAX_VALUE);

                System.out.println("information about int data type");
                System.out.println("int size="+Integer.SIZE);
                System.out.println("int min value="+Integer.MIN_VALUE);
                System.out.println("int max value="+Integer.MAX_VALUE);
        }
};
```

Java Variables:-

- Variables are used to hold the constant values by using these values we are achieving project requirements/functionality.
- While declaring variable must specify the type of the variable by using data types.
- Variables are also known as **fields** of a class or **properties** of a class.

Note :- All variables must have a type. You can use primitive types such as int, float, boolean, etc. Or you can use reference types, such as strings, arrays, or objects.

Variable declaration is composed of three components in order,

1) Zero or more modifiers. 2) The variable type. 3) The variable name.

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)      //execution starts from main method
        {       int a=10;                   //local variables
                int b=20;
                System.out.println(a);
                System.out.println(b);
        }
}
```

- We are able to access local variable 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;           //local variable
                System.out.println(a);    //possible
        }
        void mul()
        {         System.out.println(a);  //not-possible
        }
```

- For the local variables memory allocated when method starts and memory released when method completed.

```
        void m1()       //memory allocated when method starts
        {       //local variable
                int a=10;
                int b=20;
        }//memory released when method completed
```

Instance variables (non-static variables):-

- The variables which are declare inside a class and outside of methods those variables are called instance variables.
- Instance variables are visible inall ***methods and constructors***of a particular class.
- Instance variables are also known as **non-static** fields. And it is having global visibility.
- For instance variables memory allocated during object creation time and memory released when object is destroyed.
- Instance variables are stored in heap memory.

Areas of java language:-

There are two typesareasin java.

1) *Instance Area.*
2) *Static Area.*

Instance Area:-

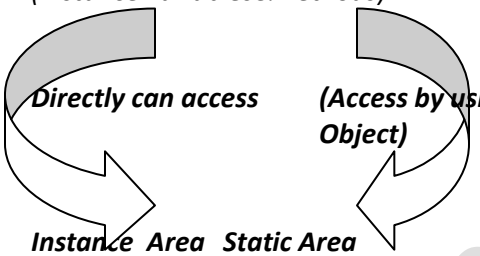
void m1()**//instance method**
{ ***Body //instance area***
}

Static Area:-

Static voidm1()***//static method***
{ ***body //static area***
}

Instance variable accessing:-

(Instance variables&methods)



Example:-

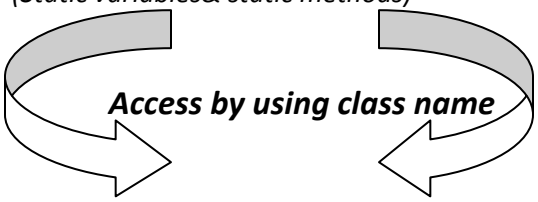
```
class Test
{       //instance variables
        int a=10;       int b=20;
        //static method
        public static void main(String[] args)
        {       //Static Area
                Test t=new Test();
                System.out.println(t.a);
                System.out.println(t.b);
    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

Static variables (class variables):-

- The variables which are declared inside the class and outside of the methods with static modifier is called static variables.
- Static variables are visible all methods and constructors of a particular class.
- Static variables memory allocated at the time of .class file loading and memory released at .class file unloading time.
- Static variables are stored in non-heap memory.

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
        }
        //instance method
        void m1()       //user defined  method called by user in main method
        {       System.out.println(Test.a);
                System.out.println(Test.b);
        }
};
```

Calling of static variables:-

We are able to access the static members inside the static area in three ways.

- ✓ Directly possible.
- ✓ By using class name.
- ✓ By using reference variable.

In above three approaches second approach is best 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: - ***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
{       //instance variable
        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
        }
};
```

Example:-

```
class Test
{       // instance variables
        int a=10;
        int b=20;
        static int c=30;  //static variables
        static int d=40;
        void m1() //instance method
        {       System.out.println(a);
                System.out.println(b);
                System.out.println(Test.c);
                System.out.println(Test.d);
        }
        static void m2()  //static method
        {       Test t = new Test();
                System.out.println(t.a);
                System.out.println(t.b);
                System.out.println(Test.c);
                System.out.println(Test.d);
        }
        public static void main(String[] args)
        {       Test t = new Test();      t.m1();  //instance method calling
                Test.m2();      //static method calling
        }
};
```

Variables VS default values:-

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

```
class Test
{       int a;
        boolean b;
        public static void main(String[] args)
        {       //access the instance variables  by using object
                Test t=new Test();
                System.out.println(t.a);
                System.out.println(t.b);
        }
};
```

Case 2:-for the static variables JVM will assign default values.

```
class Test
{       static int a;
   static float b;
        public static void main(String[] args)
        {       //access the static variable  by using class Names
                System.out.println(Test.a);
                System.out.println(Test.b);
        }
};
```

Case 3:-

- *For the instance and static variables JVM will assign default values butfor the local variables the JVM won't provide default values.*
- *In java before using localvariables must initialize some values to the variables otherwise compiler will raise compilation error "variable a might not have been initialized"***.**

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

D:\>javac Test.java
Test.java:6: variable a might not have been initialized
System.out.println(a);

Class Vs Object:-

- ***Class is a logical entity it contains logics where as 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 (blue print) it is possible to create multiple objects but every object occupies memory.***
- *Civil engineer based on blue print of house it is possible to create multiple houses in different places but every house required some area.*
- ***We are declaring the class by using class keyword but we are creating object by using new keyword.***
- *We are able to create object in different ways like*
    - *By using new operator*
    - *By using clone() method*
    - *By using new Instance()*
    - *By using factory method.*
    - *By using deserialization….etc*

  But we are able to declare the class by using class keyword.
- ***We will discuss object creation in detailed in constructor concept.***

Instance vs. Static variables:-

- *For the instance variables the JVM will create separate memory for each and every object it means separate instance variable value for each and every object.*
- *For the static variables irrespective of object creation per class single memory is allocated, here all objects of that class using single copy.*

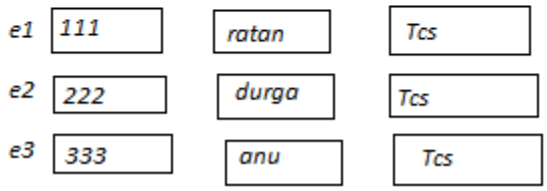
Example :-

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

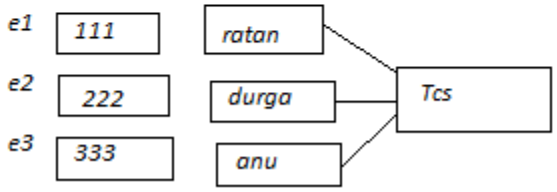
Instance variable vs static variable :-

```
class Emp
{       //instance variable
        int eid;
        String ename;
        String company;

}
```




```
class Emp
{       //instance variable
        int eid;
        String ename;
        //static variable
        static String company;
}
```



Note 1 :- *The variables which are declared inside the method or constructor or blocks those variables are called local variables and we are able to access(usage) local variables only inside the method or constructor or blocks , for the local variables memory is allocated when method starts and memory id destroyed when method ends and these variables are stored in stack memory & call the local variables directly .*

Note 2:-*The variables which are declared inside the class and outside of the method those variables are called instance variables and we are able to access (usage) instance members inside the class,for the instance variables memory is allocated duringobject creation and memory destroyed when object is destroyed & these variables are stored in heap memory & call the instance members (variables & methods) by using object.*

Note 3 :- *The variables which are declared inside the class and outside of the method with static modifier those variables are called static variables & we are able to access (usage) static members inside the class& for the static variables memory is allocated during .class loading and memory destroyed .class file unloading & access the static members(variables & methods) by using class Name.*

Different ways to initialize the variables :-

```
class Test
{       int s=10;
        int a,b,c;
        int x,y,z=10;
        int i=10,j=20,k;
        static int p=100,q=200,r=300;
        public static void main(String[] args)
        {       Test t = new Test();
                System.out.println(s);
                System.out.println(t.a+" "+t.b+" "+t.c);
                System.out.println(t.x+" "+t.y+" "+t.z);
                System.out.println(t.i+" "+t.j+" "+t.k);
                System.out.println(Test.p+" "+Test.q+" "+Test.r);
        }
}
```

Summary of variables:-

Characteristic	***Local variable***	***instance variable***	***static variables***
where declared	*inside method or Constructor or block.*	*inside the classoutside Of methods*	*inside the class outside of methods.*
Use	*within the method*	*inside the class all the methods and constructors.*	*inside the class all Methods& constructors.*
When memory allocated	*whenmethod starts*	*when objectcreated*	*when .class file loading*
When memory destroyed	*when method ends.*	*When object destroyed*	*when .class unloading.*
Initial values	none, must initialize the value *before first use.*	default values are *Assigned by JVM.*	*default values are Assigned by JVM.*
Relation with Object	*no way related to object.*	*for every object one copy Of instance variable created It means memory.*	*for all objects one copy is created. Single memory.*
Accessing	*directly possible.*	*By using object name.* ***Test t = new Test(); System.out.println(t.a);***	*by using class name.* ***System.out.println(Test.a);***
Memory	*stored in stack memory.*	*Stored in heap memory*	*non-heap memory.*

+ operator:-

- ✓ One operator with multiple behaviors is called operator over loading but java is not supporting operator overloading concept and only one overloaded operator in java is **+**
  - If two operands are integers then + perform addition.
  - If at least one operand is String then + perform concatenation.

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

Java.util.Scanner(Dynamic Input):-

1. Scanner class present in **java.util** package and it is introduced in 1.5 version.
2. Scanner class is used to take dynamic input from the keyboard.

Scanner s = new Scanner(System.in);
to get int value ----> s.nextInt()
to get float value ---> s.nextFloat()
to get byte value ---> s.nextbyte()
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);       //used to take dynamic input from keyboard
                System.out.println("enter emp hobbies");
                String ehobbies = s.nextLine();
                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("*****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
        }
}
```

Java Methods (behaviors):-

- Methods are used to write the business logics of the project.
- Coding conversion of method is method name starts with lower case letter if method contains more than one word then every inner word starts with uppercase letter.

Example:- post() , charAt() , toUpperCase() , compareToIgnoreCase()……etc

There are two types of methods

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

- Inside the class it is possible to declare any number of instance methods &static methods based on the developer requirement.
- It will improve the reusability of the code and we can optimize the code.

Note :- 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: - for the instance members memory is allocated during object creation hence access the instance members by using object(reference-variable).

Syntax:-

```
Void m1() {  }  //instance method
Test t = new Test();
 Objectname.instancemethod( ); //calling  instance method
t.m1( );
```

static method:-

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

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

Syntax:-

```
Static void m2()  { } //static method
Classname.staticmethod( ); //   call static method by using class name
Test.m2( );
```

Syntax:-

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

Modifiers-list ------→ represent access permissions.---- →***[optional]***
Return-type ------→ functionality return value---- →***[mandatory]***
Method name ------→ functionality name ----- →***[mandatory]***
Parameter-list ------→ input to functionality ---- →***[optional]***
Throws Exception ------→ representing exception handling--- →***[optional]***

Example:- ***Public void m1()***
Private int m2(int a,int b)

Method Signature:-

Method Signature is nothing but name of the method and parameters list. Return type and modifiers list not part of a method signature.

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

Every method contains two parts.

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

Ex:- *void m1()* ------> ***method declaration***
{ Body (Business logic); -----> ***method implementation***
}

Example-1 :- instance methods without arguments.

Instance methods are bounded with objects hence call the instance methods by using object name(reference variable).

```
class Test
{       //instance methods
        void Sravya()   {       System.out.println("Sravya");                   }
        void soft()     {       System.out.println("software solutions");      }
        public static void main(String[] args)
        {       Test t=new Test();
                t.Sravya();     //calling of instance method by using object name [ t ]
                t.soft();       //calling of instance method by using object name [ t ]
        }
}
```

Example-2:-instance methods with parameters.

- If the method is taking parameters at that situation while calling that method must provide parameter values then only that method will be executed.
- Parameters of methods is nothing but inputs to method.
- While passing parameters number of arguments and argument order is important.

```
void m1(int a)                  -->t.m1(10);            -->valid
void m2(int a,int b)            -->t.m2(10,'a');        -->invalid
void m3(int a,char ch,float f)  -->t.m3(10,'a',10.6);   -->invalid
void m4(int a,char ch,float f)  -->t.m4(10,10,10.6);    -->invalid
void m5(int a,char ch,float f)  -->t.m3(10,'c');        -->invalid
```

```
class Test
{       //instance methods
        void m1(int i,char ch)  //local variables
        {       System.out.println(i+"-------"+ch);          }
        void m2(double d ,String str) //local variables
        {       System.out.println(d+"------"+str);          }
        public static void main(String[] args)
        {       Test t=new Test();
                t.m1(10,'a');           //m1() method calling
                t.m2(10.2,"ratna");     //m2() method calling
        }
}
```

Example-3 :- static methods without parameters.

Static methods are bounded with class hence call the static members by using class name.

```
class Test
{       //static methods
        static void m1()
        {       System.out.println("m1 static method");}
        static void m2()
        {       System.out.println("m2 static method");}
        public static void main(String[] args)
        {       Test.m1();      //call the static method by using class name
                Test.m2();      //call the static method by using class name
        }
}
```

Example -4 :-static methods with parameters.

```
class Test
{       //static methods
        static void m1(String str,char ch,int a)    //local variables
        {       System.out.println(str+"---"+ch+"---"+a);
        }
        static void m2(boolean b1,double d)  //local variables
        {       System.out.println(b1+"---"+d);
        }
        public static void main(String[] args)
        {       Test.m1("ratan",'a',10);        //static m1() calling by using class name
                Test.m2(true,10.5);             //static m2() calling by using class name
```

```
        }
};
```

Example 6:-For java methods it is possible to provide Objects as a parameters(in real time project level).

Case 1:- project code at student level.

```
class X{}
class Emp{}
class Y{}
class Test
{       void m1(X x ,Emp e)
        {       System.out.println("m1 method");
        }
        static void m2(int a,Y y)
        {       System.out.println("m2 method");
        }
        public static void main(String[] args)
        {       Test t = new Test();
                X x = new X();
                Emp e = new Emp();
                t.m1(x,e);              //calling of instance method by using object
                Y y = new Y();
                Test.m2(10,y);          //calling of static method by using class-name
        }
}
```

Case 2: project code at realtime project level

```
class X{}
class Emp{}
class Y{}
class Test
{       void m1(X x ,Emp e)//taking objects as a parameter
        {       System.out.println("m1 method");
        }
        static void m2(int a,Y y) //taking objects as a parameter
        {       System.out.println("m2 method");
        }
        public static void main(String[] args)
        {       new Test().m1(new X(),new Emp());
                Test.m2(10,new Y());
        }
}
```

Example-7 :- method vs. data- types

- By default the numeric values are integer valuesbut to represent other format like byte, short perform typecasting.
- By default the decimal values are double values but to represent float value perform typecasting or use "**f**" constant.
  - **double d=10.5; float f=20.5f;**

```
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);         //by default 10 is int value
                t.m2((short)20);        //by default 20 is int value
                t.m3(30);
                t.m4(40);
                t.m5(10.6f);            //by default 10.6  value is  double
                t.m6(20.5);                     t.m7('a');              t.m8(true);
        }
}
```

Example-8:-method calling

m1()--→calling --→m2()----→calling------→ m3()

m1()<-------after completion-m2()<------after completion m3()

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

Example-9:-

For java methods return type is mandatory otherwise the compilation will generate error message "invalid method declaration; return type required".

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

Example-10 :-

- Inside the java class it is not possible to declare two methods with same signature , if we are trying to declare two methods with same signature compiler will raise compilation error "m1() is already defined in Test "
- Java class not allowed Duplicate methods.
- It is possible to write,

  void m1()

  Void m1(int a)

  But the above method signatures are different it is method overloading concept.

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

Example-11 :-

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

```
class Test
{       void m1()
        {       void m2() //inner method
                {       System.out.println("m2() inner method");          }
        System.out.println("m1() outer method");
        }
        public static void main(String[] args)
        {       Test t1=new Test();
                t.m1();
        }
};
```

Example-12 :- methods vs return type.

1. Everyfunctionality is able to return some functionality return value just like acknowledgement.
   Ex : whenwe applied for driving license then after one month we will receive ID card.
2. In java every method is able to return some return value (int , char , String…….) if we are not interestedthen return nothing by using **void** return type.
3. If the method is having return type other than void at that situation must return the value by using **return** keyword otherwise compiler will generate error message "**missing return statement**"

   Ex : below syntax invalid because method must return int value by using return statement.

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

   Ex :- the below example is valid because it is returning int value by using return statement.

   ```
   int m1()
   {   System.out.println("Anushka");
           return 100;
    }
   ```

4. Inside the method we are able to use only one return statement(except flow control statement) that must be last statement of the method otherwise compiler will generate error message "**unreachable statement**".

   Ex : the below example is invalid because return statement is must be last statement.

   ```
   int m1()
   {       return 100;
           System.out.println("Anushka");
    }
   ```

   Ex :the below example valid because return statement is last statement .

   ```
   int m1()
   ```

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

5. Every method is able to return value but holding (storing) that return value is optional ,but it is recommended to hold the value.

```
class Test
{       int m1(int a,char ch)       //local variables
        {       System.out.println("***m1 method***");
                System.out.println(a+"---"+ch);
                return 100;         //method return value
        }
        boolean m2(String str1,String str2)         //local variables
        {       System.out.println("****m2 method****");
                System.out.println(str1+"---"+str2);
                return true;        //method return value
        }
        String m3()
        {       return "ratan";     } //method return value
        public static void main(String[] args)
        {       Test t=new Test();
                int x = t.m1(10,'a');                           //m1(int,char) method calling
                System.out.println("m1() return value-->"+x);    //printing m1() method return value
                boolean b = t.m2("ratan","anu");                //m2(String,String) method calling
                System.out.println("m2() return value-->"+b); //printing m2() method return value
                String str = t.m3();                            //m3() method calling
                System.out.println("m3() return value-->"+str);  //printing m3() method return value
        }//end main
}//end class
```

Example-13:- ***methods vs. return variables***

Returns local variable as a return value

```
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(x);
        }
}
```

D:\>java Test
m1() method
100

Returns instance variable as a return value(no local variable)

```
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(x);
        }
}
```

D:\>java Test
m1() method
10

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

Example 14:- ***The java class is able to return user defined class as a return value.***

```
class Person
{       void eat(){System.out.println("person takes 4-idles");}
}
class Ratan
{       void eat(){System.out.println("ratan takes 10-idles");}
}
class RatanKid
{       void eat(){System.out.println("person takes 2-idles");}
}
class Test
{       Person m1()
        {       System.out.println("m1 method");
                Person p = new Person();
                return p;
        }
        Ratan m2()
        {       System.out.println("m2 method");
                Ratan r = new Ratan();
                return r;
        }
        RatanKid m3()
        {       System.out.println("m3 method");
                RatanKid k = new RatanKid();
                return k;
        }
        public static void main(String[] args)
        {       Test t =new Test();
                Person p = t.m1();
                p.eat();
                Ratan r = t.m2();
                r.eat();
                RatanKid k = t.m3();
                k.eat();
        }
};
```

The project level used code:-

```
Person m1()
{       System.out.println("m1 method");
        return new Person();
}
Ratan m2()
{       System.out.println("m2 method");
        return new Ratan();
}
RatanKid m3()
{       System.out.println("m3 method");
        return new RatanKid();
}
```

Example 15:

Note :- ***This keyword representing current class objects.***

In java method is able to return current class object return value in two ways.

1) By creation of object.
2) Return **this** keyword because it is representing current class object.

In above two approaches 2nd approach is recommended to return the current class object.

```
class Test
{       Test m1()       //first approach to return same class(Test) object
        {       return new Test();
        }
        Test m2()       //second approach to return same class(Test) object
        {       return this;    }
        public static void main(String[] args)
        {       Test t = new Test();      //it creates object of Test class
                System.out.println(t.getClass());
                Test t1 = t.m1();         //m1() method return Object of Test class
                System.out.println(t1.getClass());
                Test t2 = t.m2();         //m2() method return Object of Test class
                System.out.println(t1.getClass());
        }
};
```

Example 16 :- Template method:-

- Let Assume to complete your task you must call four methods at that situation you must remember number of methods and order of calling.
- To overcome above limitation take one x( ) method it is calling four methods internally to complete our task then instead of calling four methods every time call x( ) method that perform our task that x( ) method is called template method.

```
class Test
{       void customer()         {       System.out.println("customer part");}
        void product()          {       System.out.println("product part");      }
        void selection()        {       System.out.println("selection part");    }
        void billing()          {       System.out.println("billing part");      }
        void deliveryManager()          //template method
        {       System.out.println("****Template method***");
                //template method is calling four methods in order to complete our task.
                customer();             product();              selection();            billing();
        }
        public static void main(String[] args)
        {       //normal approach
                Test t = new Test();
                t.customer();           t.product();            t.selection();          t.billing();
                //by using template method
                Test t1 = new Test();
                t1.deliveryManager();  //this method is calling four methods to complete our task.
        }
};
```

Example 17:- Method recursion*A method is calling itself during execution is called recursion.*

Example 1:- (normal output)

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

Example 2:- (StackOverFlowError)

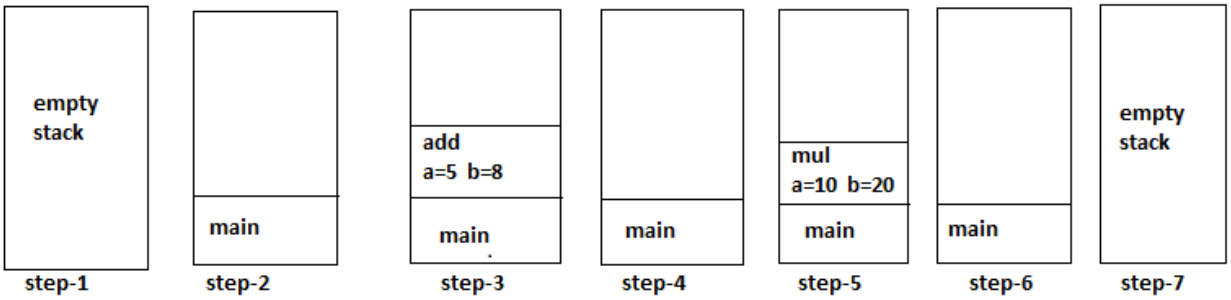
```
class RecursiveMethod
{       static void recursive(int a)
        {       System.out.println("number is :- "+a);
                if (a==0)
                {return;
                }
```

```
            recursive(++a);
        }
        public static void main(String[] args)
        {       RecursiveMethod.recursive(10);
        }
};
```

Example 18 :- Stack Mechanism:-

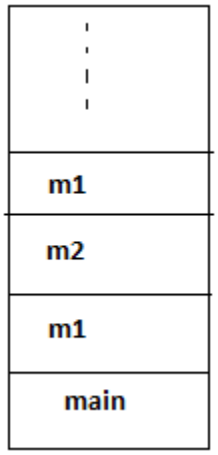
- In java program execution starts from main method, just before program execution JVM creates one empty stack for that application.
- Whenever JVM calling particular method then that method entry and local variables of that method stored in stack memory.
- When the method exists, that particular method entry and local variables of that method are deleted from memory that memorybecomes available to other called methods.
- Based on 2 & 3 the local variables are stored in stack memory and for these variables memory is allocated when method starts and memory is deleted when program ends.
- The intermediate calculations are stored in stack memory at final if all methods are completed that stack will become empty then that empty stack is destroyed by JVM just before program completes.
- The empty stack is created by JVM and at final empty stack is destroyed by JVM.

```
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 18:-we are getting 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();
        }
}
```



this keyword:-

this keyword is holding current class reference variable and it is used to represent,

a. **Current class variables.**
b. **Current class methods.**
c. **Current class constructors.**

Current class variables:-

This keyword not required:-

```
class Test
{       //instance variables
        int a=100;
        int b=200;
        void add(int i,int j)//local variables
        {       System.out.println(a+b);//instance  variables addition
                System.out.println(i+j);//local variables addition
        }
        public static void main(String[] args)
        {       Test t = new Test();
                t.add(10,20);
        }
}
```

In bove exmple instance variables and local variables having different names so this keyword not required.

This keyword required:-

```
class Test
{       //instance variables
        int a=100;
        int b=200;
        void add(int a,int b)//local variables
        {       System.out.println(a+b);//local variables addition
                System.out.println(this.a+this.b);//instance variables addition
        }
        public static void main(String[] args)//static method
        {       Test t = new Test();
                t.add(10,20);
        }
}
```

In bove example intstance variables and local variables having same name at that situation we are able to print local variables directly but to represent instance variables use ***this*** *keyword.*

Conversion of local variables to instacevariables:-

This keyword not Required:-

```
class Test
{       int i, j; //instance variables
        void values(int val1,int val2)//local variables
        {       //conversion of local variables to instance variables (passing local variable
                values to instance variables)
                i=val1;
                j=val2;
        }
        void add(){System.out.println(i+j);}
        void mul(){System.out.println(i*j);}
        public static void main(String[] args)
        {       Test t=new Test();
                t.values(100,200);
                t.add();        t.mul();
        }//end main
}//end class
```

In above example local variables and instance variables having different names hence this keyword not required.

This keyword Required:-

```
class Test
{       //instance variables
        int val1;
        int val2;
        void values(int val1,int val2)//local variables
        {       //printing local variables
                System.out.println(val1);
                System.out.println(val2);
                //conversion of localvariables to instance varaibles (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();
        }//end main
}//end class
```

In above example local variables and instance variables having same names so while conversion to represent instance vaariable use this keyword.

Current class method calling:-

Ex:- to call the current class mehods this keyword optional hence the both examples are same.

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

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


CONSTRUCTORS:-

Object creation syntax:-

Test t = new Test();

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

1. ***When we create new instance (Object) of a class using new keyword, a constructor for that class is called.***
2. ***Constructors are used to initialize the instance variables of a class***

New :-

- *new keyword is used to create object in java.*
- *Different approaches are there create objects like*
  - *By usinginstance factory method.*
  - *By using stati factory method*
  - *By using pattern factory method*
  - *By using new operator.*
  - *By using Deserialization .*
  - *By using newInstance() method.*
  - *By using clone() method…….etc*
- *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) ***Constructor is able to take parameters.***
3) ***Constructor not allowed explicit return type (return type declaration not possible).***

There are two types of constructors,

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

Default Constructor:-

1) If we are not write constructor for a class then compiler generates one constructor for you that constructor is called default constructor. And it is not visible in code.
2) Compiler generates Default constructor inside the class when we are not providing any type of constructor (0-arg or parameterized).
3) The compiler generated default constructor is always **0-argumnet**constructor with **empty implementation (empty body).**

Application before compilation :-

```
class Test
{       void m1()       {       System.out.println("m1 method"); }
        public static void main(String[] args)
        {       //at object creation time 0-arg constructor executed
        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 **"Test()"**constructor inside the since not there 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()
        {       }
        public static void main(String[] args)
        {//object creation time 0-arg constructor executed
        Test t = new Test();
        t.m1();
        }
}
```

In above example at run time JVM execute compiler provide default constructor during object creation.

Example-3:-

- *Inside the class if we declaring at least one constructor (either 0-arg or parameterized) the compiler won't generate default constructor.*
- *if we are trying to compile below application the compiler will generate error message "cannot find symbol" .*
- *Inside the class if we are declaring at least one constructor the compiler won't generate default constructor.*

```
class Test
{       Test(int i)
        {       System.out.println(i);          }
        Test(int i,String str)
        {       System.out.println(i);
                System.out.println(str);        }
        public static void main(String[] args)
        {Test t1=new Test(); // in this line compiler is searching 0-arg cons but not available
        Test t2=new Test(10);
        Test t3=new Test(100,"rattaiah");
        }
```

```
}
```

Exampl-2:- default constructor execution vs. user defined constructor execution

Case 1:- default constructor execution process.

```
class Employee
{       //instance variables
        int eid;
        String ename;
        double esal;
        void display()
        {       //printing instance variables values
                System.out.println("****Employee details****");
                System.out.println("Employee name :-->"+ename);
                System.out.println("Employee eid :-->"+eid);
                System.out.println("Employee sal :-->"+esal);
        }
        public static void main(String[] args)
        {       // during object creation 0-arg cons executed then values are assigned
                Employee e1 = new Employee();
                e1.display();
        }
}
```

```
D:\morn11>javac Employee.java
D:\morn11>java Employee
****Employee details****
Employee name :-->null
Employee eid :-->0
Employee sal :-->0.0
```

Note: - in above example during object creation time default constructor is executed with empty implementation and initial values of instance variables (default values) are printed .

Case 2:- user defined o-argument constructor execution process.

```
class Employee
{       //instance variables
        int eid;
        String ename;
        double esal;
        Employee()//user defined 0-argument constructor
        {       //assigning values to instance values during object creation
                eid=111;
                ename="ratan";
                esal =60000;
        }
        void display()
```

```
    {  //printing instance variables values
        System.out.println("****Employee details****");
        System.out.println("Employee name :-->"+ename);
        System.out.println("Employee name :-->"+eid);
        System.out.println("Employee name :-->"+esal);
    }
    public static void main(String[] args)
    {// during object creation 0-arg cons executed then values are assigned
        Employee e1 = new Employee();
        e1.display();//calling display method
    }
}
```

Compilation & execution process:-

```
D:\morn11>javac Employee.java
D:\morn11>java Employee
****Employee details****
Employee name :-->ratan
Employee name :-->111
Employee name :-->60000.0
```

Note: - in above example during object creation user provided 0-arg constructor executed used to initialize some values to instance variables.

The problem with above approach:-

When we create two employee objects but every object same values are initialized.

```
Emp e1 = new Emp();
e1.display();
Emp e2 = new Emp();
e2.display();
D:\morn11>java Employee
****Employee details****
Employee name :-->ratan
Employee name :-->111
Employee name :-->60000.0
Employee name :-->ratan
Employee name :-->111
Employee name :-->60000.0
```

To overcome above limitation just use parameterized constructor to initialize different values.

Case 3:- user defined parameterized constructor execution.

User defined parameterized constructors:-

- Inside the class if the default constructor is executed means the initial values of variables only printed.
- To overcome above limitation inside the class we are declaring user defined 0-argument constructor to assign some values to instance variables but that constructor is able to initialize the values only for single object.
- To overcome above limitation declare the parameterized constructor and pass the different values during different objects creation.
- Parameterized constructor is nothing but the constructor is able to parameters.

Example :-

```
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()
        {       //printing instance variables values
                System.out.println("****Employee details****");
                System.out.println("Employee name :-->"+ename);
                System.out.println("Employee name :-->"+eid);
                System.out.println("Employee name :-->"+esal);
        }
        public static void main(String[] args)
        {       // during object creation parameterized  constructor executed
                Employee e1 = new Employee(111,"ratan",60000);
                e1.display();
                Employee e2 = new Employee(222,"anu",70000);
                e2.display();
                Employee e3 = new Employee(333,"Sravya",80000);
                e3.display();
        }
}
```

Note: - by using parameterized constructor we are able to initialize values to instance variables during object creation and it is possible to initialize different values to different objects during object creation time.

Note :- the main objective of constructor is initialize some values to instance variables during object creation time.

Example :-

- ***Constructor s are performing fallowing operations***
  - ***Constructors are useful to initialize some user provided values to instance variables during object creation.***
  - ***Constructors are used to write the functionality of project that functionality is executed during object creation.***
- ***Inside the class it is possible to declare multiple constructors.***

```
class Test
{       Test()   //user defined 0-arg constructor
        {
                System.out.println("0-arg cons logics");
        }
        Test(int a,int b) //user defined parameterized constructor
        {
                System.out.println("2-arg cons logics");
        }
        void m1()  { System.out.println("m1 method"); }
        public static void main(String[] args)
        {       Test t1 = new Test();           t1.m1();
                Test t2 = new Test(10,20);      t2.m1();
        }
}
```

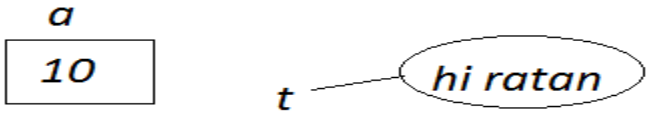
Example-4:- constructors vs all data-types

```
class Test
{       Test(byte a)    {       System.out.println("Byte value-->"+a);          }
        Test(short a)   {       System.out.println("short value-->"+a);         }
        Test(int a)     {       System.out.println("int value-->"+a);           }
        Test(long a)    {       System.out.println("long value is-->"+a);       }
        Test(float f)   {       System.out.println("float value is-->"+f);      }
        Test(double d)  {       System.out.println("double value is-->"+d);     }
        Test(char ch)   {       System.out.println("character value is-->"+ch); }
        Test(boolean b) {       System.out.println("boolean value is-->"+b);    }
        public static void main(String[] args)
        {       Test t1=new Test((byte)10);
                Test t2=new Test((short)20);
                Test t3=new Test(30);
                Test t4=new Test(40);
                Test t5=new Test(10.5);
                Test t6=new Test(20.5f);
                Test t7=new Test('a');
                Test t8=new Test(true);
        }
```

```
}
```

Example :-

- ***a*** *is variable of primitive type such as int,char,double,boolean…etc*
- ***t****isreference variable &it is the memory address of object.*
- *Reference variable used to hold particular object & class decides object creation.*




```
class Test
{       Test(String str){}
        public static void main(String[] args)
        {       //a is primitive variable
                int a=10;
                System.out.println(a);
                //t is reference variable
                Test t = new Test("hi ratan");
                System.out.println(t);
        }
}
```

This keyword :-

To call Current class constructoruse 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
this(10 , ”ratan” , ’a’) ----→ current class 3-arg constructor calling

Example-1:-

To call the current class contructor use 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");
                System.out.println(a);
        }
        Test(char ch,int a)
        {       System.out.println("2-arg constructor logics");
                System.out.println(ch+"----"+a);
        }
```

```
    public static void main(String[] args)
    {       Test t = new Test();       //at object creation time 0-arg costructor executed
    }
}
```

Example 2:-

Inside the constructor this keyword must be first statement otherwise compiler generate error message "**call to this must be first statement in constructor**".

Constructor calling must be first statement in constructor.

No compilation error:-(this keyword first statement)

```
    Test()
    {       this(10);  //current class 1-argument constructor calling
            System.out.println("0 arg");
    }
    Test(int a)
    {       this(10,20); //current class 2-argument constructor calling
            System.out.println(a);
    }
```

Compilation error:-(this keyword not a first statement)

```
    Test()
    {       System.out.println("0 arg");
            this(10);  //current class 1-argument constructor calling
    }
    Test(int a)
    {       System.out.println(a);
            this(10,20); //current class 2-argument constructor calling
    }
```

Example-3:-

1. Constructor calling must be first statement in constructor it means this keyword must be first statement in constructor.
2. In java One constructor is able to call only one constructor at a time it is not possible to call more than one constructor.

Compilation error:-

```
    Test()
    {       this(100);//1-arg constructor calling
            this('g',10);//2-arg constructor calling[compilation error]
            System.out.println("0-arg constructor  logics");
    }
```

Note :-

Every object creation having three parts.

1) ***Declaration:-***

```
Test t;                 //t is Test type
Student s;              //s is Student  type
A a;                    //a is A type
```

2) ***Instantiation:-(just object creation)***

```
new Test();             //Test object
new Student();          //student object
new A();                //A object
```

3) ***initialization:-(during object creation perform initialization)***

```
new Test(10,20);            //during object creation 10,20 values initialized
new Student("ratan",111);   //during object creation values are initialized
new A('a',true)             //during object creation values are initialized
```

Example :- *in java object creation done in 2-ways.*

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 [having reference variable]
                Test t = new Test();
                t.m1();
                //nameless object     [without reference variable`]
                new Test().m1();
        }
}
```

Example *:Two formats of object creation.*

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();
        }
}
```

Example :- assign values to instance variables [constructor vs. method]

```
class Student
{       //instance variables
```

```
        int sid;
        String sname;
        int smarks;
        //constructor assigning values to instance variables
        Student(int sid,String sname,int smarks)      //local variables
        {       //conversion [passing local variable values to instance variables]
                this.sid=sid;           this.sname=sname;           this.smarks=smarks;
        }
        //method assigning values to instance variables
        void assign(int sid,String sname,int smarks)  //local variable
        {       //conversion
                this.sid=sid;           this.sname=sname;           this.smarks=smarks;
        }
        void disp()
        {       System.out.println("****student Details****");
                System.out.println("student name = "+sname);
                System.out.println("student id = "+sid);
                System.out.println("student mrks = "+smarks);
        }
        public static void main(String[] args)
        {       Student s = new Student(111,"ratan",100);
                s.assign(222,"anu",200);
                s.disp();
        }
}
```

Example :- By using constructors copy the values of one object to another object.

```
class Student
{       //instance variables
        int sid;
        String sname;
        int smarks;
        Student(int sid,String sname,int smarks)
        {       //conversion [converting localvariable values  to instance varaibles]
                this.sid=sid;       this.sname=sname;       this.smarks=smarks;
        }
        Student(Student s)        //constructor expected Student object
        {       this.sid=s.sid;     this.sname=s.sname;     this.smarks=s.smarks;
        }
        void disp()
        {       System.out.println("****student Details****");
                System.out.println("student name = "+sname);
                System.out.println("student id = "+sid);
                System.out.println("student mrks = "+smarks);
```

```
    }
    public static void main(String[] args)
    {       Student s = new Student(111,"ratan",100);
            Student s1 = new Student(s); //constructor is taking Student object
            s.disp();
            s1.disp();
    }
}
```

Difference between methods and constructors:-

Property	***methods***	***constructors***
1)Purpose	*methods are used to write logics but these logics will be executed when we call that method.*	*Constructor is used write logics of the project but the logics will be executed during Object creation.*
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	***M****ethod 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.*

Constructor chaining :-

- ✓ *One constructor is calling same class constructor is called constructor calling.*
- ✓ *We are achieving constructor calling by using this keyword.*
- ✓ *Inside constructor we are able to declare only one this keyword that must be first statement of the constructor.*

```
class Test
{       Test()
        {       this(10);
                System.out.println("0-arg cons");
        }
        Test(int i)
        {       this(10,20);
                System.out.println("1-arg cons");
        }
        Test(int i,int j)
        {       System.out.println("2-arg cons");
        }
        public static void main(String[] args)
        {       new Test();
        }
}
```

Instance Blocks:-

- ➢ *Instance blocks are executed during object creation just before constructor execution.*
- ➢ *Instance blocks execution depends on object creation it means if we are creating 10 objects 10 times instance blocks are executed.*

Example 1:-

```
class Test
{       {       System.out.println("instance block:logics");  }//instance block
        Test()
        {       System.out.println("constructor:logics");
        }
        public static void main(String[] args)
        {       Test t = new Test();
        }
```

```
}
```

Example 2:-

```
class Test
{       {       System.out.println("instance block-1:logics");        }
        Test()  {       System.out.println("0-arg constructor:logics");      }
        {       System.out.println("instance block-2:logics");        }
        Test(int a)
        {       System.out.println("1-arg constructor:logics");      }
        public static void main(String[] args)
        {       Test t1 = new Test();
                Test t2 = new Test();
                Test t3 = new Test(10);
        }
}
```

Example 3:-

```
class Test
{       {       System.out.println("instance block-1:logics");        }
        Test()
        {       this(10);
                System.out.println("0-arg constructor:logics");      }
        Test(int a)
        {       System.out.println("1-arg constructor:logics");
        }
        public static void main(String[] args)
        {       Test t1 = new Test();
        }
}
```

Example 1:-

```
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)
        {       Test t = new Test();
        }
}
```

D:\morn11>java Test
instance block

m1() method called by variable

example :-

```
class Test
{       int a=m1();     //instance varaibles
        int m1()
        {System.out.println("m1() method called by variable");
                return 100;
        }
        {       System.out.println("instance block"); }        //static blocks
        public static void main(String[] args)
        {       Test t = new Test();
        }
}
```

D:\morn11>java Test
m1() method called by variable
instance block

example :-

- *instance blocks are used to initialize instance variables during object creation but before constructor execution.*

```
class Emp
{       //instance blocks
        int eid;                //0
        String ename;           //null
        double esal;            //0.0
        //instance block used to initialize instance varaibles
        {       ename="ratan";
                eid=111;
                esal=20000;
        }
        void disp()
        {System.out.println("emp name="+ename);
        System.out.println("emp id="+eid);
        System.out.println("emp sal="+esal);
        }
        public static void main(String[] args)
        {       Emp e1 = new Emp();         //default constructor & instance block is executed
                e1.disp();
        }
};
```

static block:-

- *Static blocks are used to write functionality of project that functionality is executed during .class file loading time.*
- *In java .class file is loaded only one time hence static blocks are executed once per class.*

Example :-

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

Example :-

```
class Test
{       static{System.out.println("static block-1");}  //static block
        static{System.out.println("static block-2");}  //static block
        {System.out.println("instance block-1");}      //instance block
        {System.out.println("instance block-2");}      //instance block
        Test(){ System.out.println("0-arg constructor"); }     //0-arg constructor
        Test(int a){ System.out.println("1-arg constructor"); }//1-arg constructor
        public static void main(String[] args)
        {       Test t1 = new Test();   //instance block & constructor executed
                Test t2 = new Test(10);         //instance blocks & constructor executed
        }
}
```

```
D:\morn11>java Test
static block-1
static block-2
instance block-1
instance block-2
0-arg constructor
instance block-1
instance block-2
1-arg constructor
```

Example:-

```
class Test
{       //instance variables
        int a=10,b=20;
        //static variables
        static int c=30,d=40;
        //instance method
        int m1(int a,int b)//local variables
        {       System.out.println(a+"---"+b);
                return 10;
        }
```

```
        //static method
        static String m2(boolean b)       //local variables
        {       System.out.println(b);
                return "ratan";
        }
        Test(int a)                 //constructor with 1-arg
        {       System.out.println("1-arg constructor");
        }
        Test(int a,int b)           //constructor with 2-arg
        {       System.out.println("2-arg constructor");
        }
        {System.out.println("instance block-1");}          //instance block
        {System.out.println("instance block-2");}          //instance block
        static {System.out.println("static block-1");}      //static block
        static {System.out.println("static block-2");}      //static block
        public static void main(String[] args)
        {       //Test object created with 1-arg constructor
                Test t1 = new Test(10);           //1-arg constructor & instance blocks executed
                //Test object created with 2-arg constructor
                Test t2 = new Test(100,200);      //2-arg constructor & instance blocks executed
                //printing instance variables by using Object name
                System.out.println(t1.a);
                System.out.println(t1.b);
                //printing static variables by using class name
                System.out.println(Test.c);
                System.out.println(Test.d);
                //instnace method calling by using object name
                int x = t1.m1(1000,2000);
                System.out.println("m1() method return value:-"+x);          //printing return value
                //static  method calling by using class name
                 String y = Test.m2(true);
                 System.out.println("m2() method return value:-"+y);         //printing return value
        }
};
```

Practical example:-

```
        class A
        {       };
        class B
        {       };
        class Test
        {       2-instnce varaibles
                2-static variables
                2-instance methods
                                1-method --->2-arg(int,char) --->return-type-->String
                                2 method ---->1-arg(boolean) ---->Test
```

```
        2-static methods
                        1static method--->2-arg(Aobj,Bobj) ---->A
                        2 static mentod --->1-arg(double)--->int
        public static void main(String[] args)
        {       print instance var
                print static var
                call instance methods
                call static methods
        }
};
class A
{};
class B
{};
class Test
{       //instance vriables
        int a=10;
        int b=20;
        //static variables
        static int c=30;
        static int d=40;
        String m1(int a,int b)
        {       System.out.println("m1 method");
                return "ratan";   //returning String value
        }
        Test m2(int a)
        {       System.out.println("m2 method");
                return this; //returning current class object
        }
        static A m3(A a,B b) //method is taking objects as a input values
        {       System.out.println("m3 method");
                A a1 = new A();
                return a1;      //returning A class object
        }
        static int m4(int a)
        {       System.out.println("m4 method");
                return 100;  //returning integer value
        }
        public static void main(String[] args)
        {               Test t = new Test();
                //printing instance variables by using object name
                System.out.println(t.a);
                System.out.println(t.b);
```

```
                //printing static variables by using class name
                System.out.println(Test.c);
                System.out.println(Test.d);
                String str=t.m1(1,2);  //holding m1() method return value(String)
                System.out.println(str); //printing return value
                Test t1 = t.m2(3);       //holding m2() method return value(Test)
                A a = new A();  B b = new B();
                //calling m3() method by pssing two objects (A,B)
                A a1 = Test.m3(a,b);//holding m3() method return value(A)
                int x = Test.m4(4);//holding m4() method return value(int)
                System.out.println(x);//printing return value
        }
};
```