

19-12-2013

# core java

By

Mr. Ratan

**Download more materials**

DurgaSoftwareSolutions

*Java Home*

**----- VISIT-----**

**<http://ameerpetmaterials.blogspot.in/>**  
**<http://ameerpetmatbooks.blogspot.in/>**  
**<http://satyajohnny.blogspot.in/>**

**Index**

1) Introduction	1	-	70	pages
2) OOPS	71	-	105	pages
3) Interfaces	106	-	111	pages
4) Packages	112	-	121	pages
5) java.lang.String	122	-	141	pages
6) Wrapper classes	142	-	146	pages
7) Java.io	147	-	154	pages
8) Exception Handling	155	-	187	pages
9) MultiThreading	188	-	210	pages
10) Nested classes	211	-	218	pages
11) Enumaration & GC	219	-	224	pages
12) Collections	225	-	237	pages
13) Java.net	238	-	243	pages
14) Java.awt	244	-	274	pages
15) Swings	275	-	283	pages
16) JVM	284	-	286	pages

**JAVA introduction:-**

Author	:	<b>James Gosling</b>
Vendor	:	Sun Micro System
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 7 2011
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)

Ex:-

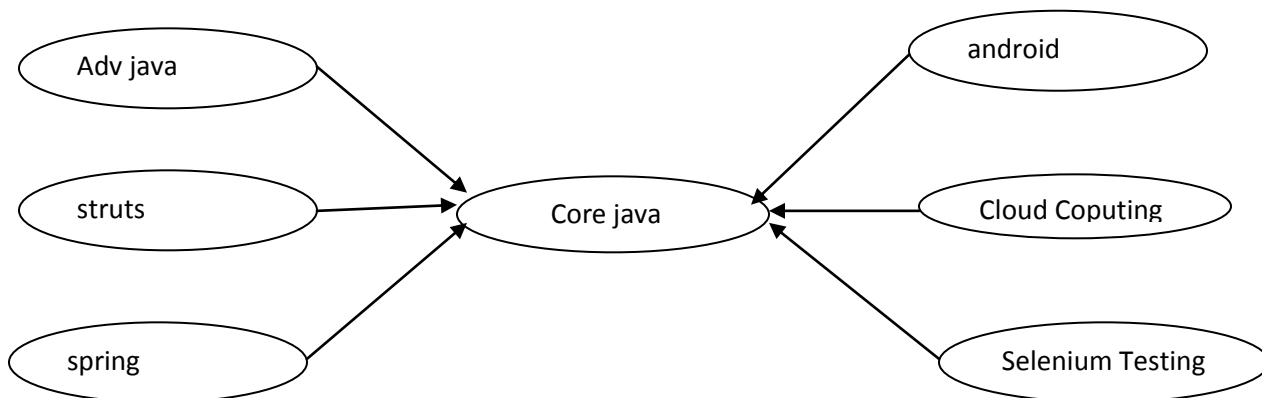
```
Class Test
{
    Public static void main (String [] args)
    {
        System.out.println ("welcome to java language");
    }
}
```

Compilation	:-	javac	FileName.java
Execution	:-	java	Class Name
Output	:-	welcome to java language	

**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 durgajobs.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:-****JAVA VERSIONS:-**

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

**C –language:-**

Author : Dennis Ritchie  
Implementation Languages : BCPL, ALGOL, FORTRAN.....  
Extensions : .c & .h  
Released in : 1972

Ex:-

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

Compilation :- alt+f5  
Execution :- ctrl+f9  
Output :- hello rattaiah

**C++ –language:-**

Author : Bjarne Stroustrup  
Implementation Languages : c  
Extensions : .cpp & .h  
Released in : 1983  
Initial Name : C with Classes  
Prasent Name : C++(c plus plus)

Ex:-

```
#include<iostream.h>
Void main()
{
Cout<<“hello durgasoft”;
}
```

Compilation :- alt+f5  
Execution :- ctrl+f9  
Output :- hello durgasoft

**Parts of the java language:-**

As per the Ameerpet standard the java language is divided into 2 types

- a. Core java
- b. Adv java

As per the **sun micro system** standard the java language is divided into three types.

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

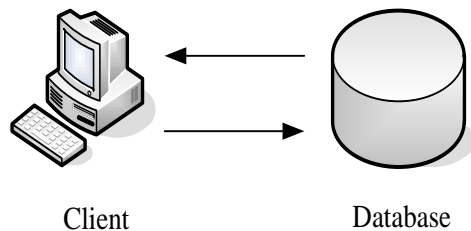
**J2SE:-**

By using j2se we are able to develop the standalone applications.

**Ex:-** notepad, WordPad, paint, Google Talk.....etc

**Standalone applications:-**

- 1) Standalone applications are the java applications which don't need the client server architecture.
- 2) The standalone applications applicable for the only one desktop hence it is called desktop applications or window based applications.
- 3) For the standalone applications doesn't need internet connections.
- 4) It is a local application it doesn't need any other external application support.
- 5) This type of the applications we can launch by using the command line or by using the executable jar.

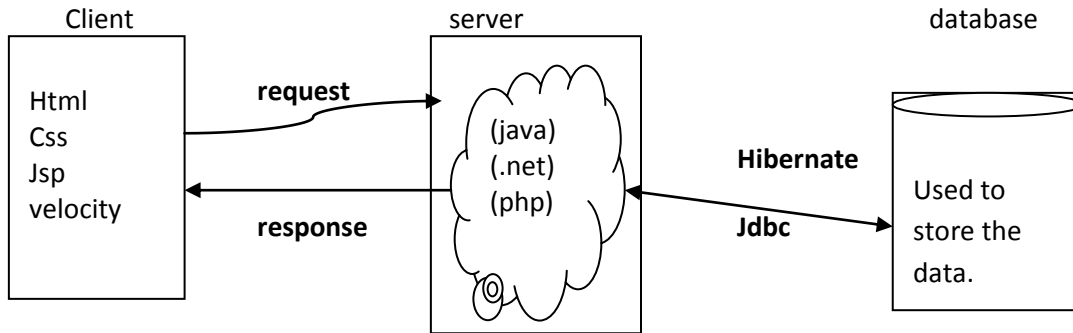
**J2EE:-**

By using j2ee we are able to develop the web based applications.

**Ex:-** Gmail, yahoo mail, bank, reservation.....etc

**Web-applications:-**

- 1) Web applications are the java applications which needs client and server concept.
- 2) Web applications must need the internet connections to access the application.
- 3) The application which is present in the internet is called the web application.
- 4) Web application can be launched by using HTTP driven. HTTP request sent to the Servlet present in the server side.

**Web-application architecture:-****Client:-**

The person who is sending the request is called client. All web browsers come under the clients.

Ex:- InternetExploral, MozillaFrefox, opera.....etc

**Server:-**

The server contains the applications. The main purpose of the server is

- It will contain the application.
- Take the request from the client.
- Based on the taken request it will identify the project resource and execute that project resource.
- By executing the project some response will be generated that response is dispatched to the client browser.

Ex:- Tomcat,GlassFish,WebLogic,JBOSS,WebSphere.....etc

**DataBase:-**

DataBase is used to store the details like client details, application details, registration details.....etc.

Ex:- Oracle,MySQL.....etc

**J2ME:-**

By using j2me we are able to develop the applications that applications only run on mobile devices.

**As a software engineer we must know these technologies:-****HTML(Hyper Text Markup Language):-**

It is used for web pages designing. Introduced by World Wide Web Consortium and developed by Tim Berners-Lee



**JavaScript(JS):-**

It is designed by Brendan Eich at NetScape Communications Corporation in 1995.it is used to validate the forms.

**Java:-**

Developed by James Gosling at **Sun Micro Systems** in 1996 used to develop particular for client-server web applications. Now it is merged into Oracle Corporation.

**.net:-**

Developed by **Microsoft** peoples in 2002 and it is used to develop the we based applications.

**PHP(Hyper Text Preprocessor):-**

It is a server side programming language used to develop the server side applications(web applications)and developed by **Rasmus Lerdorf** in 1995.

**Oracle:-**

It is a data base used to store the data permanently. Developed by oracle people in 1977.





**Android:-**

It is a Linux based mobile phone operating system used to develop the mobile applications that runs only on mobile devices.

**Server side technologies:-(servlets,structs,spring,php,.net)**

By using server side technologies we are able to prepare the applications these application only run on server side.

**Servlets vs structs:-**

Servlets is a technology used to develop the server side application and structs is a frame work it developed on the basis of the servelts.

**Jdbc vs hibernate:-**

Jdbc and hibernate is used to provide the connection between java application and data base. Jdbc is a technology and hibernate is a framework it is developed on the basis of the jdbc.

**Frame work:-**

Frame is a semi implemented component working with the frame works is very seay.

**Technology:-**

1. J2SE
2. J2EE
3. J2ME

**Frame works:-**

1. Struts
2. Spring
3. Hibernate

**Servers:-**

1. Tomcat
2. Glassfish
3. Jboss
4. Weblogic
5. Websphere.....

**IDE(integrated development environment):-**

1. Eclips
2. Myeclips
3. Netbeans
4. Jdeveloper.....

**Database :-**

Oracle Mysql.....

**Difference between**

<b>C-lang</b>	<b>Cpp-lang</b>	<b>Java -lang</b>
1) The program execution starts from main method and main method is called by Operating system.	The program execution starts from main method called by operating system.	The program execution starts from main method called by JVM(java virtual machine).
2) In the c-language the predefined support is maintained in the form of header files Ex:- stdio.h,conio.h	In the cpp language the predefined is maintained in the form of header files. Ex:- iostream.h	In the java language the predefined is maintained in the form of packages. Ex:- java.lang java.io java.net java.awt
3) the header files contains predefined functions. Ex:- printf,scanf.....	The header files contains predefined functions. Ex:- cout,cin....	The packages contains predefined classes. Ex:- String,System
4) to make available predefined support into our program we have to use #include statement. Ex:- #include<stdio.h>	To make available predefined support into our program we have to use #include statement. Ex:- #include<iostream>	To make available predefined support into our program we have to use import statement. Ex:- import java.lang.*;
5) memory allocation: malloc Memory deallocation: free	Allocation :constructor Deallocation:destructors	Allocation :constructor Deallocation:Garbage collector(part of the jvm)
6) size of the data types are varied from operating system to the operating system. 16-bit int -2bytes char- 1byte 32-bit int -4bytes char – 2 bytes	Irrespective of any os Int -4 Char-2	Irrespective of any os Int -4 Char-2
7) to print some statements into the console we have to use "printf" function.	To print the statements we have to use "cout"	To print the statements we have to use "System.out.println"
8) C is dynamic language the memory is allocated at compilation time.	Cpp is static language the memory is allocated at compilation time.	Java is dynamic language the memory is allocated at runtime time

**c-language:-**

c-language  
↓  
headerfiles  
↓  
functions

Dennis Ritchie  
↓  
stdio.h, conio.h  
↓  
printf, scanf.....

**void main()** (program starting point)  
  
printf("ratan");

**cpp-language:-**

cpp-language  
↓  
headerfiles  
↓  
functions

Bjarne Stroustrup  
↓  
iostream.h  
↓  
cout, cin....

**void main()** (program starting point)  
  
cout<<"ratan";

**java-language:-**

java-language  
↓  
packages  
↓  
classes & interfaces  
↓  
methods & variables

james gosling  
↓  
java.lang  
↓  
System, String.....  
↓  
length(), charAt(), concat()...

**public static void main(String[] args)**  
(program starting point)  
System.out.println("ratan");

**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

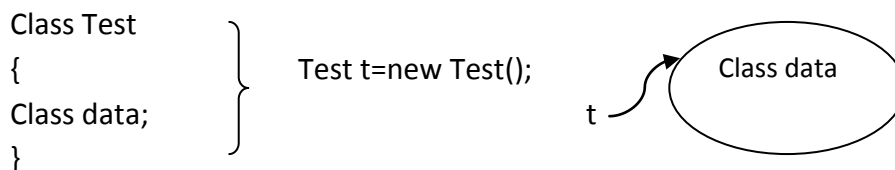
### 1. Simple:-

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:-

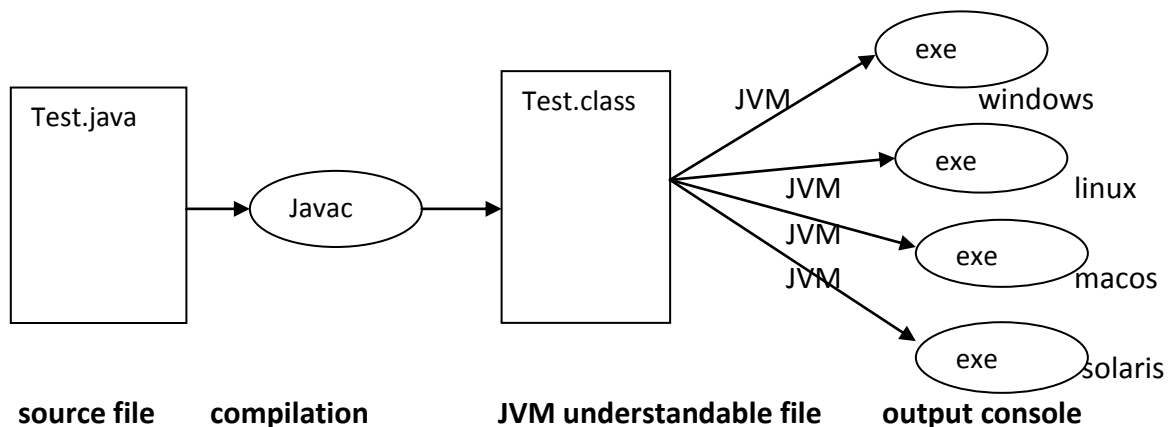
Java is object oriented technology because to represent total data in the form of object. By using object reference we are calling all the methods, variables which is present in that class.



The total java language is dependent on object only hence we can say java is a object oriented technology.

### 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 allows its applications compilation one operating system that compiled (.class) files can be executed in any operating system.



**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

- a. JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.
- b. 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.

Web security for web applications we are having JAAS(Java Authentication and Authorization Services) for distributed applications.

**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 Compleitive 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 :-

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 following 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 following 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 following 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 an Editor.

Step2:- Write a Program & save the program.

Step3:- Compile the program.

Step4:- Execute the program.

### **Step1:- Select an Editor**

Editor is a tool or software it will provide very good environment to type the java application

Ex:- Ameerpet Editors----->Notepad, Notepad++ ,Edit Plus---etc

Hi-Tech City Editors---> Eclipse, Myeclips, Net Beans----etc

### **Step 2:- Write the program and save the program**

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

Java is a case Sensitive Language while writing the program we have to take care about the case (Alphabet symbols)

After writing the program we have to save the program while saving the program we have to consider fallowing steps

Check whether the source file contain the public class or not

1. if the source file contain public class the name of the public class and the name of the Source file must be same(publicClassName.java). Otherwise we are getting compilation error.
2. if the source file does not contain any public class at that situation we can save the source file with (anyName.java)

### Step3:- Compile the program

To single source file

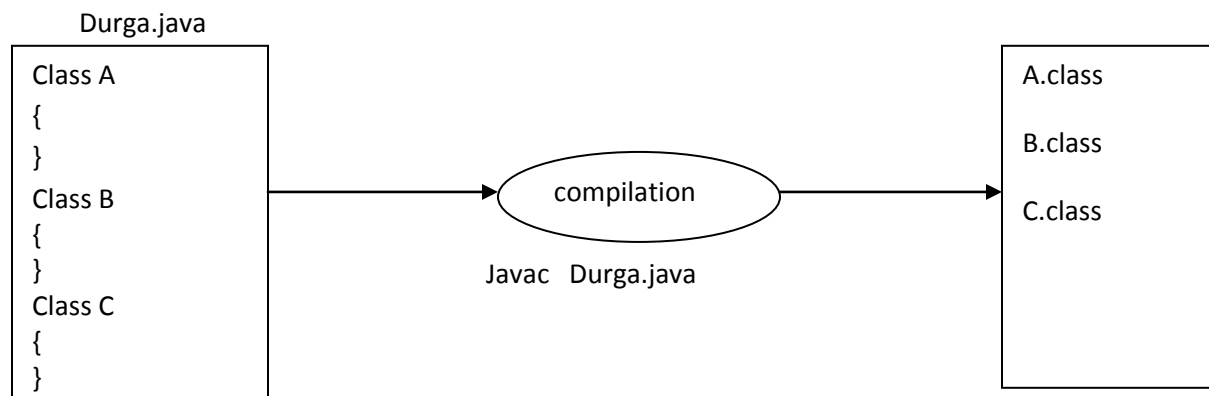
Ex:- `Javac filename.java`  
`Javac Test.java`

The java compiler goes to Test.java file and it will search for syntactical error if the syntactical errors are presented the java compiler raise compilation error if there is no syntactical errors are presented at that situation the java compiler converts the .java files into the .class file.

Note:- the .class file generation totally based on the number of classes present in the source it is not depending the number of classes present in the source file it is not depending on the name of the file.

To compile multiple source files at a time

`Javac *.java`



### Step4:- Execute the program

`Java class-name`

`Java Test`

Whenever we are typing the Test class in the command prompt first it will search for the Test class if the Test class is available then it will search for the Main method if the main method is there the program execution starts from main method onwards.

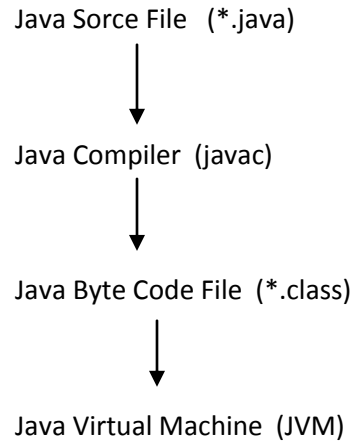
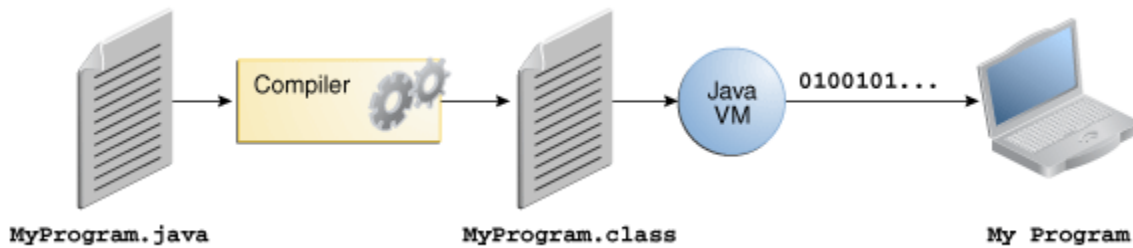
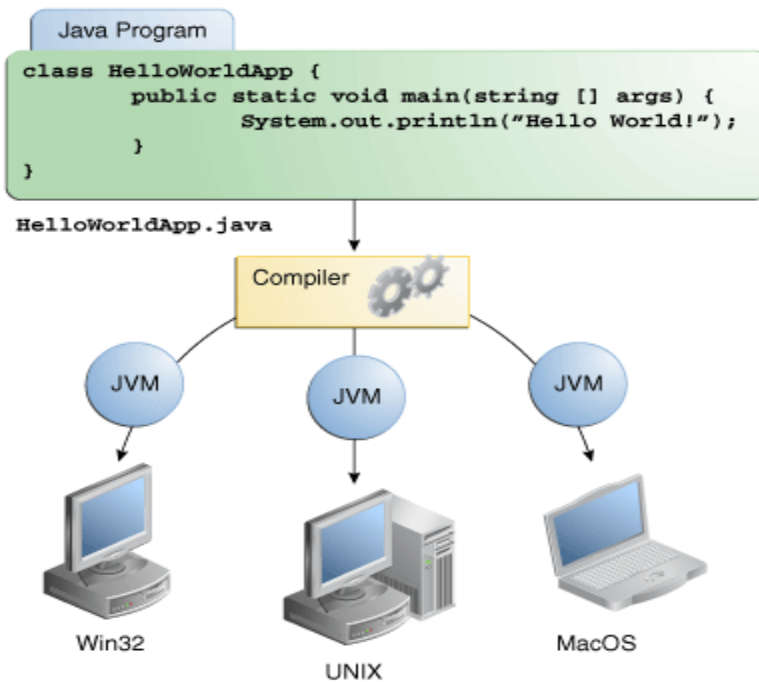
The JVM will search for the .class file if the .class file is not available then JVM will raise an exception

**Exception inn thread "main" java.lang.NoClassDefFoundError**

The JVM will search for the .class file if the .class is available the JVM will search for the main method if the main method is not available then the JVM will raise an Eception.

**Exception in thread "main" java.lang.NoSuchMethodError: Main**



**JAVA ENVIRONMENT:-****Environment of the java programming development:-****First program development :-**

**Class Contains Five elements:-**

Class Test

- {
- 1. variables
- 2. methods
- 3. constructors
- 4. instance blocks
- 5. static blocks
- }

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

Ex:-Class Test

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

**Tokens are**----->class,test,{,},[,-----etc

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

Print is used to print the statement into the console and the control is available in the same line.

Ex:-     System.out.print("durgaSoftware");  
          System.out.print("core java");

Output:-durgasoftwarecorejava

**Println():-**

In the println statement Print is used to print the statement into the console and ln represent go to the new line now the control is available in the next line.

Ex:-     System.out.println("durgasoftware");  
          System.out.println("core java");

Output: -         durgasoftware  
                  Core java

**Identifiers:-**

any name in the java program like variable name, class name, method name, interface name is called identifier.

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

Test----->identifier  
add----->identifier  
a----->identifiers  
b----->identifiers

**Rules to declare identifiers:-**

- the java identifiers should not start with numbers, it may start with alphabet symbol and underscore symbol and dollar symbol.
  - Int abc=10;---->valied
  - Int 2abc=20;---->not valied
  - Int \_abc=30;---->valied
  - Int \$abc=40;---->valied
  - Int @abc=50;---->not valied

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

- The identifier should not duplicated.

```
class Test
{
    void add()
    {
        int a=10;
        int a=20;
    }
};
```

the identifier should not be duplicated.

- 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 recamanded to use.

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

**JAVA NAMING CONVENTIONS:-**

Java is a case sensitive language so the way of writing code is important.

1. All Java classes, Abstract classes and Interface names should start with uppercase letter ,if any class contain more than one word every innerword also start with capital letters.

Ex: String

StringBuffer

FileInputStream

2. All java methods should start with lower case letters and if the method contains more than one word every innerword should start with capital letters.

Ex :-        post()  
              toString()  
              toUpperCase()

3. All java variables should start with lowercase letter and inner words start with uppercase letter.

Ex:-        pageContent  
              bodyContent

4. All java constant variables should be in uppercase letter.

Ex: MIN\_PRIORITY  
     MAX\_PRIORITY  
     NORM\_PRIORITY

5. All java packages should start with lower case letters only.

Ex:        java.awt  
             Java.io

**NOTE:-**

The coding standards are applicable for predefined library not for user defined library .But it is recommended to follow the coding standards for user defined library also.

**JAVA COMMENTS :-**

To provide the description about the program we have to use java comments.

There are 3 types of comments present in the java language.

**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:-

This comment is used to provide description about our program in more than one page. In general we are using document comment to prepare API kind of documents but it is not sujastable.

Syntax: -       /\*.....line-1  
                  \*.....line-2  
                  \*.....line-3  
                  \*/

Ex:-/\*project name:-green project

team size:-       6

team lead:- ratan

\*/

class Test

{

    //main method

    public static void main(String[] args)

    {

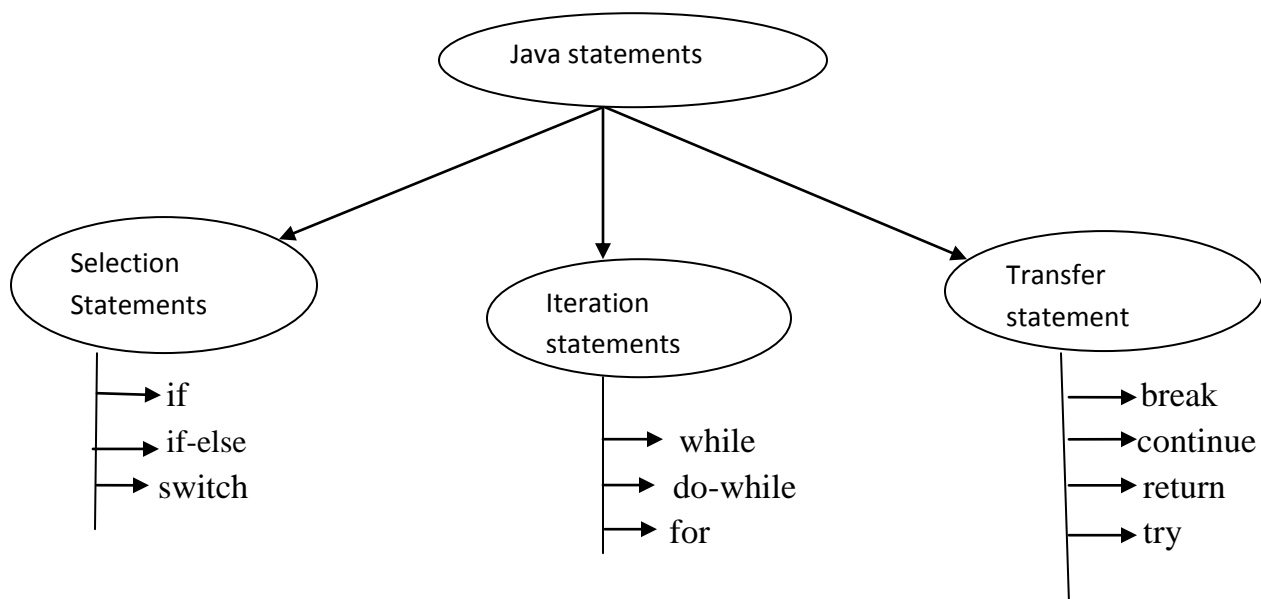
        //printing statement

        System.out.println("ratan");

    }

};

### java Statements:-



**If syntax:-**

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

The curly braces are optional whenever we are taking single statements.

The curly braces are mandatory whenever we are taking multiple statements.

**Ex 1:-**

```
class Test
{
    public static void main(String[] args)
    {
        int a=10;
        if (a>5)
        {
            System.out.println("if body / true body");
        }
    }
}
```

**Ex 2:-**

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

**Ex 3:-**

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

**Ex 4:-good in c-language but in java we are getting compilation error**

class Test

```
{
    public static void main(String[] args)
    {
        if (0)
        {
            System.out.println("if body / true body");
        }
        System.out.println("hi rattaiah");
    }
}
```

**If-else syntax:-**

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

The curly braces are optional whenever we are taking single statements.

The curly braces are mandatory whenever we are taking multiple statements.

**Ex:-**

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 ");
        }
        System.out.println("hi rattaiah");
    }
}
```

**Switch statement:-**

- 1) Switch statement is used to take multiple selections.
- 2) Curly braces are mandatory if we are not taking we are getting compilation error.
- 3) Inside the switch It is possible to declare any number of cases but is possible to declare only one default.
- 4) Switch is taking the argument the allowed arguments are
  - a. Byte
  - b. Short
  - c. Int
  - d. Char
  - e. String(allowed in 1.7 version)
- 5) 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.
- 6) If the case is matched particular case will be executed if there is no case is matched default case is executed.

**Syntax:-**

```
switch(argument)
{
    Case label1      :    sop(" ");
                      break;

    Case label2      :    sop(" ");
                      break;

    |
    |

    Default          :    sop(" ");
                      break;
}
```



**Eg:**

```
class Test
{
    public static void main(String[] args)
    {
        int a=10;
        switch (a)
        {
            case 10:System.out.println("10");
                break;
            case 20:System.out.println("20");
                break;
            case 30:System.out.println("30");
                break;
            case 40:System.out.println("40");
                break;
            default:System.out.println("default");
                break;
        }
    }
};
```

Output: 10

**Ex:-the case label must be unique. Duplicate case labels are not allowed in switch statement.**

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

**Eg:- the case labels must be constant expression if we are providing variables as a case labels the compiler will raise compilation error.**

class Test

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

**Ex:- for the case label it is possible to provide constant expressions.**

class Test

```
{
    public static void main(String[] args)
    {
        int a=100;
        switch (a)
        {
            case 10+20+70: System.out.println("10");
                break;
            case 100+200: System.out.println("20");
                break;
            default : System.out.println("default");
                break;
        }
    }
};
```

**Ex:- inside the switch the default is optional part.**

class Test

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

```
        case 10: System.out.println("10");
                break;
        }
    }
};
```

**Ex 4:- Inside the switch case is optional part.**

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

**Ex 5:- inside the switch both case and default is optional.**

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

**Ex :-inside the switch independent statements are not allowed. If we are declaring the statement 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:- Inside the switch statement break 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 break is available up to the end of the switch is executed. This situation is called as fall through 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 30:System.out.println("30");
            case 40:System.out.println("40");
                break;
            default: System.out.println("default");
                break;
        }
    }
};
```

**Ex :-inside the switch statement we are able to take the default anywhere else.(starting ,middle ,ending)**

```
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 7:- from the 1.7 version onwards the string also allowed argument for the switch statement.**

```
class Durga
{
    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;
    }
}
```

Output:- hai

**Ex :-**

class Test

```
{
    public static void main(String[] args)
    {
        char ch='a';
        switch (ch)
        {
            case 'a' : System.out.println("Hai");
                      break;
            case 'b' : System.out.println("Hello");
                      break;
            case 'c' : System.out.println("how");
                      break;
            default  : System.out.println("what");
                      break;
        }
    }
}
```

**Ex:- swirch argument data type and case label values must be same data type otherwise the compiler will raise compilation error.**

class Test

```
{
    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 'c' :System.out.println("how");
                      break;
            default   : System.out.println("what");
        }
    }
}
```

```
        break;
    }
}
```

**Ex:-inside switch the case label must be within the range of provided argument data type.**

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

**Ex :- this program only executed only in 1.7 version.**

```
class Test
{
    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;
            default    : System.out.println("what");
                        break;
        }
    }
}
```

**Iteration Statements:-**

If we want to execute group of statements repeatedly or more number of times then we should go for iteration statements.

Three types of iteration statements present in the java language

- 1) for
- 2) while
- 3) do-while

**for syntax:-**

```

    for (part 1;part 2 ;part 3 )
    {
        Body;
    }
Ex:-  for (initialization ;condition ;increment/decrement )
    {
        Body;
    }

```

- 1) The for loop contains three parts initialization, condition, increment/decrement part.
- 2) Each and every part is separated by semicolon and it is mandatory.

The curly braces are optional whenever we are taking single statement.

The curly braces are mandatory whenever we are taking more than one statements.

**Flow of execution in for loop:-**

```

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

```

Step1:- initialization is down

Step 2:- condition will be checked.

Step 3:- if the condition is true body will be executed. If the condition is false loop stopped.

Step 4:- after body increment/decrement part will be executed.

Step 5:- condition will be checked.

Step 6:- if the condition is true body is executed.

Step 7:- after body increment/decrement part will be executed.

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

**Initialization part:-**

- 1) Inside the for loop initialization part is optional.
- 2) Instead of initialization it is possible to take any number of System.out.println("ratna"); also and each and every statement is separated by coma(,).
- 3) Initialization part it is possible to take the single initialization it is not possible to take the more than one initialization.

**Ex 1:-normal for loop**

```

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

```

**Ex 2:-initialization part is optional**

```

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

```

**Ex 3:-instead of initialization possible to take System.out.pritnln("ratna");**

```

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

```

**Ex 5:- 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 6:-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:-**

- 1) Inside the for loop conditional part is optional.
- 2) If we are not taking any condition then the compiler places the true value.
- 3) The condition is always must return Boolean(true, false) values only.

**Ex 1:-if we are not providing condition it is always true representing infinite loop.**

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

**Ex 2:-**

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

**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");
    }
}
```

**Increment/decrement:-**

- 1) Inside the for loop increment/decrement session is optional.
- 2) Instead of increment/decrement it is possible to take the any number of  
System.out.println("ranta"); also and each and every statement is separated by coma(,).

**Ex1:-inc/dec part is optional**

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

**Ex 2:- instead if inc/dec it is possible to take the any number of System.out.println("ranta");**

```
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 :**

each and every part inside the for loop is optional.

for(;;)----->represent infinite loop because the condition is always true.

**While:-**

If we want to execute group of statements repeatedly or more number of times then we should go for while loop.

**Syntax:-**

```
while (condition)
{
    body;
}
```

**Ex 1 :-**

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

```
        while (i<10)
        {
            System.out.println("rattaiah");
            i++;
        }
    }
}
```

**Ex 2:-represent infinite loop**

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

**Ex 3:- compilation error unreachable statement**

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

**Do-While:-**

- 1) If we want to execute the loop body at least one time then 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);
```

**Ex :-**

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

**Ex :- unreachable statement**

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

**Ex :-**

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

Output:-        Rattaiah        durgasoft

**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:-** we are able to use the break statement only two places if we are using any other place the compiler will raise compilation error.

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

Ex :-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);
        }
    }
}
```

**Continue:-(skip the current iteration 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);
        }
    }
}
```

**Key words:-**

Keywords for Modifier	Keywords for Flow Control	Keywords for Exception Handling	Keywords for Class	Keywords for Object	Keywords for primitive data Types
Public private protected static abstract final native volatile synchronized transient strictfp (11)	If else switch break case default for do while continue (10)	try catch finally throw throws  (5)	Import class interface extends package implements  (6)	New instanceof super this  (4)	Byte short int long float double char boolean (8)
		Unused keywords	Metho level	Reserved literals	1.5 version keywords
		goto const (2)	Void return (2)	true false null (3)	assert enum (2)

**Data Types:-**

- 1) Data types are used to represent the type of the variable and type of the expression.
- 2) Data types are used to specify the how much memory is allocated for variables.

Data type	Size	Range	Default values
Byte	1	-128 to 127	0
short	2	-32768 to 32767	0
int	4	-2147483648 to 2147483647	0
long	8	-2 <sup>31</sup> to 2 <sup>31</sup> -1	0
float	4	-3.4e38 to 3.4e308	0.0
double	8	-1.7e308 to 1.7e308	0.0
char	2	0 to 65535	Single space character
Boolean	NA	Not Applicable	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

**Literals:-**

Literal is a constant value assigned to the variables.

'a'-----→char literal  
 10-----→integral literal  
 False-----→boolean literal  
 10.2345-----→double literal

**Floating point literal & double literal:-**

By default the decimal point values represent double so if we want to assign the floating point values to the variables we must attach the suffix **F or f** to the number. If we are not providing the number we will get compilation error possible loss of precision.

double d=100.9898-----good

float f=10.897-----compilation error(possible loss of precision)

float f=12345.67890f;-----good suffix with f

it is optionally to attach the d or D with double double value

double d=1234.5678;-----good

double d=1234567.7654321d;-----good

**Boolean literal:-**

The Boolean values are true or false.

boolean b=true;-----→good

Boolean b=0;-----→it's working in c-lang but not in java

**Char literal:-**

A character literal is represented by a character in single quote.

Char ch='a';

Char ch1='#';

Char ch='1';

```
class Test
{
    public static void main(String[] args)
    {
        char ch1="";
        System.out.println((int)ch1);
        char ch2='\n';
        System.out.println((int)ch2);
    }
}
```

**String literal:-**

A string literal represented by group of character in double quotes.

String str="ratan";

```
class Test
{
    public static void main(String[] args)
    {
        String str1="ratan";
        System.out.println(str1+"soft");
        System.out.println("durga"+"javahome");
    }
}
```

Ex:-

byte+byte=int	byte+short=int	short+int=int	byte+int=int	byte+long=long
short+float=float	long+float=float	float+double=double	String+int=String	



**There are two types of areas:-**

- 1) Instance area
- 2) Static area

**Instance area:-**

```
void m1()
```

```
{
```



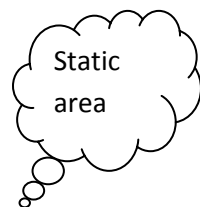
```
}
```

Instance method

**Static area:-**

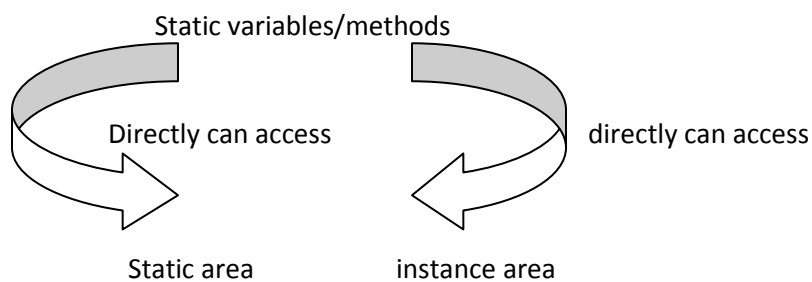
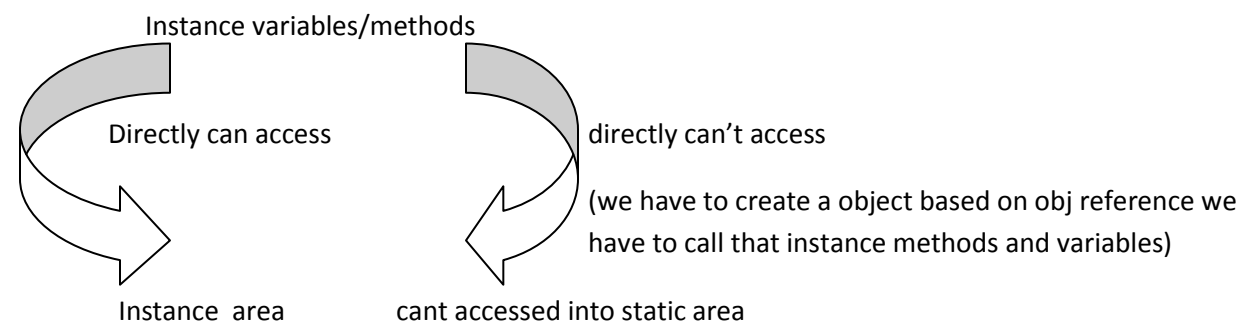
```
Static Void m1()
```

```
{
```



```
}
```

Static method

**Static variables & methods accessing:-****Instance variables/methods declaration:-**

**Two figures are heart of the java language**

**Types of variables:-**

- Variables are used to store the values. By storing that values we are achieving the functionality of the project.
- While declaring variable we must specify the type of the variable by using data type's concept.

In the java language we are having three types of variables

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

**Local variables:-**

- a. The variables which are declare inside a method & inside a block & inside a constructor is called local variables
- b. The scope of local variables are inside a method or inside a constructor or inside a block.
- c. We are able to use the local variable only inside the method or inside the constructor or inside the block only.

Ex:-

```
class Test
```

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

**Instance variables:-**

1. The variables which are declare inside a class and outside of the methods is called instance variables.
2. We are able to access instance variables only inside the class any number of methods.

```
class Test
```

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

**3. Static variables:-**

- The instance variables which are declared as a static modifier such type of variables are called static variables.
- We are able to access static variables within the class any number of methods.

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

**Calling of static variables:-**

- a. Directly possible.
- b. By using class name possible.
- c. By using reference variable possible.

```
class Test
{
    static int x=100;
    public static void main(String[] args)
    {
        //1-way(directly possible)
        System.out.println(a);
        //2-way(By using class name)
        System.out.println(Test.a);
        //3-way(By using reference variable)
        Test t=new Test();
        System.out.println(t.a);
    }
};
```

**Instance vs Static variables:-**

1. Instance variable for the each and every object one separate copy is maintained.
2. Static variable for all objects same copy is maintained. One Object change the value another object is affected.

```
class Test
{
    int a=10;
    static int b=20;
    public static void main(String... ratan)
    {
```

```
        Test t1=new Test();
        System.out.println(t1.a);//10
        System.out.println(t1.b);//20
        t1.a=444;
        t1.b=555;
        Test t2=new Test();
        System.out.println(t2.a);//10
        System.out.println(t2.b);//555
        t2.b=111;
        System.out.println(t2.b);//111
        Test t3=new Test();
        System.out.println(t3.a);//10
        System.out.println(t3.b);//111
        Test t4=new Test();
        System.out.println(t4.a);//10
        System.out.println(t4.b);//111
    }
};
```

**Variables default values:-**

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

```
class Test
{
    int a=10;
    int b;
    public static void main(String[] args)
    {
        Test t=new Test();
        System.out.println(t.a);//10
        System.out.println(t.b);//0
    }
}
```

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

```
class Test
{
    static double d=10.9;
    static boolean b;
    public static void main(String[] args)
    {
        System.out.println(d);//10.9
        System.out.println(b);//false
    }
}
```

**Case 3:- for the local variables the JVM won't provide the default values before using those variables we have to provide the default values.**

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

Compilation error:- Variables b might not have been initialized  
System.out.println(b);

**Practice example:-**

```
class Test
{
    //2-instance variables
    int a=10;
    boolean b;
    //2-static variables
    static int c=20;
    static double d;
    //1-instance methods
    void m1()
    {
        System.out.println(a);//10
        System.out.println(b);//false
        System.out.println(c);//20
        System.out.println(d);//0.0
    }
    //1-static method
    static void m2()
    {
        Test t=new Test();
        System.out.println(t.a);//10
        System.out.println(t.b);//false
        System.out.println(c);//20
        System.out.println(d);//0.0
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
        m2();
    }
}
```

## Class vs Object:-

- Class is a group of objects that have common property.
- Java is classes based language we are able to design the program by using classes and objects.
- Object is a realworld entity. Object orientation is methodology to design a program by using classes and objects.
- Object is physical entity where as class is a logical entity.
- A class is a template or blue print from which type of objects are created.
- Object is nothing but instance of a class.

### Every objects contains 3 characteristics

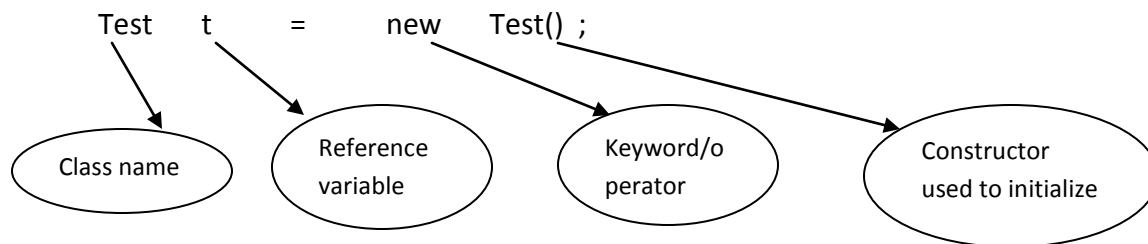
1. State(represent data of an object)
2. Behavior(represent behavior of an object)
3. Identity(used to identify the objects uniquely).

PEN(object):-

State:- name raynolds,color red etc.....

Behaviour:- used to write

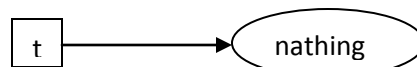
**Declaration** :- it is representing the variable associated with object.  
**Instantiation** :- The new keyword is a Java operator that creates the object.  
**Initialization** :- The new operator is followed by a call to a constructor, which initializes the the new object.



**Test t;**

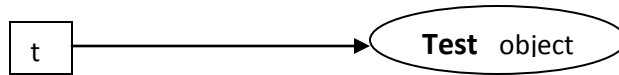
- a. It notifies the compiler refers the Test data with t reference variable.
- b. Declaring reference variable doesn't mean creating object we must use new operator to create object.

t is reference variable pointing to nothing.



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

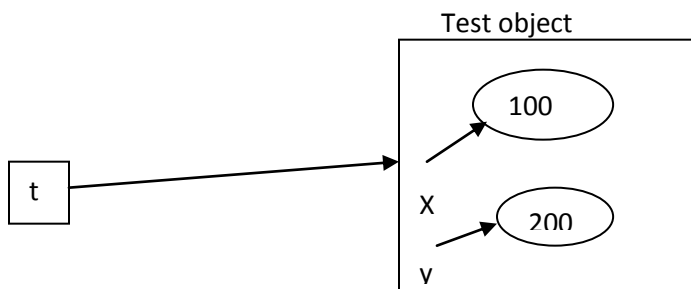
- a. The new operator instantiating class by allocating memory for new object and the reference variable pointing to that object.



**Test t=new Test(100,200);**

Constructors are executed as part of the object creation. If we want perform any type of initializations at the time of object creation use constructors.

```
class Test
{
    int x;
    int j;
    //constructor
    Test(int a,int b)
    {
        x=a;
        y=b;
    }
    public static void main(String[] args)
    {
        Test t=new Test(100,200);
    }
};
```

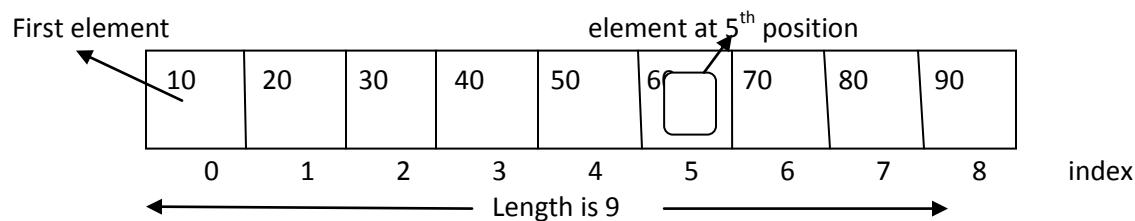


## Arrays:-

- 1) Array is a final class inheritance is not possible.
- 2) Arrays are used to store the multiple numbers of elements of single type.
- 3) The length of the array is established at the time of array creation. After creation the length is fixed.
- 4) The items presented in the array are classed elements. Those elements can be accessed by index values. The index is begins from (0).

### Advantages of array:-

- 1) Length of the code will be decreased
- 2) We can access the element present in the any location.
- 3) Readability of the code will be increased.



### Root structure:-

java.lang.Object

|

+--java.lang.reflect.Array

public final class **Array** extends Object

### single dimensional array declaration:-

```
int[] a;
int []a;
int a[];
```

### declaration & instantiation & initialization :-

```
approach 1:- int a[]={10,20,30,40};
approach 2:- int[] a=new int[100];
               a[0]=10;
               a[1]=20;
               a[2]=30;
               a[4]=40;
```



**Ex:-printing the array elements**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a={10,20,30,40};
        System.out.println(a[0]);
        System.out.println(a[1]);
        System.out.println(a[2]);
        System.out.println(a[3]);
    }
}
```

**Ex:-printing the array elements by using for loop**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a={10,20,30,40};
        for (int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

**Ex:-**

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        int[] a=new int[5];
        Scanner s=new Scanner(System.in);
        System.out.println("enter values");
        for (int i=0;i<a.length;i++)
        {
            System.out.println("enter "+i+" value");
            a[i]=s.nextInt();
        }
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
}
```

**Ex:-printing the array elements by using for-each loop(1.5 version)**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a={10,20,30,40};
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
}
```

**Ex:-**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a=new int[100];
        System.out.println(a.length);
        System.out.println(a[99]);
        boolean[] b=new boolean[100];
        System.out.println(b[99]);
        char[] ch=new char[50];
        System.out.println(ch[44]);
        byte[] bb=new byte[100];
        System.out.println(bb[100]); //ArrayIndexOutOfBoundsException
    }
}
```

**Ex:-**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a=new int[4]; // allocates memory for 4 elements
        a[0]=10;
        a[1]=100;
        a[2]=1000;
        a[3]=10000;
        System.out.println(a.length);
        for (int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

**To get the class name of the array:-**

**getClass() method is used to get the class.**

**getName() method is used to print the name of the class.**

```
class Test
{
    public static void main(String[] args)
    {
        int[] a={10,20,30};
        System.out.println(a.getClass().getName());
    }
}
```

**declaration of two dimensional array:-**

```
int[][] a;  
int [][]a;  
int a[][];  
int []a[];
```

Ex:-

class Test

{

public static void main(String[] args)

{

int[][] a={{10,20,30},{40,50,60}};

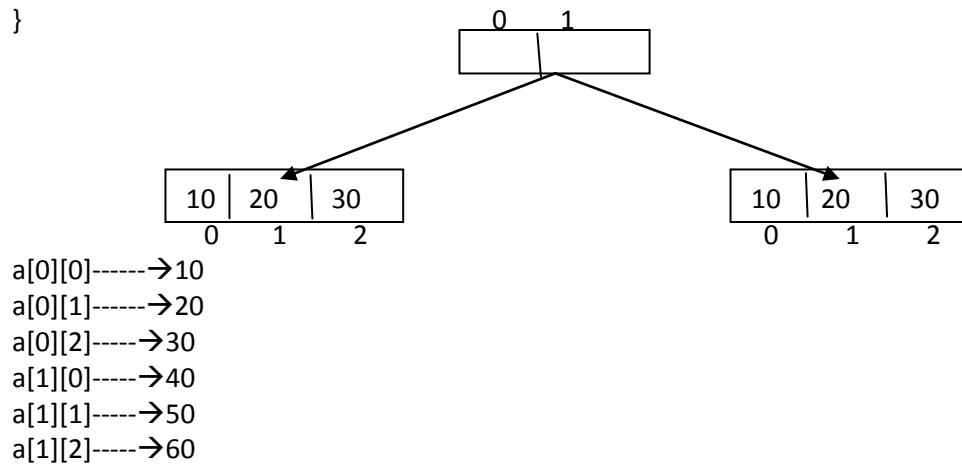
System.out.println(a[0][0]);//10

System.out.println(a[1][0]);//40

System.out.println(a[1][1]);//50

}

}



**Methods (behaviors):-**

- 1) Methods are used to provide the business logic of the project.
- 2) The methods like a functions in C-language called functions, in java language is called methods.
- 3) Inside the class it is possible to declare any number of methods based on the developer requirement.
- 4) As a software developer while writing method we have to follow the coding standards like the method name starts with lower case letters if the method contains two words every inner word also starts uppercase letter.
- 5) It will improve the reusability of the code. By using methods we can optimize the code.

**Syntax:-**

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

Ex:-

```
Public void m1()
Public void m2(int a,int b)
```

**Method Signature:-**

The name of the method and parameter list is called Method Signature. Return type and modifiers list not part of a method signature.

Ex:-     m1(int a,int b)-----→Method Signature  
           m2();-----→Method signature

**Method declaration:-**

Ex:-

```
void m1() -----→ declaration of the method.
{
    statement-1
    statement-2
    statement-3
    statement-4 } implementation/body
}
```

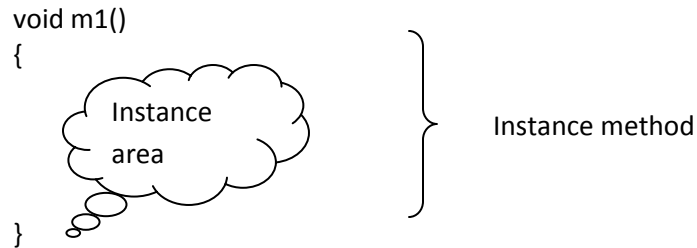
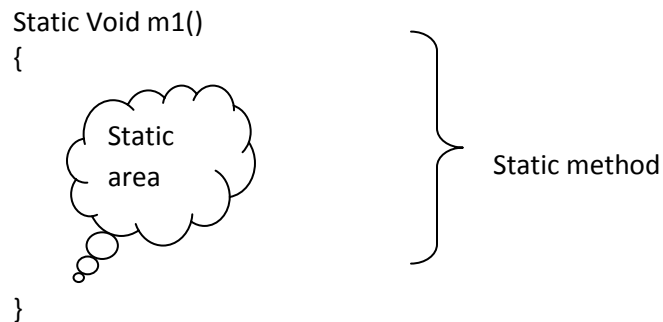
Ex:-

```
Void deposit()
{
    System.out.println("money deposit logic");
    System.out.println("congrats your money is deposited");
}
```

defining a method  
 functionality of a method

There are two types of methods:-

**Instance method**  
**Static method**

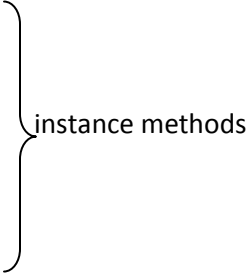
**Instance method:-****Static Method:-****Ex :- instance methods without arguments.**

```
class Test
{
    void durga()
    {
        System.out.println("durgasoftware solutions");
    }
    void soft()
    {
        System.out.println("software solutions");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.durga();
        t.soft();
    }
}
```

**Ex:-instance methods with parameters.**

class Test

```
{
    void m1(int i,char ch)
    {
        System.out.println(i+"-----"+ch);
    }
    void m2(float f,String str)
    {
        System.out.println(f+"-----"+str);
    }
    public static void main(String[] args)
    {
        System.out.println("program statrs ");
        Test t=new Test();
        t.m1(10,'a');
        t.m2(10.2f,"ratna");
    }
}
```

**Ex :- static methods without parameters.**

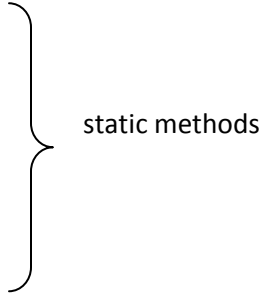
class Test

```
{
    static void durga()
    {
        System.out.println("durgasoftware solutions");
    }
    static void soft()
    {
        System.out.println("software solutions");
    }
    public static void main(String[] args)
    {
        durga()
        soft()
    }
}
```

**Ex:-static methods with parameters**

class Test

```
{
    static void m1(int i,char ch)
    {
        System.out.println(i+"-----"+ch);
    }
    static void m2(float f,String str)
    {
        System.out.println(f+"-----"+str);
    }
    public static void main(String[] args)
    {
        m1(10,'a');
        m2(10.2f,"ratna");
    }
}
```



**Ex :-while calling methods it is possible to provide the variables as a parameter values to the methods.**

class Test

```
{
    void m1(int a,int b)
    {
        System.out.println(a);
        System.out.println(b);
    }
    void m2(boolean b)
    {
        System.out.println(b);
    }

    public static void main(String[] args)
    {
        int a=10;
        int b=20;
        boolean b=true;
        Test t=new Test();
        t.m1(a,b);
        t.m2(b);
    }
}
```

**m1()**--→**calling** --→**m2()**----→**calling**-----→ **m3()**  
m1()<-----after completion-**m2()**<-----after completion **m3()**

**Ex:-calling of methods**

```
class Test
{
    void m1()
    {
        m2();
        System.out.println("m1 ratan");
    }
    void m2()
    {
        m3(100);
        System.out.println("m2 durga");
        M3(200);
    }
    void m3(int a)
    {
        System.out.println(a);
        System.out.println("m3 naresh");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}
```

**Ex :- methods with return type.**

```
class Test
{
    static int add(int a,int b)
    {
        int c=a+b;
        return c;
    }
    static float mul(int a,int b)
    {
        int c=a*b;
        return c;
    }
    public static void main(String[] args)
    {
        int a=add(10,20);
        System.out.println(a);
        float b=mul(100,200);
        System.out.println(b);
    }
}
```



**Ex :-for the java methods return type is mandatory otherwise the compilation will raise error return type required.**

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

**Ex :-Duplicate method signatures are not allowed in java language if we are declaring duplicate method signatures the compiler will raise compilation error m1() is already defined in test**

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

**Ex :-**

- a. Declaring the class inside the class is called inner classes concept java supports inner methods concept.
- b. Declaring the methods inside the methods inner methods concept java not supporting inner methods concept.

```
class Test
{
    void m1()
    {
        void m2()
        {
            System.out.println("this ia a m2-static method");
        }
        System.out.println("this ia a m1-instance method");
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        t.m1();
    }
};
```

**Stack Mechanism:-**

- 1) Whenever we are executing the program stack memory is created.
- 2) The each and every method is called by JVM that method entries are stored in the stack memory and whatever the variables which are available within the method that variables are stored in the stack.
- 3) If the method is completed the entry is automatically deleted from the stack means that variables are automatically deleted from the stack.
- 4) Based on the 1 & 2 points the local variables scope is only within the method.

**Ex :-**class Test

```
{
    void deposit(int accno)
    {
        System.out.println("money is deposited into    "+accno);
    }
    void withdraw(float amount)
    {
        System.out.println("money  withdraw is over amount    "+amount);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.deposit(111);
        t.withdraw(10000);
    }
}
```

Step 1:- One empty stack is created

Step 2:-

Whenever the JVM calls the main method the main method entry is stored in the stack memory.  
The stored entry is deleted whenever the main method is completed.

Step 3:-

Whenever the JVM is calling the deposit () the method entry is stored in the stack and local variables are store in the stack. The local variables are deleted whenever the JVM completes the Deposit () method execution. Hence the local variables scope is only within the method.

Step 4 :-

The deposit method is completed Then deposit () method is deleted from the stack.

Step 5 :-

Only main method call present in the stack.

Step 6:-

Whenever the JVM calls the withdraw method the entry is stored in the stack.

Step 7:-

Whenever the Withdraw method is completed the entry is deleted from the stack.

Step 8:-

Whenever the main method is completed the main method is deleted from the stack.

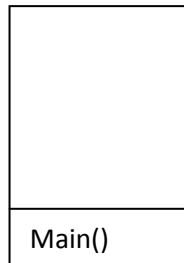
Step 9:- the empty stack is deleted from the memory.

Step 1:-



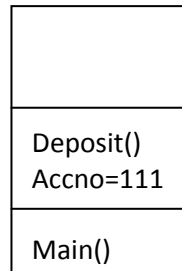
**Empty stack**

step 2 :-



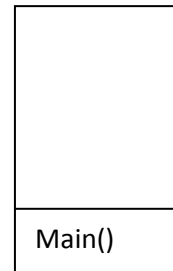
**stack**

step 3:-



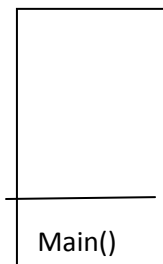
**stack**

step 4:-



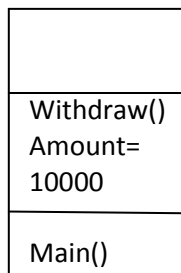
**stack**

Step 5:-



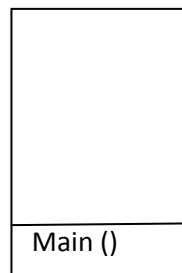
**Stack**

step 6:-



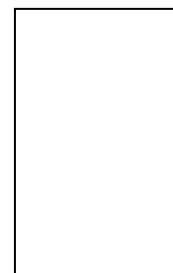
**stack**

step 7:-



**stack**

step 8 :-



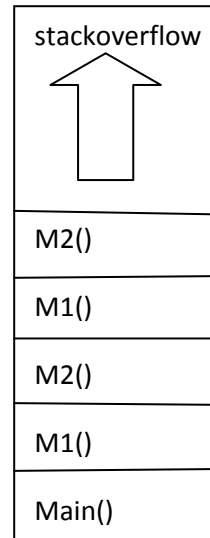
**empty stack**

**Ex :-we are getting StackOverflowError**

```

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

```

**Stack****Ex:-**

```

class Operations
{
    int a;
    int b;
    void add()
    {
        System.out.println(a+b);
    }
};

class Test
{
    public static void main(String[] args)
    {
        Operations o1=new Operations();
        Operations o2=new Operations();
        o1.a=100;
        o1.b=200;
        o2.a=1000;
        o2.b=2000;
        o1.add();
        o2.add();
    }
};

```

**Ex:-there are two types of instance methods**

- 1) Accessor methods just used to read the data. To read the reading the data use getters methods.
- 2) Mutator methods used to store the data and modify the data for the storing of data use setters methods.

```
class Test
{
    String name;
    int id;

    //mutator method we are able to access and the data
    void setName(String name)
    {
        this.name=name;
    }
    void setId(int id)
    {
        this.id=id;
    }

    //accessor methods are used to read the data
    String getName()
    {
        return name;
    }
    int getId()
    {
        return id;
    }

    public static void main(String[] args)
    {
        Test t=new Test();
        t.setName("ratan");
        t.setId(12345);
        String name=t.getName();
        System.out.println(name);
        int id=t.getId();
        System.out.println(id);
    }
}
```

**Local variable vs instance variable vs static variable :-**

Heap memory	Stack memory
1) It is used to store the objects	1) It is used to store the function calls and local variables.
2) We are getting outOfMemoryError	2) If there is no memory in the stack to store method calls or local variables the JVM will throw the StackOverflowError.
3) Having more memory compared with the stack memory.	3) Stack memory is very less memory when compared with the heap memory
4) Heap memory is also known as public memory. This is applicable all the objects. This memory is shared by all threads.	4) Stack memory also known as private memory. This is applicable only for owners.
5) The objects are created in the heap memory with the help of new operator	Destroy when the method is completed. JVM is creating stack memory

**Methods practice example:-**

```

class Test
{
    int a=10;
    int b=20;

    static int c=30;
    static int d=40;

    void m1(char ch,String str)
    {
        System.out.println(ch);
        System.out.println(str);
    }
    void m2(int a,double d,boolean b)
    {
        System.out.println(a);
        System.out.println(d);
        System.out.println(b);
    }
    static void m3(String str)
    {
        System.out.println(str);
    }
}

```

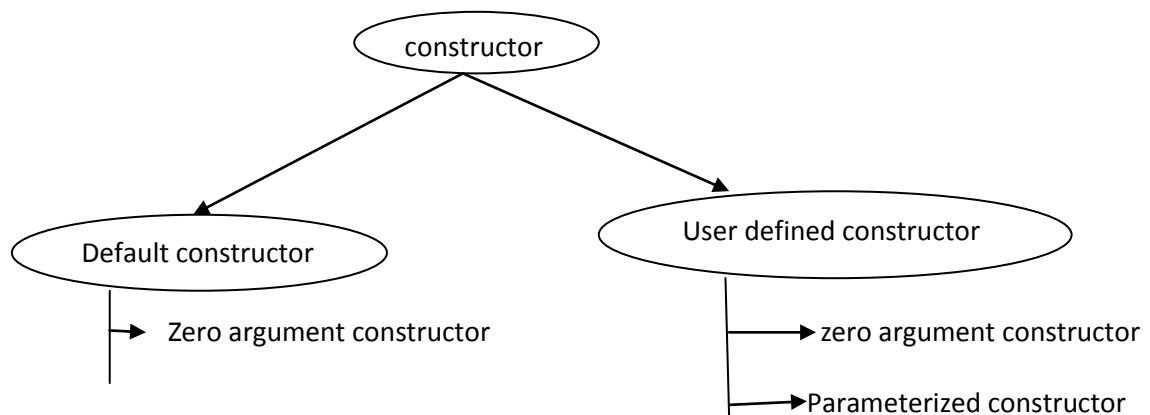
```
}  
static void m4(char ch,char ch1)  
{  
    System.out.println(ch);  
    System.out.println(ch1);  
}  
  
public static void main(String[] args)  
{  
    Test t=new Test();  
    System.out.println(t.a);  
    System.out.println(t.b);  
    System.out.println(c);  
    System.out.println(d);  
    t.m1('a',"ratan");  
    t.m2(100,10.8,true);  
    m3("anu");  
    m4('d','w');  
}  
}
```

**CONSTRUCTORS:-**

- 1) Constructors are executed as part of the object creation.
- 2) If we want to perform any operation at the time of object creation the suitable place is constructor.
- 3) Inside the java programming the compiler is able to generate the constructor and user is able to declare the constructor. so the constructors are provided by compiler and user.

There are two types of constructors

- 1) Default Constructor.
  - a. Zero argument constructor.
- 2) User defined Constructor
  - a. zero argument constructor
  - b. parameterized constructor



**Default Constructor:-**

- 1) in the java programming if we are not providing any constructor in the class then compiler provides default constructor.
- 2) The default constructor is provided by the compiler at the time of compilation.
- 3) The default constructor provided by the compiler it is always zero argument constructors with empty implementation.
- 4) The compiler generated default constructor default constructor is executed by the JVM at the time of execution.

**Ex:- Before compilation**

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

**After compilation:-**

```
class Test
{
    Test() } default constructor
           } provided by compiler
    void good()
    {
        System.out.println("good girl");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.good();
    }
}
```

**User defined constructors:-**

Based on the user requirement user can provide zero argument constructor as well as parameterized constructor.

**Rules to declare a constructor:-**

- 1) Constructor name must be same as its class name.
- 2) Constructor doesn't have no explicit return type if we are providing return type we are getting any compilation error and we are not getting any runtime errors just that constructor treated as normal method.
- 3) In the class it is possible to provide any number of constructors.



**user provided zero argument constructors.**

```

class Test
{
    Test()
    {
        System.out.println("0-arg cons by user ");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}

```

**Compiler provided default constructor executed**

```

class Test
{
    /*Test()
    {
    }
    */
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}

```

} compiler provided default constructor

**Ex 2:- user provided parameterized constructor is executed.**

```

class Test
{
    Test(int i)
    {
        System.out.println(i);
    }
    void good()
    {
        System.out.println("good girl");
    }
    public static void main(String[] args)
    {
        Test t=new Test(10);
        t.good();
    }
}

```

**Ex 3:-inside the class it is possible to take any number of constructors.**

```

class Test
{
    Test()
    {
        System.out.println("this is zero argument constructor");
    }
    Test(int i)
    {
        System.out.println(i);
    }
    Test(int i,String str)
    {
        System.out.println(i);
        System.out.println(str);
    }
}

```

} constructor overloading

```
        public static void main(String[] args)
        {
            Test t1=new Test();
            Test t2=new Test(10);
            Test t3=new Test(100,"rattaiah");
        }
    }
```

**Practice example:-**

```
class Test
{
    //2-instance variables
    int a=1000;
    int b=2000;
    //2-static variables
    static int c=3000;
    static int d=4000;
    //2-instance methods
    void m1(int a,char ch)
    {
        System.out.println(a);
        System.out.println(ch);
    }
    void m2(int a,int b,int c)
    {
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    //2-static methods
    static void m3(String str)
    {
        System.out.println(str);
    }
    static void m4(String str1,String str2)
    {
        System.out.println(str1);
        System.out.println(str2);
    }
    //2-constructors
    Test(char ch,boolean b)
    {
        System.out.println(ch);
        System.out.println(b);
    }
    Test(int a)
    {
        System.out.println(a);
    }
}
```

```
public static void main(String[] args)
{
    //calling of constructors
    Test t1=new Test('s',true);
    Test t2=new Test(10);
    //calling of instance variables
    System.out.println(t1.a);
    System.out.println(t2.b);
    //calling of static variables
    System.out.println(c);
    System.out.println(d);
    //calling of instance methods
    t1.m1(100,'s');
    t1.m2(100,200,300);
    //calling of static methods
    m3("durga");
    m4("anu","dhanu");
}
};
```

**this keyword:-**

This keyword is used to represent

1. Current class variables.
2. Current class methods.
3. Current class constructors.

**Current class variables:-****Ex 1:-No need of this keyword**

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

**Note:- this keyword is required**

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

**Ex:-conversion of local variables into the instance/static variables**

```

class Test
{
    static int i;
    int j;
    void values(int a,int b)
    {
        i=a;
        j=b;
    }
    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();
    }
}

```

**Ex:- to convert local variables into the instance/static variables need of this keyword.**

```

class Test
{
    static int a;
    int b;
    void values(int a,int b)
    {
        this.a=a;
        this.b=b;
    }
    void add()
    {
        System.out.println(a+b);
    }
    void mul()
    {
        System.out.println(a*b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.values(100,200);
        t.add();
        t.mul();
    }
}

```

**Ex:- to call the current class methods we have to use this keyword The both examples gives same output( both examples are same)**

```

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

```

```

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

```

**Calling of current class constructors:-****Syntax :-**

This(parameter -list);

Ex:-     this();-----→to call zero argument constructor

Ex:-     this(10)-----→to call one argument constructor

**Ex:- In side the Constructors this keyword must be first statement in constructors(no compilation error)**

```
class Test
{
    Test()
    {
        this(10);
        System.out.println("0 arg");
    }
    Test(int a)
    {
        this(10,20);
        System.out.println(a);
    }
    Test(int a,int b)
    {
        System.out.println(a+"-----"+b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
};
```

**note:-**

1. inside the constructors constructor calling must be the first statement of the constructor otherwise the compiler will raise compilation error.
2. The above rule applicable only for constructors.

**Compilation error:-**

```
class Test
{
    Test()
    {
        System.out.println("0 arg");
        this(10);
    }
    Test(int a)
    {
        System.out.println(a);
        this(10,20);
    }
    Test(int a,int b)
    {
        System.out.println(a+"-----"+b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
};
```

**Ex:- conversion of local variables to the instance variables by using this keyword**

```
import java.util.*;
class Student
{
    String sname;
    int sno;

    Student()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("enter sname:");
        String sname=s.next();
        this.sname=sname;
        System.out.println("enter no");
        int sno=s.nextInt();
        this.sno=sno;
    }
    void display()
    {
        System.out.println("**student details*****");
        System.out.println("student name:---"+sname);
        System.out.println("student no:---"+sno);
    }
    public static void main(String[] args)
    {
        Student s=new Student();
        s.display();
    }
}
```

**Ex:-providing dynamic input to instance variables directly.**

```
import java.util.*;
class Student
{
    String sname;
    int sno;

    Student()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("enter sname:");
        sname=s.next();
        System.out.println("enter sno");
        sno=s.nextInt();
    }

    void display()
    {
        System.out.println("**student deatils**");
        System.out.println("student sname"+sname);
        System.out.println("student sno"+sno);
    }
    public static void main(String[] args)
    {
        Student s=new Student();
    }
}
```

**Oops concepts:-**

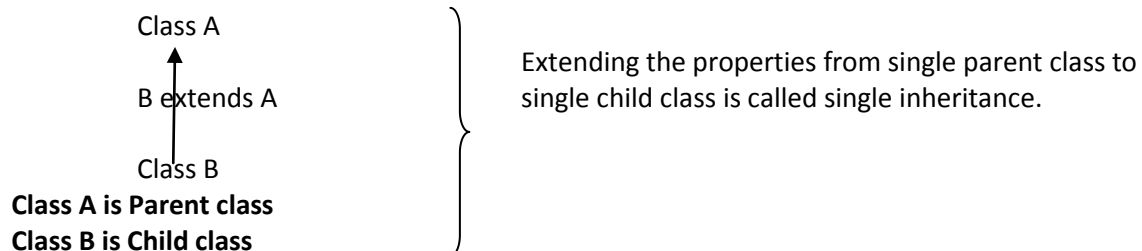
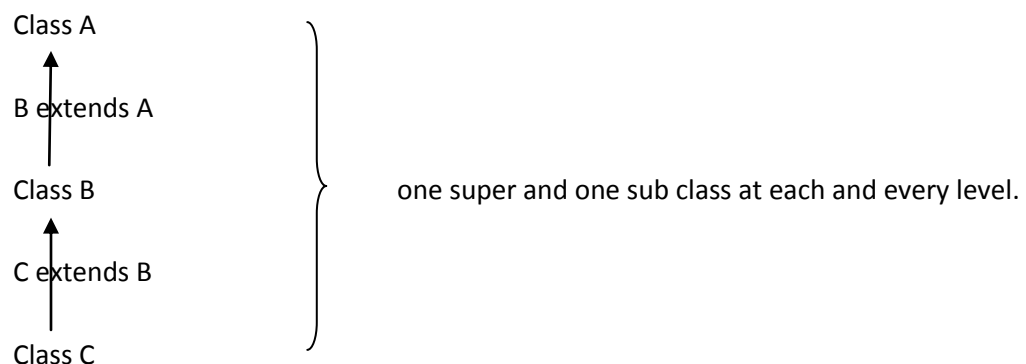
- 1)inheritance
- 2)polymorphisum
- 3)abstraction
- 4)encapsulation

**Inheritance:-**

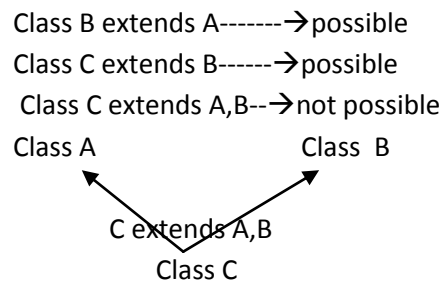
The process of getting properties and behaviors from one class to another class is called inheritance.

Properties : variables  
Behaviors : methods

1. The main purpose of the inheritance is code extensibility whenever we are extending automatically the code is reused.
2. In inheritance one class giving the properties and behavior and another class is taking the properties and behavior.
3. Inheritance is also known as **is-a** relationship means two classes are belongs to the same hierarchy.
4. By using **extends** keyword we are achieving inheritance concept.
5. In the inheritance the person who is giving the properties is called parent the person who is taking the properties is called child.
6. To reduce length of the code and redundancy of the code sun peoples introducing inheritance concept.

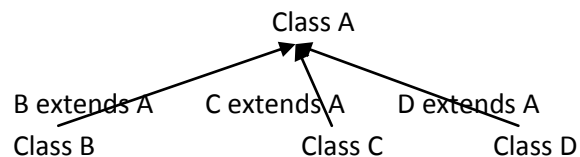
**Types of inheritance:-****Single inheritance:-****1) Multilevel inheritance:-****2) Multiple inheritance:-**

The process of getting properties and behaviors form more than one super class to the one child class. The multiple inheritance is not possible in the java language so one class can extends only one class at time it is not possible to extends more than one class at time.



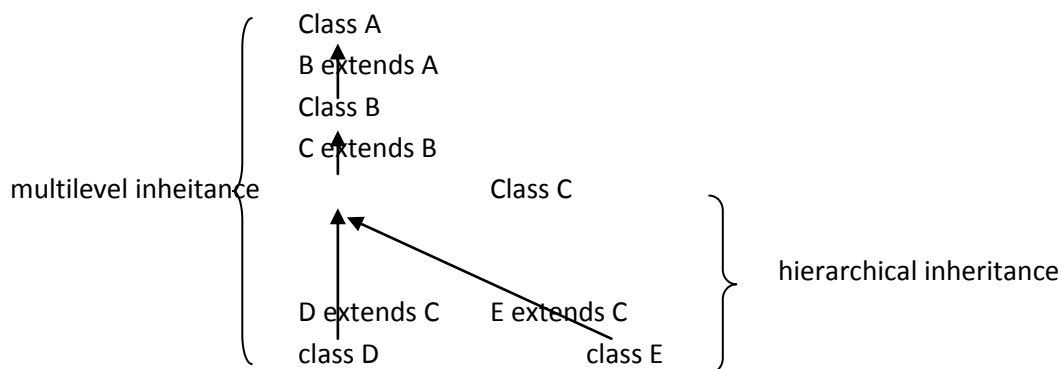
### 3) Hierarchical inheritance:-

The process of getting properties and behaviors from one super class to the more than one sub classes is called hierarchical inheritance.



### 4) Hybrid inheritance:-

Combination of any two inheritances is called as hybrid inheritance. If are taking the multilevel and hierarchical that combination is called hybrid inheritance.



#### Before inheritance :-

```

class A
{
    Void m1();
    Void m2();
};
class B
{
    Void m1();
    Void m2();
    Void m3();
    Void m4();
};
class C
{
    Void m1();
    Void m2();
    Void m3();
    Void m4();
    Void m5();
    Void m6();
};
  
```



**After inheritance:-**

class A

{

Void m1()

Void m2()

Void m3()

}

class B extends A

{

Void m4()

**Note 1:-**

It is possible to create objects for both parent and child classes.

- a. If we are creating object for parent class it is possible to call only parent specific methods.

A a=new A();

a.m1();

a.m2();

a.m3();

- b. if we are creating object for child class it is possible to call parent specific and child specific.

B b=new B();

b.m1();

b.m2();

b.m3();

b.m4();

b.m5();

- c. if we are creating object for child class it is possible to call parent specific methods and child specific methods.

C c=new C();

c.m1();

c.m2();

c.m3();

c.m4();

c.m5();

c.m6();

**Ex:-**

class Parent

{

public void m1()

{

System.out.println("m1-method");

}

public void m2()

{

System.out.println("m2-method");

}

}

Void m5()

}

Class C extends B

{

Void m6();

}

derived class

```
class Child extends Parent
{
    public void m3()
    {
        System.out.println("m3-method");
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.m1();
        c.m2();
        c.m3();
        Parent p=new Parent();
        p.m1();
        p.m2();
    }
}
```

**Note:-**

1. Every class in the java programming is a child class of object.
2. The root class for all java classes is Object class.
3. The default package in the java programming is java.lang package.

**Both declaration are same:-**

class Test	class Test extends Object
{	{
};	};
class String	class String extends Object
{	{
};	};

**Final:-**

- Final is the modifier applicable for classes, methods and variables (for all instance, Static and local variables).
- if a class is declared as final, then we cannot inherit that class i.e., we cannot create any child class for that final class.
- Every method present inside a final class is always final by default but every variable present inside the final class need not be final.
- The main advantage of final modifier is we can achieve security as no one can be allowed to change our implementation.
- But the main disadvantage of final keyword is we are missing key benefits of OOPS like

inheritance and polymorphism. Hence is there is no specific requirement never recommended to use final modifier.

### Case 1:-

**Ex :-overriding the method is possible.**

```
class Parent
{
    void andhra()
    {
        System.out.println("Andhra Pradesh");
    }
}
class Child extends Parent
{
    void andhra()
    {
        System.out.println("division is not possible");
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.andhra();
    }
};
```

### Case 2:-

**child class creation is possible**

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

### Case 3:-

**Reassignment is possible.**

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

**Ex:- overrrding the method is not possible**

```
class Parent
{
    void andhra()
    {
        System.out.println("Andhra Pradesh");
    }
}
class Child extends Parent
{
    final void andhra()
    {
        System.out.println("division is not possible");
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.andhra();
    }
};
```

**child class creation is not possible getting compilation error.**

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

**For the final variables reassignment is not possible.**

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

**Note :-**

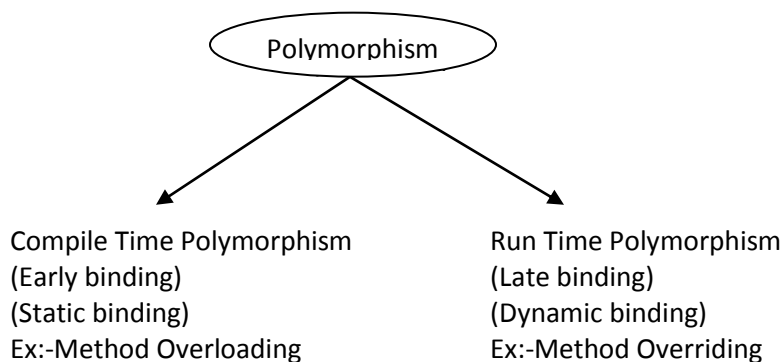
**Every method present inside a final class is always final by default but every variable present inside the final class need not be final.**

final class Test

```
{
    int a=10;
    void m1()
    {
        System.out.println("m1 method is final");
        System.out.println(a+10);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}
```

**POLYMORPHISM:-**

- 1) One thing can exhibits more than one form called polymorphism.
- 2) The ability to appear in more forms.
- 3) Polymorphism is a Greek word poly means **many** and morphism means forms.

**Method Overloading:-**

- 1) Two methods are said to be overloaded methods if and only if two methods are having same name but different argument list.
- 2) We can overload the methods in two ways in java language
  - a. Provide the different number of arguments to the same methods.
  - b. Provide the same number of arguments with different data types.
- 3) If we want achieve overloading one class is enough.
- 4) It is possible to overload any number of methods.

**Types of overloading:-**

- |                            |   |
|----------------------------|---|
| a. Method overloading      | } explicitly by the programmer                      |
| b. Constructor overloading |   |
| c. Operator overloading    | } implicitly by the JVM('+ addition& concatenation) |

**Method overloading:-**

Ex:-

class Test

```
{
    void m1(char ch)
    {
        System.out.println(" char-arg constructor ");
    }
    void m1(int i)
    {
        System.out.println("int-arg constructor ");
    }
    void m1(int i,int j)
    {
        System.out.println(i+"-----"+j);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1('a');
        t.m1(10);
        t.m1(10,20);
    }
}
```

Overloaded Methods

**Ex:-**

class Test

```
{
    public static void main(String[] args)
    {
        System.out.println("String[] parameter main method start");
        main(100);
        main('r');
    }
    public static void main(int a)
    {
        System.out.println("integer parameter argument");
        System.out.println(a);
    }
    public static void main(char ch)
    {
        System.out.println("character paramer argument");
        System.out.println(ch);
    }
}
```

overloaded main method

Ex :-

```
class Test
{
    void sum(int a,int b)
    {
        System.out.println("int arguments method");
        System.out.println(a+b);
    }
    void sum(long a,long b)
    {
        System.out.println("long arguments method");
        System.out.println(a+b);
    }

    public static void main(String[] args)
    {
        Test t=new Test();
        t.sum(10,20);
        t.sum(100L,200L);
    }
}
```

### Constructor Overloading:-

The constructors are having same name but different argument list such type of constructors are called Overloaded constructors.

```
class Test
{
    Test()
    {
        System.out.println("0-arg constructor");
    }
    Test(int i)
    {
        System.out.println("int argument constructor");
    }
    Test(char ch,int i)
    {
        System.out.println(ch+"-----"+i);
    }
    public static void main(String[] args)
    {
        Test t1=new Test();//calling of zero argument constructor
        Test t2=new Test(10);//calling of one argument constructor
        Test t3=new Test('a',100);//calling of two argument constructor
    }
}
```

Overloaded Constructor

**Operator overloading:-**

One operator can act as more than one form is called Operator overloading . the only operator overloaded in java language is '+'.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str1="Rattaiah";
        String str2="Durga";
        System.out.println(str1);
        System.out.println(str1);
        System.out.println(str1+str2);
        int a=10;
        int b=20;
        System.out.println(a);
        System.out.println(b);
        System.out.println(a+b);
    }
}
```

**Method Overriding:-**

- 1) If the child class not satisfy the parent class method implementation then it is possible to override that method in the child class based on child class requirement.
- 2) If we want to achieve method overriding we need two class(child and parent).
- 3) In the overriding concept the child class and parent class method signatures must be same otherwise we are getting compilation error.

The parent class method is called-----→overridden method

The child class method is called-----→overriding method

**Ex:-methodOverriding**

```
class Parent
{
    void property()
    {
        System.out.println("money+land+hhouse");
    }
    void marry()
    {
        System.out.println("black girl");
    }
};

class Child extends Parent
{
    void marry()
    {
        System.out.println("white girl/red girl");
    }
    public static void main(String[] args)
```

} Overridden method

} overriding method

```

{
    Child c=new Child();
    c.property();
    c.marry();
    Parent p=new Parent();
    p.property();
    p.marry();
};

```

**Abstraction:-**

Hiding the internal implementation and highlighting the set of services that process is called abstraction.

Ex:-

- a. Bank ATM Screens (Hiding the internal implementation and highlighting set of services like withdraw, money transfer, mobile registration).
- b. Mobile phones (The mobile persons are hiding the internal circuit implementation and highlighting touch screen).
- c. Syllabus copy (the institutions persons just highlighting the set of contents that persons provided the persons are not highlighting the whole content).

Ex:- Abstract classes  
Interfaces

The way of representation the methods are divided into two types

- 1) Normal methods
- 2) Abstract methods

**Normal methods:-**

Normal method is a method which contains declaration as well as implementation.

Ex:-

```

Void m1()
{
    -----
    -----body;
    -----
}

```

**Abstract methods:-**

- The method which is having declaration but not implementations such type of methods are called abstract Method. Hence every abstract method should end with “;”.
- The child classes are responsible to provide implementation for parent class abstract methods.

Ex: - void m1 (); -----abstract method



Based on above representation of methods the classes are divided into two types

- 1) Normal classes
- 2) Abstract classes

#### Normal classes:-

Normal class is a java class it contains only normal methods.

#### Abstract class:-

- Abstract class is a java class which contains at least one abstract method.
- To specify the particular class is abstract and particular method is abstract method to the compiler use abstract modifier.
- For the abstract classes it is not possible to create an object. Because it contains the unimplemented methods.
- For any class if we don't want instantiation then we have to declare that class as abstract i.e., for abstract classes instantiation (creation of object) is not possible.

#### Normal class:-

```
class Test
{
    void m1()
    {
        body;
    }
    void m2()
    {
        body;
    }
    void m3()
    {
        body
    }
}
```

#### Abstract class:-

```
Abstract class Test
{
    void m1() {    body;  }
    void m2() {    body;  }
    Abstract void m3();
}
```

#### Abstract class:-

```
class Test
{
    Abstract void m1();
    Abstract void m2();
    Abstract void m3();
}
```

**Ex :-the abstract class contains abstract methods for that abstract methods provide the implementation in child classes.**

```
abstract class Test
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}
class AbstractDemo extends Test
{
    void m1()
    {
        System.out.println("m1-method");
    }
    void m2()
```

```
    {
        System.out.println("m2-method");
    }
    void m3()
    {
        System.out.println("m3-method");
    }
    public static void main(String[] args)
    {
        AbstractDemo ad=new AbstractDemo();
        ad.m1();
        ad.m2();
        ad.m3();
    }
};
```

**Ex :- if the child class is unable to provide the implementation for parent class abstract methods at that situation we can declare that class is an abstract then take one more child class in that class provide the implementation for remaining methods.**

```
abstract class Test
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}
abstract class AbstractDemo1 extends Test
{
    void m1()
    {
        System.out.println("m1-method");
    }
    void m2()
    {
        System.out.println("m2-method");
    }
};
class AbstractDemo extends AbstractDemo1
{
    void m3()
    {
        System.out.println("m3-method");
    }
    public static void main(String[] args)
    {
        AbstractDemo ad=new AbstractDemo();
        ad.m1();
        ad.m2();
        ad.m3();
    }
}
```

```

    }
};

```

**Note :-**

In the above program

**abstract Test**-----→not possible to create an object.

**abstract AbstractDemo1**-----→not possible to create an object.

**AbstractDemo**-----→possible to create a object.

**Note :-**

we can take the any number of child classes but we have to provide the implementation for each and every abstract method.

**Ex:- for the abstract methods it is possible to provide any type of return type(void, int, char, Boolean.....)**

```

abstract class Test1
{
    abstract int m1();
    abstract boolean m2();
}
class Test2 extends Test1
{
    int m1()
    {
        System.out.println("m1 method");
        return 100;
    }
    boolean m2()
    {
        System.out.println("m2 method");
        return true;
    }
    public static void main(String[] args)
    {
        Test2 t=new Test2();
        int a=t.m1();
        System.out.println(a);
        boolean b=t.m2();
        System.out.println(b);
    }
};

```

**Ex:- for the abstract methods it is possible to provide arguments.**

```

abstract class Test1
{
    abstract int m1(int a,int b);
    abstract boolean m2(char ch);
}
class Test2 extends Test1
{

```

```
int m1(int a,int b)
{
    System.out.println("m1 method");
    System.out.println(a);
    System.out.println(b);
    return 100;
}
boolean m2(char ch)
{
    System.out.println("m2 method");
    System.out.println(ch);
    return true;
}
public static void main(String[] args)
{
    Test2 t=new Test2();
    int a=t.m1(100,200);
    System.out.println(a);
    boolean b=t.m2('c');
    System.out.println(b);
}
};
```

**Ex :-**

**The abstract classes is java class it contains zero number of abstract methods.**

**Even though class does not contain any abstract method still we can declare the class as abstract i.e. abstract class can contain zero number of abstract methods. The abstract classes it is not possible to create object.**

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

**Encapsulation:-**

The process of binding the data and code as a single unit is called encapsulation.

We are able to provide more encapsulation by taking the private data(variables) members.

To get and set the values from private members use getters and setters to set the data and to get the data.

Ex:-

class Encapsulation

```
{
    private int sid;
    private int sname;
    public void setSid(int x)
    {
        this.sid=sid;
    }
    public int getSid()
    {
        return sid;
    }
    public void setSname(String sname)
    {
        this.sname=sname;
    }
    public String getSname()
    {
        return sname;
    }
};
```

**To access encapsulated use following code:-**

class Test

```
{
    public static void main(String[] args)
    {
        Encapsulation e=new Encapsulation();
        e.setSid(100);
        e.setSname("ratan");
        int num=e.getSid();
        String name=e.getSname();
        System.out.println(num);
        System.out.println(name);
    }
};
```

**Super:-** Super keyword is used to represent

- 1) Call the Super class variable.
- 2) Call the super class constructor.
- 3) Call the super class methods.

**Calling of super class variables:-**

**Super keyword is not required**

```
class Parent
{
    int a=10;
    int b=20;
};
class Child extends Parent
{
    int x=100;
    int y=200;
    void add(int i,int j)
    {
        System.out.println(i+j);
        System.out.println(x+y);
        System.out.println(a+b);
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.add(1000,2000);
    }
};
```

**Super keyword is required**

```
class Test1
{
    int a=10;
    int b=20;
};
class Test extends Test1
{
    int a=100;
    int b=200;
    void add(int a,int b)
    {
        System.out.println(a+b);

        System.out.println(this.a+this.b);

        System.out.println(super.a+super.b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.add(1000,2000);
    }
};
```

**Calling of super class methods:-****Super keyword is not required**

```
class Parent
{
    void m1()
    {
        System.out.println("parent class method");
    }
};
class Child extends Parent
{
    void m2()
    {
        m1();
        System.out.println("child class method");
    }
    void m3()
    {
        m1();
        System.out.println("child class method");
        m2();
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.m3();
    }
};
```

**Super keyword is required**

```
class Parent
{
    void m1()
    {
        System.out.println("parent class method");
    }
};
class Child extends Parent
{
    void m1()
    {
        System.out.println("child class method");
    }
    void m2()
    {
        this.m1();
        System.out.println("child class method");
        super.m1();
    }
    public static void main(String[] args)
    {
        Child c=new Child();
        c.m2();
    }
};
```

**Calling of super class constructors:-**

Ex 1:- inside the constructors super keyword must be first statement of the constructor otherwise the compiler raise compilation error.

**No compilation error**

```
class Test1
{
    Test1(int i,int j)
    {
        System.out.println(i);
        System.out.println(j);
    }
    System.out.println("two arg constructor");
};
class Test extends Test1
{
    Test(int i)
    {
        super(100,200);
        System.out.println("int -arg constructor");
    }
    public static void main(String[] args)
    {
        Test t=new Test(10);
    }
}
```

**Compilation error**

```
class Test1
{
    Test1(int i,int j)
    {
        System.out.println(i);
        System.out.println(j);
    }
    System.out.println("two arg constructor");
};
class Test extends Test1
{
    Test(int i)
    {
        System.out.println("int -arg constructor");
        super(100,200);
    }
    public static void main(String[] args)
    {
        Test t=new Test(10);
    }
}
```

**Note :- inside the constructors it is possible to call only one constructor at a time that calling must be first statement of the constructor.**



**The compiler will provide the super() keyword at the time of compilation.**

```
class Test2
{
    Test2()
    {
//super(); provided by compiler
        System.out.println("software");
    }
};
class Test1 extends Test2
{
    Test1()
    {
//super(); provided by compiler
        System.out.println("durga");
    }
};
class Test extends Test1
{
    Test()
    {
//super(); provided by compiler
        System.out.println("0 arg constructor");
    }

    public static void main(String[] args)
    {
        Test t1=new Test();
    }
}
```

**In this example user is providing super() keyword .**

```
class Test2
{
    Test2()
    {
        Super();
        System.out.println("software");
    }
};
class Test1 extends Test2
{
    Test1()
    {
        Super();
        System.out.println("durga");
    }
};
class Test extends Test1
{
    Test()
    {
        super();
        System.out.println("0 arg constructor");
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
    }
}
```

**Note :-**

1. Inside the constructor if we are providing **super** and **this** key words at that situation the compiler will place the zero argument " super() ; " keyword at first line of the constructors.
2. If we are declaring any constructor calling then compiler wont generate any type of keywords.

**Parent class default constructor will be executed and in the child constructor the super keyword is provided by compiler**

```
class Parent
{
};
class Child extends Parent
{
    Child()
    {
        //super(); compiler provided
        System.out.println("child class 0 arg cons");
    }
    public static void main(String... ratan)
    {
        Child c=new Child();
    }
};
```

**Parent class userdefined constructor will be executed and in the child class the super keyword is provided by user.**

```
class Parent
{
    Parent()
    {
        System.out.println("");
    }
};
class Child extends Parent
{
    Child()
    {
        super();
        System.out.println("child class 0 arg cons");
    }
    public static void main(String... ratan)
    {
        Child c=new Child();
    }
};
```

**Ex:- parent class 0-arg cons is executed(provided by compiler default constructor).**

```
class Parent
{
};
class Child extends Parent
{
    Child()
    {
        super();
        System.out.println("child class 0 arg cons");
    }
    Child(int i)
    {
        super();
        System.out.println("child class 1 arg");
        System.out.println(i);
    }
    public static void main(String... ratan)
    {
        Child c1=new Child(100);
        Child c2=new Child();
    }
};
```

**Ex:- Object class constructor is executed.**

```

class Test extends Object
{
    Test()
    {
        //super(); provided by compiler
        System.out.println("0 arg cons");
    }
    public static void main(String... ratan)
    {
        Test t=new Test();
    }
};

```

**Main Method:-****Public static void main(String[] args)**

**Public:-** To provide access permission to the jvm declare main is public.

**Static :-** To provide direct access permission to the jvm declare main is static(with out creation of object able to access main method)

**Void:-** don't return any values to the JVM.

**String[] args:-** used to take command line arguments(the arguments passed from command prompt)

**String-----**→represents possible to take any type of argument.

**[ ]-----**→represent possible to take any number of arguments.

**Modifications on main():-**

- 1) instead of                   :       public static  
Possible to take               :       static public
- 2) the following declarations are valid  
string[] args  
String args[]  
String []args
- 3) instead of                   :       args  
possible to take               :       any variable name (a,b,c,-----rattaiah etc)
- 4) insted of                   :       String[] args  
possible to take               :       String... args(only three dots represent variable argument )
- 5) the applicable modifiers are
  - a. public
  - b. static
  - c. final
  - d. strictfp
  - e. synchronized

**Ex 1:-**

```
class Test
{
    final strictfp synchronized static public void main(String...ratan)
    {
        System.out.println("hello ratan sir");
    }
};
```

**Ex 2:-**

```
class Test1
{
    final strictfp synchronized static public void main(String...ratan)
    {
        System.out.println("Test-1");
    }
};
class Test2
{
    final strictfp synchronized static public void main(String...ratan)
    {
        System.out.println("Test-2");
    }
};
class Test3
{
    final strictfp synchronized static public void main(String...ratan)
    { System.out.println("Test-3");
    }
};
```

**Ex 3:-**

```
class Parent
{
    public static void main(String[] args)
    {
        System.out.println("parent class");
    }
};
class Child extends Parent
{
    public static void main(String[] args)
    {
        System.out.println("child class");
    }
};
```

**Compilation :-** javac Test.java

**Execution:-**      java Parent      java Child  
                          Parent class      child class

```
class Parent
{
    public static void main(String[] args)
    {
        System.out.println("parent class");
    }
};
class Child extends Parent
{
};
```

**Compilation :-** javac Test.java

**Execution:-**      java Parent      java Child  
                          Parent class      parent class

**Strictfp:-**

- Strictfp is a modifier applicable for classes and methods.
- If a method is declared as strictfp all floating point calculations in that method will follow IEEE754 standard. So that we will get platform independent results.
- If a class is declared as strictfp then every concrete method in that class will follow IEEE754 standard so we will get platform independent results.

**Ex 1 :-in the main() method the floating point calculations are follows IEEE754 format.**

```
class Test
```

```
{
    public static strictfp void main(String[] args)
    {
        System.out.println("main method follows IEEE754 standard");
    }
};
```

**Ex 2:- The main() and m1() methods the floating point calculations are follows IEEE754 format.**

```
strictfp class Test
```

```
{
    void m1()
    {
        System.out.println("m1 follows floating point calculations IEEE754 format");
    }
    public static void main(String[] args)
    {
        System.out.println("main method follows IEEE754 format");
    }
};
```

**Native:-**

- Native is the modifier only for methods but not for variables and classes.
- The native methods are implemented in some other languages like C and C++ hence native methods also known as "foreign method".
- For native methods implementation is always available in other languages and we are not responsible to provide implementation hence native method declaration should compulsory ends with ";".

```
Ex: -    public native String intern();
        Public static native void yield();
        Public static native void sleep(long);
```

**Which of the following declarations are valid:-**

- public static void main(String[] args)-----valid
- public static void main(String... a)-----valid
- public static int main(String... args)-----invalid
- static public void main(String a[])-----valid
- final public static void main(String[] args)---valid
- final strictfp public static void main(String[] args)----valid
- final strictfp synchronized public static void main(String... args)-----valid

**Command Line Arguments:-**

The arguments which are passed from command prompt is called command line arguments. We are passing command line arguments at the time program execution.

Ex1:-

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println(args.length);
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
};
```

Compilation : javac Test.java

Execution : java Test a b c

Output : 3

a

b

**a,b,c are the command line arguments**

Ex :-

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println(args.length);
        for (String str:args)
        {
            System.out.println(str);
        }
    }
};
```

Compilation : javac Test.java

Execution : java Test a b c "dhhfgh dhgdgh gdhg"

Output : 3

a

b

c

dhhfgh dhgdgh gdhg

**Var-arg method:-**

1. introduced in 1.5 version.
2. it allows the methods to take any number of parameters.

**Syntax:-(only 3 dots)**

Void m1(int... a)

The above m1() is allows any nuber of parameters.(0 or 1 or 2 or 3.....)

**Ex:-**

```

class Test
{
    void m1(int... a)
    {
        System.out.println("Ratan");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
        t.m1(10);
        t.m1(10,20);
    }
}

```

**Ex:-print the var-arg elements by using for-each loop.**

```

class Test
{
    void m1(int... a)
    {
        System.out.println("Ratan");
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
        t.m1(10);
        t.m1(10,20);
        t.m1(10,20,30,40);
    }
}

```

**Ex:-inside the variable argument method it is possible to declare only one argument otherwise the compiler will raise compilation error.**

Getting Compilation Error

```

class Test
{
    void m1(int... a,String... str)
    {
        System.out.println("Ratan");
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1(10,20,"ratan","sir","hello");
        t.m1(10,20,30,40);
    }
}

```

**Ex:- inside the methods the var-arg statement must be last statement**

```

class Test
{
    void m1(int a,double d,char ch,String... str)
    {
        System.out.println(a);
    }
}

```

```
        System.out.println(d);
        System.out.println(ch);
        for (String str1:str)
        {
            System.out.println(str1);
        }
    }
    public static void main(String... args)
    {
        Test t=new Test();
        t.m1(10,20.5,'s');
        t.m1(10,20.5,'s',"aaaa");
        t.m1(10,20.5,'s',"aaaa","bbb");
    }
};
```

Ex:- inside the java methods if we are declaring normal argument and variable argument at that situation the variable argument must be last parameter otherwise the compiler will raise compilation error.

Getting Compilation Error

Ex:-

```
class Test
{
    void m1(int... a,String str)
    {
        System.out.println("Ratan");
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1(10,20,"anu");
        t.m1(10,20,30,40,"hello");
    }
}
```



**Ex:- var-arg method vs overloading**

```

class Test
{
    void m1(int... a)
    {
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
    void m1(String... str)
    {
        for (String str1:str)
        {
            System.out.println(str1);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1(10,20,30);
        t.m1("ratan","durga");
    }
}

```

**Ex:-var-arg method vs ambiguity.**

```

class Test
{
    void m1(int... a)
    {
        for (int a1:a)
        {
            System.out.println(a1);
        }
    }
    void m1(String... str)
    {
        for (String str1:str)
        {
            System.out.println(str1);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1(10,20,30);
        t.m1("ratan","durga");
        t.m1();//compilation error ambiguous
    }
}

```

**Instance Blocks:-**

- 1) The instance blocks are executed irrespective of any condition.
- 2) The instance blocks and instance variables are executed before constructor execution. If you are giving a chance to the constructors then only instance blocks are executed.
- 3) In the class it is possible to take the any number of instance blocks. the execution order is top to bottom.
- 4) Instance blocks are executed based on the object creation. If we are creating ten objects ten times instance blocks will be executed.
- 5) If the source file contains inheritance concept at that situation first parent class instance block will be executed then child class instance blocks will be executed.

**Ex 1:-**

```

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

```

```

        Test t=new Test();
    }
};

```

**Ex 2:-**

**instance variables and instance blocks will be executed as part of the object creation just before the constructors execution.**

**The instance variables and instance blocks are having same priority at that situation the execution order is top to bottom.**

**1<sup>st</sup> instance variable****2<sup>nd</sup> instance block**

```

class Test
{
    int i=m1();
    int m1()
    {
        System.out.println("m-1 instance method");
        return 10;
    }
    {
        System.out.println("instance Block");
    }
    public static void main(String[] args)
    {
        Test3 t=new Test3();
    }
}

```

**1<sup>st</sup> instance block will be executed****2<sup>nd</sup> instance variable is executed**

```

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

```

**Static Block:-**

- 1) The static blocks are executed at the time of class loading into the memory.
- 2) Whenever we are using (java className) the class is loaded into the memory at that moment the static blocks are loaded
- 3) The static blocks are executed at only one time for each and every class loading. But the instance blocks are executed based on number of constructor's execution.
- 4) Without using main method it is possible to print some statements into the console in java language with the help of static blocks. but this rule is applicable only up to 1.5 version if we are using higher versions if we want to execute static block the main method is mandatory
- 5) In the higher version the static blocks are executed only if the class contains main method.
- 6) Based on the above reason we can say in the higher version it is not possible to print some statements into the console without using main method.
- 7) Whenever we are loading child class into the memory then automatically parent class is loaded hence the parent class static block is executed first and then child class static blocks are loaded.

**Ex 1:-**

```
class Test1
{
    static
    {
        System.out.println("durga");
        System.out.println("durga");
        System.out.println("durga");
        System.exit(0);
    }
}
```

**Ex 2:-**

```
class Test1
{
    static
    {
        System.out.println("Rattaiah");
        System.exit(0);
    }
    static
    {
        System.out.println("durga");
        System.out.println("durga");
        System.out.println("durga");
    }
}
```

System.exit(0):- the JVM shut downing.

**Ex:- write a program demonstrates the instance blocks and static blocks in the parent and child class relation.**

```
class Test1
{
    static
    {
        System.out.println(" parent class static block");
    }
    {
        System.out.println("parent class instance block");
    }
};
class Test extends Test1
{
    static
    {
        System.out.println("child calss static block");
    }
    {

```

```
        System.out.println("child class instance block 1");
    }
    {
        System.out.println("Child class instance block 2");
    }
    Test()
    {
    }
    Test(int i)
    {
        System.out.println(i);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
};
```

Output:-

Parent class static block  
Child class static block  
parent class instance block  
child class instance block 1  
child class instance block 2

**Ex:- (variables,methods,constructors,instanceBlocks,staticBlocks )**

```
class Test
{
    //instance variables
    int a=10;
    int b;

    //static variables
    static int i=100;
    static int j;

    //instance methods
    void m1()
    {
        System.out.println("m1-instance method");
    }
    void m2(int i,char ch,String str)
    {
        System.out.println(i);
        System.out.println(ch);
        System.out.println(str);
    }
}
```

```
//static methods
static void m3(int i,int j)
{
    System.out.println(i);
    System.out.println(j);
}
static void m4(String str,char ch)
{
    System.out.println(str);
    System.out.println(ch);
}

//constructors
Test()
{
    System.out.println("0 arg constructor");
}
Test(int i,int j)
{
    System.out.println("2 arg constructor");
}

//instance blocks
{
    System.out.println("instance block-1");
}

{
    System.out.println("instance block-2");
}

//static block
static
{
    System.out.println("durga");
    System.out.println("durga");
    System.out.println("durga");
}
static
{
    System.out.println("software");
    System.out.println("software");
    System.out.println("software");
}

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

```

    {
        Test t=new Test();
        Test t1=new Test(10,20);
        t.m1();
        t.m2(10,'r',"ratna");
        m3(100,200);
        m4("rattaiah",'d');
        System.out.println(t.a);
        System.out.println(t.b);
        System.out.println(i);
        System.out.println(j);
    }
};

```

**Ex:-(variables scope)**

```

class Test
{
    static int a=100;           //static variable
    int b;                     //instance variable
    {
        b=200;                 //initialization block
        int c=300;             //local variable
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    Test()
    {
        int d=400;             //local variables
        System.out.println(d);
    }
    void m1()
    {
        int e=500;             //local variables
        System.out.println(e);
        for (int i=0;i<10;i++) //localvariables
        {
            System.out.println(i);
        }
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}

```

**Scopes of the variables:-**

1. Static variable have the longest scope they are created when the class is loaded and they are survive as long as the class is saty in the JVM(java virtual machine)

2. Instance variables are next , they are created when a new object is created they are saty as long as object is removed.
3. The local variables next they are stored in the stack the scope of the local variables are up to the particular method or constructor of block.
4. Block level variables are available as long as block is executed.

### Dynamic input:-

The input given to the java application at the time of execution(runtime) is called Dynamic input. In java technology we can give the dynamic input in three ways.

- 1) By using BufferedReader class present in the java.io package(1.2 version)
- 2) By using Scanner class present in the java.util package(5.0 version)
- 3) By using Console class present in the java.io package(6.0 version)

### Take the dynamic data with the help of BufferedReader class:-

We are having two methods to read the data

- 1) Stirng readLine()-----→read the total line as a String format
- 2) Int read()-----→read the data character by character

Ex:-

```
import java.io.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        System.out.println("enter some text");//rattaiah
        String firstText=br.readLine();

        System.out.println("enter same text again");//rattaiah
        int secondText=br.read();

        System.out.println("first entered text "+firstText);//rattaiah
        System.out.println("second entered text "+secondText);//114
    }
}
```

Ex:-

```
import java.io.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

        System.out.println("enter first number");//23
        String a=br.readLine();
        System.out.println("enter second number");//23
        String b=br.readLine();
        System.out.println("addition is:"+(a+b));//2323
    }
}
```

```

        System.out.println("enter first number");25
        int a=Integer.parseInt(br.readLine());
        System.out.println("enter second number");25
        int b=Integer.parseInt(br.readLine());
        System.out.println("addition is:"+(a+b));50
    }
}

```

#### Java.util.Scanner:-

To get the integer value from the keyboard-----:s.nextInt()

To get the String value from the keyboard-----:s.next()

To get the floating values from the keyboard-----:s.nextFloat ();

```
import java.util.*;
```

```
class Test
```

```
{
```

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

```
    {
```

```
        while (true)
```

```
        {
```

```
            Scanner s=new Scanner(System.in);
```

```
            System.out.println("enter emp no");
```

```
            int eno=s.nextInt();
```

```
            System.out.println("enter emp name");
```

```
            String ename=s.next();
```

```
            System.out.println("enter emp salary");
```

```
            float esal=s.nextFloat();
```

```
            System.out.println("emp no----->" + eno);
```

```
            System.out.println("emp name---->" + ename);
```

```
            System.out.println("emp sal----->" + esal);
```

```
            System.out.println("do u want one more record(yes/no)");
```

```
            String option=s.next();
```

```
            if (option.equals("no"))
```

```
            {
```

```
                break;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

#### Dynamic input by using scanner class:-

```
import java.util.*;
```

```
class Test
```

```
{
```

```
    String ename;
```



```
int eid;
double esal;
int eage;
void details()
{
    Scanner s=new Scanner(System.in);
    System.out.println("enter emp name");
    String ename=s.next();
    this.ename=ename;
    System.out.println("enter emp id");
    int eid=s.nextInt();
    this.eid=eid;
    System.out.println("enter emp sal");
    double esal=s.nextDouble();
    this.esal=esal;
    System.out.println("enter age");
    int eage=s.nextInt();
    this.eage=eage;
}
void display()
{
    System.out.println("*****emp details*****");
    System.out.println(ename);
    System.out.println(eid);
    System.out.println(esal);
    System.out.println(eage);
}
void status()
{
    if (eage>40)
    {
        System.out.println("not eligible");
    }
    else
    {
        System.out.println("eligible");
    }
}
public final strictfp synchronized static void main(String... ratan)
{
    Test t=new Test();
    t.details();
    t.display();
    t.status();
}
};
```

## Interfaces

1. Interface is also one of the type of class it contains only abstract methods.
2. For the interfaces also .class files will be generated.
3. Each and every interface by default abstract hence it is not possible to create an object.
4. Interfaces not alternative for abstract class it is extension for abstract classes.
5. 100 % pure abstract class is called interface.
6. The Interface contains only abstract methods means unimplemented methods.
7. Interfaces giving the information about the functionalities it are not giving the information about internal implementation.
8. To provide implementation for abstract methods we have to take separate class that class we can called it as implementation class for that interface.
9. Interface can be implemented by using implements keyword.
10. For the interfaces also the inheritance concept is applicable.

### Syntax:-

Interface interface-name

Ex:- interface it1

### Note: -

if we are declaring or not By default interface methods are public abstract

Interface it1	} Both are same	abstract interface it1
{		{
Void m1();		public abstract void m1();
Void m2();		public abstract void m2();
Void m3();		public abstract void m3();
}		}

### Ex 1:-

```
Interface it1
{
Void m1();
Void m2();
Void m3();
}
Class Test implements it1
{
    Public void m1()
    {
        System.out.println("m1-method implementation");
    }
    Public void m2()
    {
        System.out.println("m2-method implementation");
    }
    Public void m3()
    {
        System.out.println("m3 –method implementation");
    }
}
```

```
    }  
    Public static void main(String[] args)  
    {  
        Test t=new Test();  
        t.m1();  
        t.m2();  
        t.m3();  
    }  
}
```

**Ex 2:-**

Interface it1

```
{  
Void m1();  
Void m2();  
Void m3();  
}
```

Abstract Class ImplClass implements it1

```
{  
    Public void m1()  
    {  
        System.out.println("m1-method implementation");  
    }  
}
```

} partially implemented class  
so we have to declare **abstract**

Class Test extends ImplClass

```
{  
    Public void m2()  
    {  
        System.out.println ("m2-method implementation");  
    }  
    Public void m3()  
    {  
        System.out.println ("m3 –method implementation");  
    }  
    Public static void main(String[] args)  
    {  
        Test t=new Test();  
        t.m1();  
        t.m2();  
        t.m3();  
    }  
}
```

} fully implemented class  
so we have to create  
obj then we can access.

Note:-

By inheritance in the Test contains abstract methods like m2() & m3() hence we have to declare that class as a abstract modifier.

We can take any number of classes but finally we have to provide implementation for the each and every method of interface.

**Note :- inheritance concept is also applicable for interfaces .**

Ex:-

**Without inheritance**

```
Interface it1
{
    Void m1();
    Void m2();
    Void m3();
}
Interface it2
{
    Void m4();
    Void m4();
}
```

Contains 3 methods

contains 2 methods

**With inheritance**

```
interface it1
{
    void m1();
    void m2();
    void m3();
}
interface it2 extends it1
{
    void m4();
    void m5();
}
```

Contains 3 Methods

Contains 5 Methods

Ex1:-

```
Interface it1
{
    Void m1();
    Void m2();
    Void m3();
}
Interface it2 extends it1
{
    Void m4();
    Void m5();
}
```

Class Test implements it1

```
{
Here we have to provide
Three methods
Implementation
m1 & m2 & m3
}
```

class Test implements it2

```
{
here we have to provide
five methods
implementation
m1 & m2 & m3& m4 & m5
}
```

class Test implements it1,it2

```
{
here we have to provide
five methods
implementation
m1 & m2 & m3 & m4 & m5
}
```

**Nested interfaces:-**

**Ex:-an interface can be declare inside the class is called nested interface.**

```
class Test1
{
    interface it1
    {
        void add();
    }
};
class Test2 implements Test1.it1
{
    public void add()
    {
        System.out.println("add method");
    }
    public static void main(String[] args)
    {
        Test2 t=new Test2();
        t.add();
    }
};
```

**Ex:-an interface can be declare inside the another interface is called nested interface.**

```
interface it2
{
    interface it1
    {
        void m1();
    }
};
class Test2 implements it2.it1
{
    public void m1()
    {
        System.out.println("m1 method");
    }
    public static void main(String[] args)
    {
        Test2 t=new Test2();
        t.m1();
    }
};
```

**Note:-**

```
class    extends class
class    implements interface
interface extends interface
```

```

class A extends B=====good
class A extends B,C====bad
class A implements it1=====good
class A implements it1,it2,it3====good

```

```

interface it1 extends it2-----good
interface it1 extends it2,it3---good
interface it1 extends A-----bad

```

```

class A extends B implements it1,i2,it3=====good
class A implements it1 extends B=====bad

```

**Adaptor class:-**

It is a intermediate class between the interface and user defined class. And it contains empty implementation of interface methods.

**Limitation of interface**

```

interface it
{
    void m1();
    void m2();
    ;
    ;
    void m100()
}
Class Test implements it
{
Must provide implementation of 100 methods
otherwise compiler raise compilation error
}

```

**advantage of adaptor classes**

```

interface it
{
    void m1();
    void m2();
    ;
    ;
    void m100()
}
class Adaptor implements it
{
    void m1(){ }
    void m2(){ }
    ;
    ;
    void m100(){ }
};
class Test implements it
{
must provide the 100 methods implementation
};
class Test extends Adaptor
{
provide the implementation of required
methods.
};

```

**Ex:-**

interface it1

```
{
    void m1();
    void m2();
}
```

**class X implements it1**

```
{
    public void m1(){}
    public void m2(){}
};
```

**class Test1 implements it1**

```
{
    public void m1()
    {
        System.out.println("m1 method");
    }
    public void m2()
    {
        System.out.println("m2 method");
    }
    public static void main(String[] args)
    {
        Test1 t=new Test1();
        t.m1();
        t.m2();
    }
};
```

**class Test2 extends X**

```
{
    public void m1()
    {
        System.out.println("adaptor m1
method");
    }
    public static void main(String... arhs)
    {
        Test2 t=new Test2();
        t.m1();
    }
};
```

## Packages

### Information regarding packages:-

- 1) The package contains group of related classes and interfaces.
- 2) The package is an encapsulation mechanism it is binding the related classes and interfaces.
- 3) We can declare a package with the help of package keyword.
- 4) Package is nothing but physical directory structure and it is providing clear-cut separation between the project modules.
- 5) Whenever we are dividing the project into the packages(modules) the sharability of the project will be increased.

### Syntax:-

```
Package package_name;  
Ex:- package com.dss;
```

The packages are divided into two types

- 1) Predefined packages
- 2) User defined packages

### Predefined packages:-

The java predefined packages are introduced by sun peoples these packages contains predefined classes and interfaces.

```
Ex:- java.lang  
      Java.io  
      Java.awt  
      Java.util  
      Java.net.....etc
```

### Java.lang:-

The most commonly required classes and interfaces to write a sample program is encapsulated into a separate package is called java.lang package.

```
`      Ex:- String(class)  
          StringBuffer(class)  
          Object(class)  
          Runnable(interface)  
          Cloneable(nterface)
```

### Note:-

the default package in the java programming is java.lang if we are importing or not importing by default this package is available for our programs.

### Java.io package:-

The classes which are used to perform the input output operations that are present in the java.io packages.

```
Ex:-      FileInputStream(class)
```



```

        FileOutputStream(class)
        FileWriter(class)
        FileReader(class)

```

**Java.net package:-**

The classes which are required for connection establishment in the network that classes are present in the java.net package.

Ex:- Socket  
       ServerSocket  
       InetAddress  
       URL

**Java.awt package:-**

The classes which are used to prepare graphical user interface those classes are present in the java.awt package.

Ex:     Button(class)  
          Checkbox(class)  
          Choice(Class)  
          List(class)

**User defined packages:-**

- 1) The packages which are declared by the user are called user defined packages.
- 2) In the single source file it is possible to take the only one package. If we are trying to take two packages at that situation the compiler raise a compilation error.
- 3) In the source file it is possible to take single package.
- 4) While taking package name we have to follow some coding standards.

Whenever we taking package name don't take the names like pack1, pack2, sandhya, sri..... these are not a proper coding formats.

**Rules to follow while taking package name:-(not mandatory but we have to follow)**

- 1) The package name must reflect with your organization name. the name is reverse of the organization domain name.

**Domain name:-**               www.dss.com  
**Package name:-**             Package com.dss;

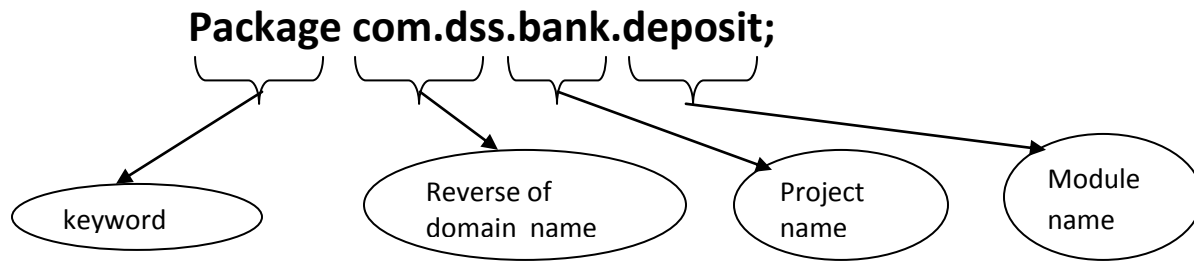
- 2) Whenever we are working in particular project(Bank) at that moment we have to take the package name is as follows.

**Project name :-**               Bank  
**package :-**                    Package com.dss.Bank;

- 3) The project contains the module (deposit) at that situation our package name should reflect with the module name also.

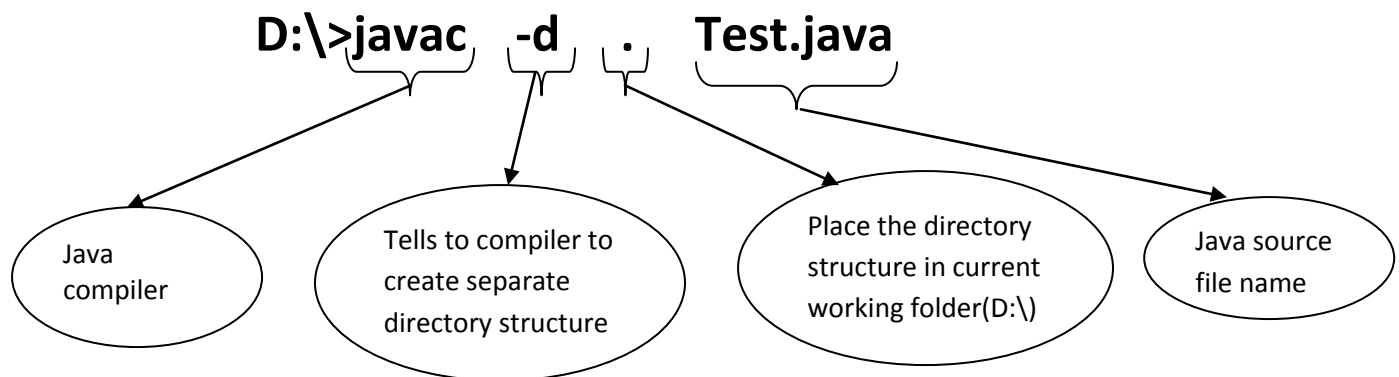
**Domain name:-**               www.dss.com  
**Project name:-**             Bank  
**Module name:-**            deposit  
**package name:-**            Package com.dss.bank.deposit;

For example the source file contains the package structure is like this:-

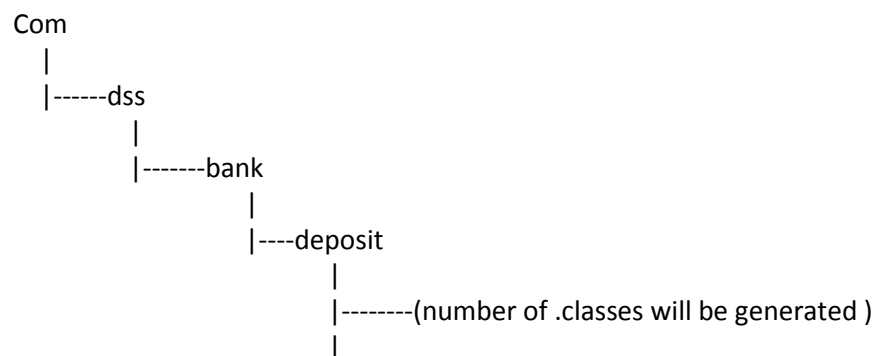


**Note:-**

If the source file contains the package statement then we have to compile that program with the help of following statements.



After compilation of the code the folder structure is as shown below.



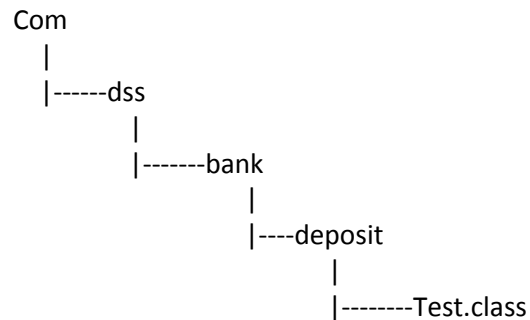
**Note :-**

If it a predefined package or user defined package the packages contains number of classes.

**Ex 1:-**

```
package com.dss.bank.deposit;
class Test
{
    public static void main(String[] args)
    {
        System.out.println("package example program");
    }
}
```

Compilation : javac -d . Test.java



Execution : java com.dss.bank.deposit.Test

#### Ex:- (compilation error)

```
package com.dss.bank.deposit;
package com.dss.online.corejava;
class Test
{
    public static void main(String[] args)
    {
        System.out.println("package example program");
    }
}
```

#### Reason:-

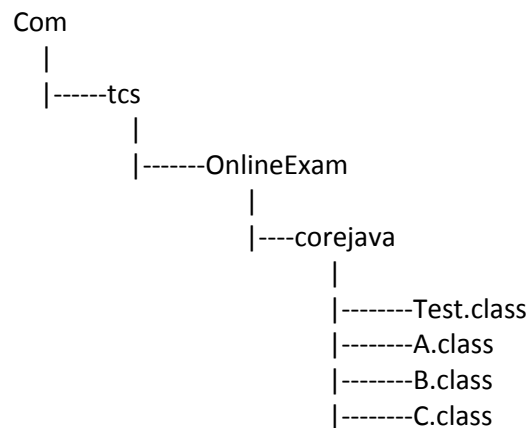
Inside the source file it is possible to take the single package not possible to take the multiple packages.

#### Ex 2:-

```
package com.tcs.OnlineExam.corejava;
class Test
{
    public static void main(String[] args)
    {
        System.out.println("package example program");
    }
}
```

```
}  
class A  
{  
}  
class B  
{  
}  
class C  
{  
}
```

Compilation                    :- javac -d . Test.java



Execution                    :- java com.tcs.onlineexam.Test

Note:-

The package contains any number of .classes the .class files generation totally depends upon the number of classes present on the source file.

#### Import session:-

The main purpose of the import session is to make available the java predefined support into our program.

#### Predefined packages support:-

Ex1:-

```
Import java.lang.String;
```

String is a predefined class to make available predefined string class to the our program we have to use import session.

Ex 2:-

```
Import java.awt.*;
```

To make available all predefined class present in the awt package into our program. That \* represent all the classes present in the awt package.

**User defined packages support:-**

I am taking two user defined packages are

- 1) Package pack1;  
    Class A  
    {  
    }  
    Class B  
    {  
    }
- 2) Package pack2  
    Class D  
    {  
    }

Ex 1:-

Import pack1.A;

A is a class present in the pack1 to make available that class to the our program we have to use import session.

Ex 2:-

Import pack1.\*;

By using above statement we are importing all the classes present in the pack1 into our program. Here \* represent the all the classes.

**Note:-**

**If it is a predefined package or user defined package whenever we are using that package classes into our program we must make available that package into our program with the help of import statement.**

**Public:-**

- This is the modifier applicable for classes, methods and variables (only for instance and static variables but not for local variables).
- If a class is declared with public modifier then we can access that class from anywhere (within the package and outside of the package).
- If we declare a member(variable) as a public then we can access that member from anywhere but Corresponding class should be visible i.e., before checking member visibility we have to check class visibility.

Ex:-

```
public class Test          // public class can access anywhere
{
    public int a=10; //public variable can access any where
    public void m1()      //public method can access any where
    {
        System.out.println("public method access in any package");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
        System.out.println(t.a);
    }
}
```

```
    }  
};
```

**Default:-**

- This is the modifier applicable for classes, methods and variables (only for instance and static variables but not for local variables).
- If a class is declared with <default> modifier then we can access that class only within that current package but not from outside of the package.
- Default access also known as a package level access.
- The default modifier in the java language is **default**.

**Ex:-**

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

**Note :-**

in the above program we are not providing any modifier for the methods and classes at that situation the default modifier is available for methods and classes that is default modifier. Hence we can access that methods and class with in the package.

**Private:-**

- private is a modifier applicable for methods and variables.
- If a member declared as private then we can access that member only from within the current class.
- If a method declare as a private we can access that method only within the class. it is not possible to call even in the child classes also.

```
class Test  
{  
    private void m1()  
    {  
        System.out.println("we can access this method only with in this class");  
    }  
    public static void main(String[] args)  
    {  
        Test t=new Test();  
    }  
}
```

```
        t.m1();
    }
};
```

**Protected :-**

- If a member declared as protected then we can access that member with in the current package anywhere but outside package only in child classes.
- But from outside package we can access protected members only by using child reference. If we try to use parent reference we will get compile time error.
- Members can be accesses only from instance area directly i.e., from static area we can't access instance members directly otherwise we will get compile time error.

**Ex:-demonstrate the user defined packages and user defined imports.****Durgasoft project source file:-**

```
package com.dss;
public class StatesDemo
{
    public void ap()
    {
        System.out.println("ANDHRA PRADESH");
    }
    public void tl()
    {
        System.out.println("TELENGANA");
    }
    public void tn()
    {
        System.out.println("TAMILNADU");
    }
}
```

**Tcs project source file:-**

```
package com.tcs;
import com.dss.StatesDemo;//or import com.dss.*;
public class StatesInfo
{
    public static void main(String[] args)
    {
        StatesDemo sd=new StatesDemo();
        sd.ap();
        sd.tl();
        sd.tn();
    }
}
```

Step 1 :- javac -d . StatesDemo.java

Step 2 :- javac -d . StatesInfo.java

Step 3 :- java com.tcs.StatesInfo

**Static import:-**

- 1) this concept is introduced in 1.5 version.
- 2) if we are using the static import it is possible to call static variables and static methods directly to the java programming.

**Ex:-without static import**

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

Ex:-package com.dss;

```
public class Test
{
    public static int a=100;
    public static void m1()
    {
        System.out.println("m1 method");
    }
};
```

**Ex :- with static import**

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

Ex:-

```
package com.tcs;
import static com.dss.Test.*;
class Test1
{
    public static void main(String[] args)
    {
        System.out.println(a);
        m1();
    }
}
```

**Source file Declaration rules:-**

The source file contains the following elements

- 1) Package declaration---→optional-----→at most one package(0 or 1)--→1<sup>st</sup> statement
  - 2) Import declaration-----→optional-----→any number of imports-----→2<sup>nd</sup> statement
  - 3) Class declaration-----→optional-----→any number of classes-----→3<sup>rd</sup> statement
  - 4) Interface declaration---→optional-----→any number of interfaces-----→3<sup>rd</sup> statement
  - 5) Comments declaration-→optional----→any number of comments----→3<sup>rd</sup> statement
- a. The package must be the first statement of the source file and it is possible to declare at most one package within the source file .
  - b. The import session must be in between the package and class statement. And it is possible to declare any number of import statements within the source file.
  - c. The class session is must be after package and import statement and it is possible to declare any number of class within the source file.
    - i. It is possible to declare at most one public class.
    - ii. It is possible to declare any number of non-public classes.
  - d. The package and import statements are applicable for all the classes present in the source file.
  - e. It is possible to declare comments at beginning and ending of any line of declaration it is possible to declare any number of comments within the source file.



**Preparation of userdefined API (application programming interface document):-**

1. API document nothing but user guide.
2. Whenever we are buying any product the manufacturing people provides one document called user guide. By using userguide we are able to use the product properly.
3. James gosling is developed java product whenever james gosling is delivered the project that person is providing one user document called API(application programming interface) document it contains the information about how to use the product.
4. To prepare userdefined api document for the userdefined projects we must provide the description by using documentation comments that information is visible in API document.
5. If we want to create api document for your source file at that situation your source file must contains all members(classes,methods,variables....) with modifiers.

```
package com.dss;
/*
 *parent class used to get parent values
 */
public class Test extends Object
{
    /** by using this method we are able get some boy*/
    public void marry(String ch)
    {
        System.out.println("xxx boy");
    }
};
```

Generated Documentation (Untitled) - Windows Internet Explorer

D:\index.html

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS  
SUMMARY: NESTED | FIELD | CONSTR | METHOD

FRAMES NO FRAMES All Classes  
DETAIL: FIELD | CONSTR | METHOD

com.dss

**Class Test**

java.lang.Object  
└ com.dss.Test

public class Test  
extends java.lang.Object

**Constructor Summary**

[Test\(\)](#)

**Method Summary**

void	<a href="#">marry</a> (java.lang.String ch)	by using this method we are able get some boy
------	---	---

**Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

**Constructor Detail**

**Test**

public Test()

## Java.lang.String

### String:-

- 1) String is a final class it is present in java.lang package.
- 2) String is nothing but a group of characters or character array.
- 3) Once we are creating String object it is not possible to do the modifications on existing object called immutability nature.
- 4) In String class .equals() is used for content comparison.

### Constructors of string class:-

- 1) String str=new String(java.lang.String);  
This constructor takes the String as a argument.

Ex:-

```
String str=new String("rattaiah");
System.out.println(str);//rattaiah
```

- 2) String str=new String(char[]);  
This constructor take the array of characters as a argument.

Ex:-

```
char[] ch={'a','b','c','d'};
String str1=new String(ch);
System.out.println(str1); //abcd
```

- 3) String str=new String(char[] ,int ,int );  
This constructor takes the array of characters with starting index and ending index. First int represent the starting position Second int represent the ending position.

Ex:-

```
char[] ch={'a','b','c','d'};
String str1=new String(ch,1,3);
System.out.println(str1); //bcd
```

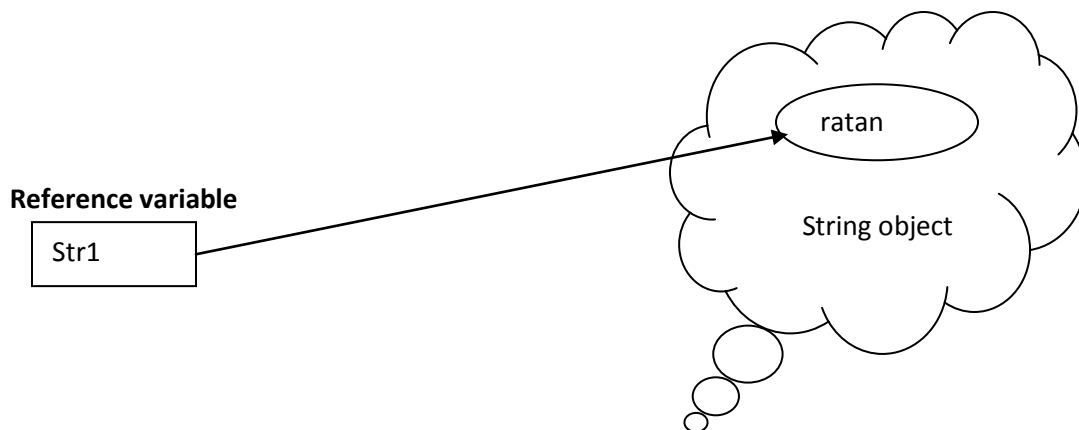
Ex:-

class Test

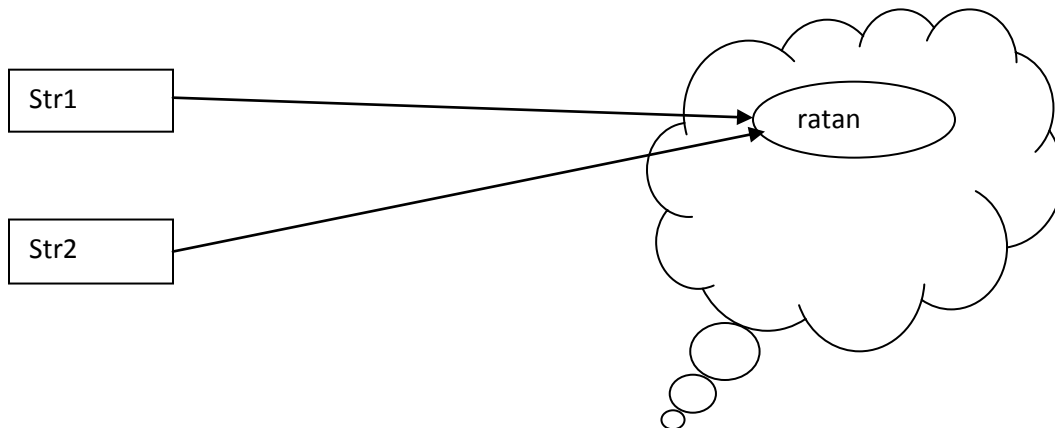
```
{
    public static void main(String[] args)
    {
        String str="ratan";
        System.out.println(str);

        String str1=new String("ratan");
        System.out.println(str1);
    }
}
```

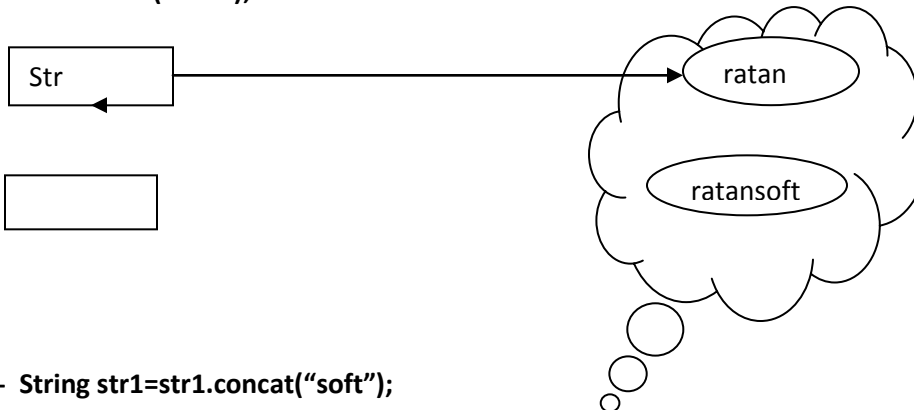
```
String str2=new String(str1);  
System.out.println(str2);  
  
char[] ch={'r','a','t','a','n'};  
String str3=new String(ch);  
System.out.println(str3);  
  
char[] ch1={'a','r','a','t','a','n','a'};  
String str4=new String(ch1,1,5);  
System.out.println(str4);  
  
byte[] b={65,66,67,68,69,70};  
String str5=new String(b);  
System.out.println(str5);  
  
byte[] b1={65,66,67,68,69,70};  
String str6=new String(b1,2,4);  
System.out.println(str6);  
}  
}
```

**String objects and their reference variables:-****Note 1:-**            **String str1="ratan" ;**

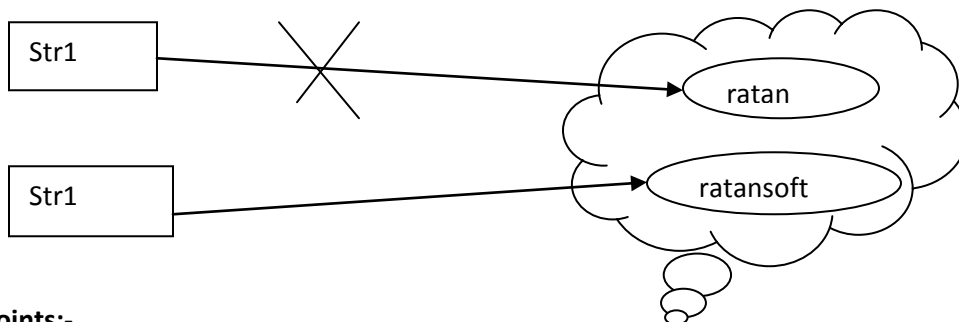
**Note 2:-**                `String str2=str1 ;`



**Note 3:-** `String str="ratan";`  
`Str.concat("soft");`



**Note 4:-** `String str1=str1.concat("soft");`



**Good points:-**

Without using new operator it is possible to create a object for the classess. such type of classes objects we can called it as **first-class objects** .

Ex:-String

**Creation of String object:-**

To create a object for string class we are having two approaches

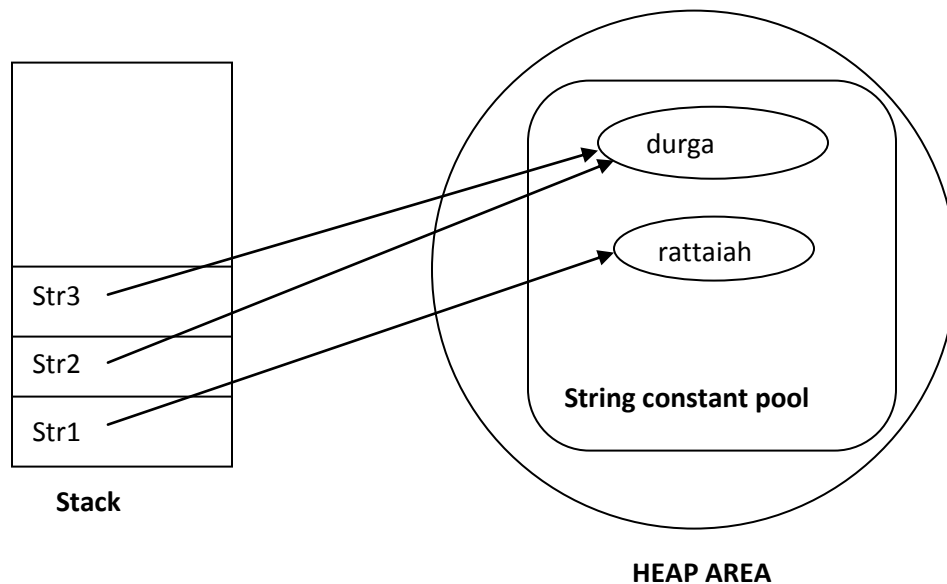
- 1) without using new operator(by using literal)
- 2) by using new operator

**Creating a string object without using new operator(by using literal):-**

```
String str1="rattaiah" ;  
String str2="durga";
```

```
String str3="durga";
```

Whenever we create string literal first jvm goes to SCP(String constant pool ) and check if the string is already present in the pool or not. If it is available it returns the existing reference from the pool , if it is not available a new String object is created



**In the above example two two objects are available**

First time JVM will not find any string object with the name "rattaiah" so JVM creates a new object.

Second time JVM will not find any String object with the name "durga" so JVM is creates the new object one more time.

Third time JVM is finding the string object with the content "durga" at this time JVM wont creates any new object just JVM is return the reference to the same instance.

**Advantage of above approach:-**

If we are using above approach memory management is very good because duplicate content objects are not presented in the String constant pool area. The objects which is present in the SCP area are unique objects.

**Good points :-**

Each and every object is stored in heap area. The heap area is divided into different pools.

1) Referenced pool

In the referenced pool all the objects references maintained.

2) Object pool

In this pool all the objects are located here by using new operator

3) Thread pool

In java we are able to create a thread object which not contains any lock

4) String pool

Without using new operator we are creating objects such type of the objects are located here(ex:-String object).

**Creating a string object by using new operator:-**

```
String str=new String("rattaiah");
```

JVM is creates a object and placed into the normal heap area and literal is placed into the string constant pool. So if we are using above approach the object is created in the two areas heap area String constant pool area.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str1=new String("rattaiah");
        str1.concat("addanki");
        String str2=str1.concat("aruna");
        str2.concat("nandhu");
        System.out.println(str1);
        System.out.println(str2);
    }
};
```

**String is immutability nature:-**

Once we are creating string object it is not possible to do the modifications on the existing object is called immutability nature.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str="rattaiah";
        str.concat("addanki");
        System.out.println(str);
    }
};
class Test
{
    public static void main(String[] args)
    {
        String str1="ratan";
        str1.concat("soft");
        String str2=str1.concat("soft");;
        String str3=str2;
        String str4="ratan".concat("durga");
        String str5=str4;
        System.out.println(str2==str3);
        System.out.println(str1==str2);
        System.out.println(str1==str3);
    }
};
```

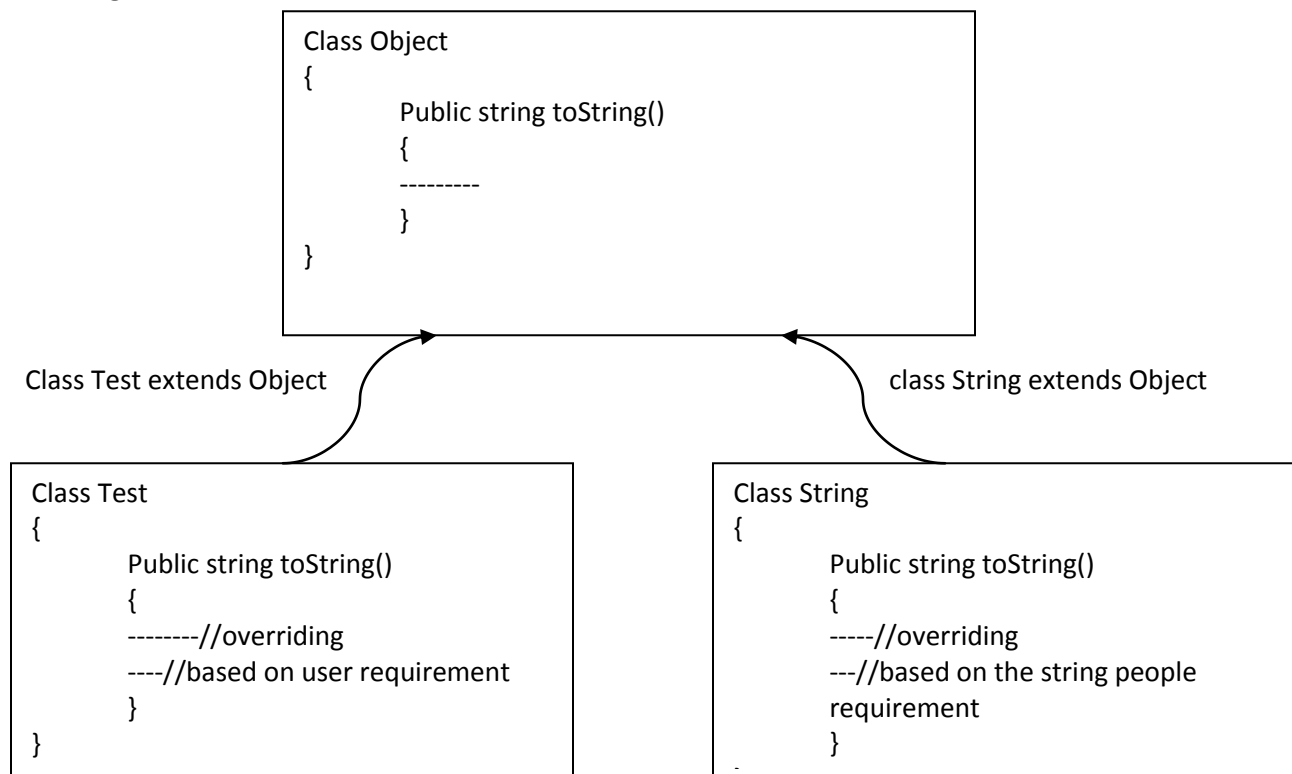
```

        System.out.println(str2==str4);
        System.out.println(str5==str4);
    }
}
class Test
{
    public static void main(String[] args)
    {
        String str1=new String("ratan");
        String str2=new String("durga");
        String str3=new String("ratan");
        String str4=new String("ratna");
        System.out.println(str1.compareTo(str2));//+ve
        System.out.println(str1.compareTo(str3));//0
        System.out.println(str1.compareTo(str4));// -ve
        System.out.println(str4.compareTo(str2));//+ve
        System.out.println(str1.compareToIgnoreCase("Ratan"));//0
        System.out.println("durgasoft".compareTo(str2));//+ve
    }
}

```

### String class Methods:-

#### toString:-



**object class toString()method is executing .**

```
class Object
{
    public String toString()
    {
        //it is returns getClass@hashCode
    }
};
class String extends Object
{
    public String toString()
    {
        // it is returns the String content      overriding toString()
    }
};
```

If we are printing the reference variable the string representation of object will be printed. The reference variable internally calling the toString() method automatically. In the below case Object class toString() got executed.

```
class Student
{
    String sname;
    int rollno;
    Student(String sname,int rollno)
    {
        this.sname=sname;
        this.rollno=rollno;
    }
    public static void main(String[] args)
    {
        Student s=new Student("rattaiah",100);
        System.out.println(s);//equal to //System.out.println(s.toString());
    }
};
```

```
class Test
{
    public static void main(String[] args)
    {
        String str1="ratan";
        str1.concat("soft");
        System.out.println(str1);//ratan
        String str2="ratan";
        System.out.println(str1.equals(str2));//String class .equals()

        StringBuffer sb=new StringBuffer("durga");
        sb.append("soft");
        System.out.println(sb);//durgasoft
    }
};
```



```
        StringBuffer sb1=new StringBuffer("durga");
        StringBuffer sb2=new StringBuffer("durga");
        StringBuffer sb3=sb2;
System.out.println(sb1.equals(sb2));//false//Object class .equals()
System.out.println(sb3.equals(sb2));//true//Object class .equals()
    }
}
class Test
{
    public String toString()
    {
        return "ratan";
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        System.out.println(t);
        System.out.println(t.toString());//Test class toString()

        String str="durgasoft";
        System.out.println(str);
System.out.println(str.toString());//String class toString()

        StringBuffer sb=new StringBuffer("ratansoft");
        System.out.println(sb);
System.out.println(sb.toString());//StringBuffer class toString()
    }
}
```

**User defined toString() is got executed:-**

```
class Student
{
    String sname;
    int rollno;
    Student(String sname,int rollno)
    {
        this.sname=sname;
        this.rollno=rollno;
    }
    //overriding of the toString method
    public String toString()
    {
        return sname+"***"+rollno;
    }
    public static void main(String[] args)
    {
        Student s=new Student("rattaiah",100);
        System.out.println(s);
    }
}
```

```
    }  
};
```

**String class toString() method is executed:-**

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

**+ operator(for String concatenation):-**

'+' is a overloaded operator in the java language the + is acting as a addition of the two numbers and the '+' is acting as a concatenation of the two strings.

if both operands are numbers the + is acting as a addition of the two numbers .

if at least one operand is String then + is acting as a concatenation of the String.

Ex:-

```
class Test  
{  
    public static void main(String[] args)  
    {  
        String str1="durga";  
        String str2=40+50+str1+"ratan"+60+70;  
        System.out.println(str2);  
    }  
}
```

**Concat():-**

Concat() method present in the String class and it is used to combine the two String.

Ex :-

```
class Test  
{  
    public static void main(String[] args)  
    {  
        String age="22";  
        String s="he is "+age+" years old.";  
        System.out.println(s);  
  
        String str1="durga";  
        String str2="ratan";  
        String str3=str1.concat(str2);  
        System.out.println(str3);  
  
        int a=22;
```

```
String s1="he is "+a+" years old.";
System.out.println(s1);
```

```
String s2="six "+2+2+2;
System.out.println(s2);
```

```
String s3="six "+(2+2+2);
System.out.println(s3);
```

```
}
```

**Length():-**

It is used to find out the length of the string

**Difference between length() method and length variable:-**

length()----→method

length--→variable

length is available in array to find the length of the array

Ex:-

```
int [] a={10,20,30};
System.out.println(a.length);//3
```

length() is method used to find the length of the given string.

Ex:-

```
String str="rattaiah";
System.out.println(str.length());//8
```

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        String str="ratan";
        System.out.println(str.length());
        System.out.println("ratansoft".length());
        int[] a={10,20,30};
        System.out.println(a.length);
    }
}
```

**charAt(int):-**

By using above method we are able to extract the character from particular index position.

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        String str="ratan";
        System.out.println(str.charAt(1));

        char ch="ratan".charAt(2);
        System.out.println(ch);
    }
}
```

```
}
```

**Split(String):-**

By using split() method we are dividing string into number of tokens.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str="hi rattaiah how r u";
        String[] str1=str.split(" ");
        for (int i=0;i<str1.length ;i++ )
        {
            System.out.println(str1[i]);
        }
    }
};
```

**CompareTo(string anotherstring) and compareToIgnoreCase():-**

- 1) By using compareTo() we are comparing two strings character by character such type of checking is called lexicographically checking or dictionary checking.
- 2) compareTo() is return three values as outcome
  - a. zero (if both are equal)
  - b. positive (first string first character is having big character compare to second string )
  - c. negative(first string first character small character compare to second String )
- 3) compareTo() method comparing two string with case sensitive.
- 4) By using above method we are comparing two strings character by character by ignoring case.

xEx:

```
class Test
{
    public static void main(String... ratan)
    {
        String str1="ratan";
        String str2="durga";
        String str3="ratan";
        String str4="Durga";
        String str5="ratna";
        System.out.println(str1.compareTo(str2));//14
        System.out.println(str1.compareTo(str4));//46
        System.out.println(str2.compareTo(str4));//32
        System.out.println(str4.compareTo(str2));//-32
        System.out.println(str1.compareTo(str3));//0
        System.out.println(str1.compareTo(str5));//-13
        System.out.println(str1.compareToIgnoreCase(str5));//-13
        System.out.println(str2.compareToIgnoreCase(str4));//0
        System.out.println(str1.compareToIgnoreCase(str3));//0
    }
}
```

```

    }
};

```

### public boolean equals():-

- String class equals() method is used for content comparison . It returns true or false value after comparison.
- The commonly used method to perform comparison.
- We are comparing total content called deep comparison.
- At the time of comparison the following possibilities occur.
  - True----->two Strings are same
  - False----->two strings are not equals

### Implementation:-

```

class Object
{
    .equals()
    {
        //used for reference comparison
    }
};
class String extends Object
{
    .equals()
    {
        //used for content comparison
    }
};
class StringBuffer extends Object
{
    .equals()
    {
        //used for reference comparison
    }
};
Ex:-
class Test
{
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test();
        Test t3=t2;
        System.out.println(t1.equals(t2));//false
        System.out.println(t1.equals(t3));//false
        System.out.println(t3.equals(t2));//true
    }
}

```

} String persons are overriding  
for content comparison

} not overriding hence .equals() used for  
reference comparison

```

        String str1="ratan";
        String str2="durga";
        String str3="ratan";
        System.out.println(str1.equals(str2));//false
        System.out.println(str1.equals(str3));//true
        System.out.println("durga".equals(str2));//true
        System.out.println("SOFT".equals("soft"));//false
        System.out.println("soft".equals("soft"));//true
    }
}
class Test
{
    public static void main(String[] args)
    {
        String str1="ratan";
        String str2="durga";
        String str3="Ratan";
        System.out.println(str1.equalsIgnoreCase(str2));//false
        System.out.println(str1.equalsIgnoreCase(str3));//true
        System.out.println("durga".equalsIgnoreCase(str2));//true
        System.out.println("SOFT".equalsIgnoreCase("soft"));//true
        System.out.println("soft".equalsIgnoreCase("soft"));//true
    }
}

```

**== operator:-**

It is used for reference comparison. Hence we can call shallow comparison.

Ex:-

```

class Test
{
    public static void main(String[] args)
    {
        String s1=new String("rattaiah");
        String s2=new String("duragsoft");
        String s3=new String("rattaiah");
        String s4=s1;
        System.out.println(s1.equals(s2));
        System.out.println(s1.equals(s3));
        System.out.println(s1==s3);
        System.out.println(s1==s4);
    }
};

```

Ex:-

```

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

```

```
{
    String str1=new String("durga");
    String str2=new String("durga");
    System.out.println(str1==str2);//false

    String str3="durga";
    String str4="durga";
    System.out.println(str3==str4);//false
}
```

Ex :-

class Test

```
{
    Test(int i)
    {
        System.out.println("0 arg cons");
        System.out.println(i);
    }
    public static void main(String[] args)
    {
        Test t1=new Test(10);
        Test t2=new Test(10);
        Test t3=new Test(10);
        Test t4=t1;
        System.out.println(t1==t2);//false
        System.out.println(t2==t3);//false
        System.out.println(t3==t1);//false
        System.out.println(t1==t4);//true
    }
}
```

### **Public Boolean equalsIgnoreCase():-**

By using above method we are comparing the strings

Ex:-

class Test

```
{
    public static void main(String... ratan)
    {
        String str1="ratan";
        String str2="durga";
        String str3="ratan";
        String str4="Durga";
        System.out.println(str1.equalsIgnoreCase(str2));//false
        System.out.println(str1.equalsIgnoreCase(str4));//false
        System.out.println(str2.equals(str4));//false
        System.out.println(str2.equalsIgnoreCase(str4));//true
    }
}
```

**};getBytes():-**

By using this method we are converting String into the byte[] .the main aim of the converting String into the byte[] format is some of the networks are supporting to transfer the data in the form of bytes only at that situation is conversion is mandatory.

Ex:-

```
class Test
{public static void main(String[] args)
    {String str="rattaiah";
      byte[] b=str.getBytes();
      System.out.println(b);
      String str1=new String(b);
      System.out.println(str1);
    }
}
```

**trim():-**

- 1) trim() is used to remove the trail and leading spaces
- 2) this method always used for memory saver

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str="  ratan  ";
        System.out.println(str.length());//7
        System.out.println(str.trim());//ratan
        System.out.println(str.trim().length());//5
        System.out.println("  ratan  ".trim());//ratan
    }
}
```

**replace(char oldchar,char newchar) replace(String oldString,String newString):-**

by using above method we are replacing the particular character of the String. And particular portion of the string.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str="rattaiah how r u";
        System.out.println(str.replace('a','A'));//rAttAiAh
        System.out.println(str.replace("how","who"));//rattaiah how r u
    }
}
```

Ex :-

```
class Test
{
    public static void main(String[] args)
    {
        String str1="durga software solutions";
```



```
        System.out.println(str1);
        System.out.println(str1.replace("software","hardware")); // Durga hardware solutions
    }
}
```

**toUpperCase() and toLowerCase():-**

The above methods are used to convert the lower case to the uppercase and uppercase to lowercase character.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        String str="ratan HOW R U";
        System.out.println(str.toUpperCase());
        System.out.println(str.toLowerCase());
        System.out.println("RATAN".toLowerCase());
        System.out.println("soft".toUpperCase());
    }
}
```

**Java.lang.String.endsWith() and Java.lang.String.startsWith():-**

endsWith() is used to find out if the string is ending with particular character/string or not.

startsWith() used to find out the particular String starting with particular character/string or not.

```
class Test
{
    public static void main(String[] args)
    {
        String str="rattaiah how r u";
        System.out.println(str.endsWith("u")); //true
        System.out.println(str.endsWith("how")); //false
        System.out.println(str.startsWith("d")); //false
        System.out.println(str.startsWith("r")); //true
    }
}
```

**substring(int startingposition) & substring(int startingposition,int endingposition):-**

By using above method we are getting substring from the whole String.

In the above methods

starting position parameter value is including

ending position parameter value is excluding

```
class Test
{
    public static void main(String[] args)
    {
        String str="ratan how r u";
        System.out.println(str.substring(2)); //tan how r u
        System.out.println(str.substring(1,7)); //atan h
        System.out.println("ratansoft".substring(2,5)); //tan
    }
}
```

**StringBuffer:-**

- 1) String Buffer is a class present in the java.lang package.
- 2) StringBuffer is a final class so it can't be inherited.
- 3) StringBuffer is a mutable class so it is possible to change the content in the same location.
- 4) StringBuffer .equals() method is used for reference comparison.

**Constructors:-**

- 1) StringBuffer sb=new StringBuffer();
- 2) StringBuffer sb1=new StringBuffer(int capacity);
- 3) StringBuffer sb2=new StringBuffer(String str);

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer();
        System.out.println(sb.capacity());//default capacity 16

        StringBuffer sb1=new StringBuffer(5);
        System.out.println(sb1.capacity());//your provided capacity

        StringBuffer sb2=new StringBuffer("rattaiah");
        System.out.println(sb2.capacity());//initial capacity+provided string length 24
        System.out.println(sb2.length()); //8
    }
}
```

**StringBuffer is mutable:-**

Once we are creating a StringBuffer Object it is possible to the modification on existing object is called mutability nature.

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        StringBuffer s1=new StringBuffer("rattaiah");
        s1.append("addanki");//mutability
        System.out.println(s1);
        StringBuffer s2=new StringBuffer("durgasoft");
        StringBuffer s3=s1;
        System.out.println(s1.equals(s2));
        System.out.println(s1.equals(s3));
        System.out.println(s1==s2);
        System.out.println(s1==s3);
    }
};
class Test
{
```

```
public static void main(String[] args)
{
    Test t1=new Test();
    Test t2=t1;
    Test t3=new Test();
    System.out.println(t1.equals(t2));//true
    System.out.println(t1.equals(t3));//false

    String str1="ratan";
    String str2="ratan";
    System.out.println(str1.equals(str2));//true

    StringBuffer sb1=new StringBuffer("durga");
    StringBuffer sb2=new StringBuffer("durga");
    StringBuffer sb3=sb2;
    System.out.println(sb1.equals(sb2));//false
    System.out.println(sb2.equals(sb3));//true
}
}
```

Note 1:- it is possible to change the content of StringBuffer Object

Note2:- StringBuffer .equals() is used for reference comparison(address comparison)

Note 3:- == operator is used for reference comparison (address comparison)

#### **reverse():-**

```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("rattaiah");
        System.out.println(sb);
        System.out.println(sb.delete(1,3));
        System.out.println(sb);
        System.out.println(sb.deleteCharAt(1));
        System.out.println(sb.reverse());
    }
}
```

#### **Append():-**

By using this method we can append the any values at the end of the string

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("rattaiah");
        String str=" salary ";
        int a=30000;
        sb.append(str);
        sb.append(a);
        System.out.println(sb);
    }
}
```

```
    }  
};
```

**Insert():-**

By using above method we are able to insert the string any location of the existing string.

Ex:-

```
class Test  
{  
    public static void main(String[] args)  
    {  
        StringBuffer sb=new StringBuffer("ratan");  
        sb.insert(0,"hi ");  
        System.out.println(sb);  
    }  
}
```

**indexOf() and lastIndexOf():-**

Ex:-

```
class Test  
{  
    public static void main(String[] args)  
    {  
        StringBuffer sb=new StringBuffer("hi ratan hi");  
        int i;  
        i=sb.indexOf("hi");  
        System.out.println(i);  
        i=sb.lastIndexOf("hi");  
        System.out.println(i);  
    }  
}
```

**replace():-**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        StringBuffer sb=new StringBuffer("hi ratan hi");  
        sb.replace(0,2,"oy");  
        System.out.println("after replaceing the string:-"+sb);  
    }  
}  
class Test  
{  
    public static void main(String[] args)  
    {  
        StringBuffer sb1=new StringBuffer("ratansoft");  
        StringBuffer sb2=new StringBuffer("ratan");  
        StringBuffer sb3=new StringBuffer("ratan");  
        StringBuffer sb4=sb3;  
        System.out.println(sb1.equals(sb2));//false  
        System.out.println(sb1.equals(sb3));//flase
```

```
        System.out.println(sb1.equals(sb4));//false
        System.out.println(sb2.equals(sb3));//false
        System.out.println(sb3.equals(sb4));//true
        System.out.println(sb1==sb2);//false
        System.out.println(sb3==sb4);//true
        System.out.println(sb2==sb3);//false
    }
}
```

**Java.lang.StringBuilder:-**

- 1) Introduced in jdk1.5 version.
- 2) StringBuilder is identical to StringBuffer except for one important difference.
- 3) Every method present in the StringBuilder is not Synchronized means that is not thread safe.
- 4) multiple threads are allow to operate on StringBuilder methods hence the performance of the application is increased.

**Cloneable:-**

- 1) The process of creating exactly duplicate object is called cloning.
- 2) We can create a duplicate object only for the cloneable classes .
- 3) We can create cloned object by using clone()
- 4) The main purpose of the cloning is to maintain backup.

```
class Test implements Cloneable
{
    int a=10;
    int b=20;
    public static void main(String[] args)throws CloneNotSupportedException
    {
        Test t1=new Test();
        Test t2=(Test)t1.clone();
        t1.a=100;
        t1.b=200;
        System.out.println(t1.a+"----"+t1.b);
        System.out.println(t2.a+"----"+t2.b);
    }
};
```

## Wrapper classes

- 1) To represent primitive data types as a Object form we required some classes these classes are called wrapper classes.
- 2) All wrapper classes present in the java.lang package.
- 3) Int,byte.... Acts as a primitives we can make the primitives into the objects is called wrapper classes the wrapper classes are Integer,Short-----.
- 4) We are having 8 primitive data types hence sun peoples are providing 8 wrapper classes.

### 5) Data types and corresponding wrapper classes:-

byte	----	Byte
short	----	Short
int	----	Integer
long	----	Long
float	----	Float
double	----	Double
boolean	----	Boolean
char	-----	Character

- 6) Byte,Short,Integer,Long,Float,Double these are child classes of Number class.

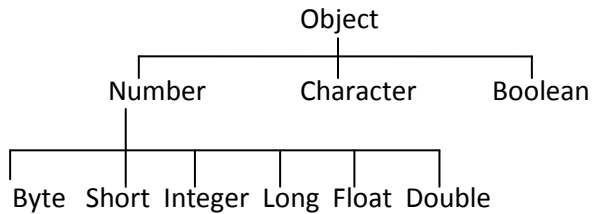
### Constructors of wrapper classes:-

All most all wrapper classes contain two constructors:-

1. Integer i=new Integer(10);
2. Integer i=new Integer("10");

Primitives	Wrapper classes	Fallowing constructor arguments
byte	Byte	Byte or String
short	Short	Short or String
int	Integer	Int or String
long	Long	Long or String
float	Float	Float or double or String
double	Double	double or String
char	Character	Char
Boolean	Boolean	Boolean or String

Wrapper classes hierarchy:-



Byte is taking two arguments:-

Maximum wrapper class taking String as a argument at that situation frequently we are getting NumberFormatException.

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        float f=10.7f;
        Float f1=new Float(f);
        System.out.println(f1);

        Float f2=new Float("10");
        System.out.println(f2);

        Float f3=new Float("ten");
        System.out.println(f3);//NumberFormatException

        Integer i1=new Integer(2);
        System.out.println(i1);
        Integer i2=new Integer("two");
        System.out.println(i2);//NumberFormatException
    }
}

class Test
{
    public static void main(String[] args)
    {
        int a=10;//primitive variable
        System.out.println(a);
        System.out.println(a.toString());//CE:-int cant be dereferenced
        Integer i=new Integer("100");//reference variable
        System.out.println(i);
        System.out.println(i.toString());
    }
}
```

**The main importance of wrapper classes:-**

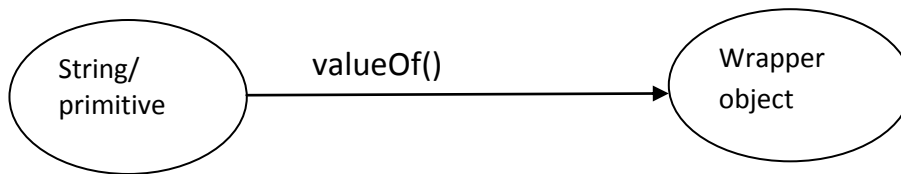
1. To convert a data types into the object means we are giving object from data types by using constructor.
2. To convert String into the data types by using parsexxx() method

**Utility methods:-**

1. valueOf()
2. xxxValue()
3. parsexxx()
4. toString()

**1) valueOf():-**

By using valueOf() we are creating wrapper object and it is a alternative to the constructor.



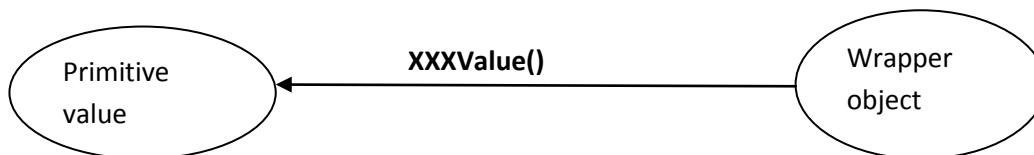
Ex :-

```

class Test
{
    public static void main(String[] args)
    {
        //by using constructor converting String/primitive to wrapper object
        Integer i=new Integer(10);
        System.out.println(i);
        //by using valueOf() converting String/primitive to the wrapper object
        Boolean b=Boolean.valueOf("true");
        System.out.println(b);
    }
}
  
```

**XxxValue():-**

by using XXXValue() method we are converting wrapper objects into the corresponding primitive values.



Ex :-

```

class Test
{
    public static void main(String[] args)
    {
        Integer i=Integer.valueOf(150);
        System.out.println("byte value :"+i.byteValue());//-106
        System.out.println("short value :"+i.shortValue());//150
        System.out.println("int value :"+i.intValue());//150
        System.out.println("long value :"+i.longValue());//150
        System.out.println("float value :"+i.floatValue());//150.0
    }
}
  
```



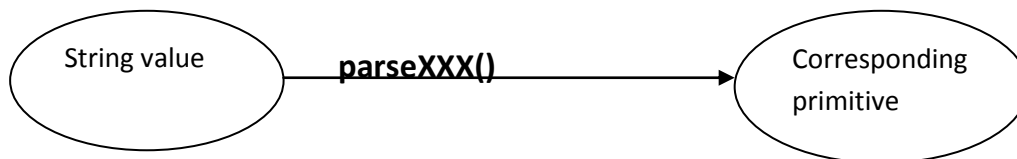
```

        System.out.println("double value      :"+i.doubleValue()); //150.0
        Character c=new Character('s');
        char ch=c.charValue();
        System.out.println(ch);
        Boolean b=new Boolean(false);
        boolean bb=b.booleanValue();
        System.out.println(bb);
    }
}

```

**parseXXX():-**

by using above method we are converting String into the corresponding primitive.



ex :-

```

class Test
{
    public static void main(String[] args)
    {
        String str1="10";
        String str2="20";
        System.out.println(str1+str2); //1020
        int a=Integer.parseInt(str1);
        float f=Float.parseFloat(str2);
        System.out.println(a+f); //30.0
    }
}

```

**toString():-**

```

class Object
{
    Public string toString()
    {
        return getClass@hashCode;
    }
};
class WrapperClasses extends Object
{
    //overriding the toString()
    Public string toString()
    {
        return the corresponding value
    }
};

```

Ex :-

```

class Test
{

```

```

public static void main(String[] args)
{
    Integer i=new Integer(10);
    System.out.println(i.toString());
    Float f=new Float(10.7);
    System.out.println(f.toString());
}
};

```

### **Autoboxing and Autounboxing:-(introduced in the 1.5 version)**

Until 1.4 version we are not allowed to place primitive in the palc wrapper and wrapper in the place of primitive. The programmer is responsible person to do the explicit conversion primitive to the wrapper and wrapper to the primitive.

#### **Autoboxing:-**

```

Integer i=10;
System.out.println(i);

```

The above statement does not work on the 1.4 and below versions. The auto conversion of the primitive into the Wrapper object is called the autoboxing these conversions done by compiler at the time of compilation.

#### **Autounboxing:-**

```

int a=new Integer(100);
System.out.println(a);

```

The auto conversion of the wrapper object to the primitive value is called autounboxing and these conversions are done by compiler at the time of compilation.

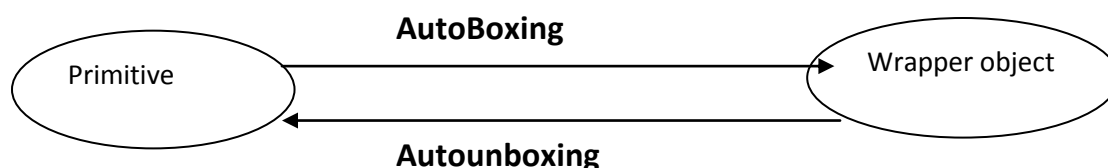
Ex :-

```

class Test
{
    static Integer i=10;//i is wrapper object
    static int j;//j is primitive variable
    static void print(int i)
    {
        j=i;
        System.out.println(j);
    }
    public static void main(String[] args)
    {
        print(i);
        System.out.println(j);
    }
}
};

```

### **Automatic conversion of the primitive to wrapper and wrapper to the primitive:-**



## Java .io package

Java.io is a package which contains number of classes by using that classes we are able to send the data from one place to another place.

In java language we are transferring the data in the form of two ways:-

1. Byte format
2. Character format

### Stream/channel:-

It is acting as medium by using steam or channel we are able to send particular data from one place to the another place.

Streams are two types:-

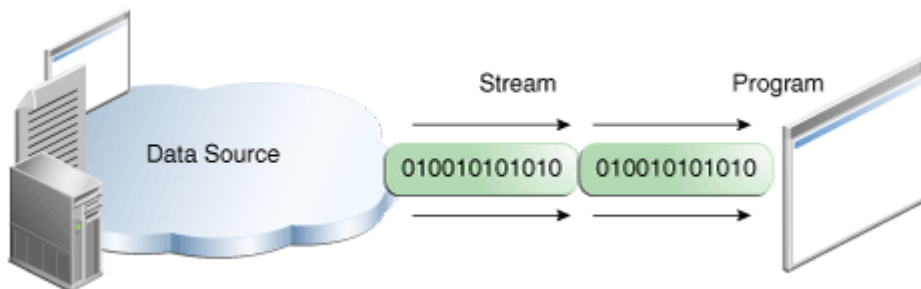
1. Byte oriented streams.(supports byte formatted data to transfer)
2. Character oriented stream.(supports character formatted data to transfer)

### Byte oriented streams:-

#### Java.io.FileInputStream

To read the data from the destination file to the java application we have to use FileInputStream class.

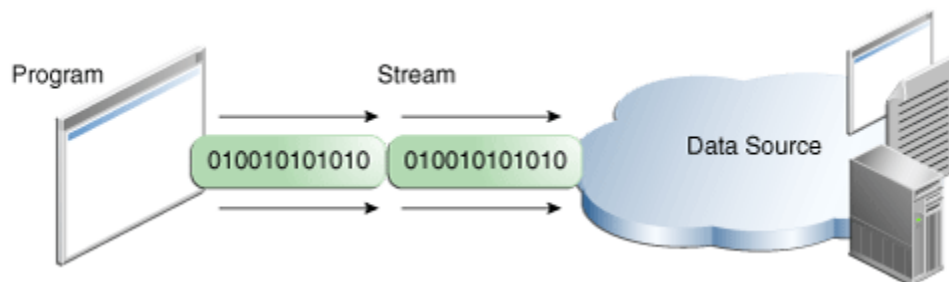
To read the data from the .txt file we have to read() method.



### Java.io.FileOutputStream:-

To write the data to the destination file we have to use the FileOutputStream.

To write the data to the destination file we have to use write() method.



Ex:- it will supports one character at a time.

```
import java.io.*;
class Test
{
    static FileInputStream fis;
    static FileOutputStream fos;
    public static void main(String[] args)
    {
        try{
            fis=new FileInputStream("get.txt");
            fos=new FileOutputStream("set.txt",true);
            int c;
            while ((c=fis.read())!=-1)
            {
                fos.write(c);
            }
            fis.close();
            fos.close();
        }
        catch(IOException io)
        {
            System.out.println("getting IOException");
        }
    }
}
```

Ex:-it will support one character at a time.

```
import java.io.*;
class Test
{
    static FileReader fr;
    static FileWriter fw;
    public static void main(String[] args)
    {
        try{
            fr=new FileReader("get.txt");
            fw=new FileWriter("set.txt",true);

            int c;
            while ((c=fr.read())!=-1)
            {
                fw.write(c);
            }
            fr.close();
            fw.close();
        }
        catch(IOException io)
        {
            System.out.println("getting IOException");
        }
    }
}
```

```
    }  
    }  
}
```

### Line oriented I/O:-

Character oriented streams supports single character and line oriented streams supports single line data.

**BufferedReader**:- to read the data line by line format and we have to use `readLine()` to read the data.

**PrintWriter** :- to write the data line by line format and we have to use `println()` to write the data.

```
import java.io.*;  
class Test  
{  
    static BufferedReader br;  
    static PrintWriter pw;  
    public static void main(String[] args)  
    {  
        try{  
            br=new BufferedReader(new FileReader("get.txt"));  
            pw=new      PrintWriter(new FileWriter("set.txt"));  
  
            String line;  
            while ((line=br.readLine())!=null)  
            {  
                pw.println(line);  
            }  
            br.close();  
            pw.close();  
        }  
        catch(IOException io)  
        {  
            System.out.println("getting IOException");  
        }  
    }  
}
```

### Buffered Streams:-

Up to we are working with non buffered streams these are providing less performance because these are interact with the hard disk, network.

Now we have to work with Buffered Streams

BufferedInputStream read the data from memory area known as Buffer.

We are having four buffered Stream classes

1. BufferedInputStream

2. BufferedOutputStream
3. BufferedReader
4. BufferedWriter

Ex:-

```
import java.io.*;
class Test
{
    static BufferedReader br;
    static BufferedWriter bw;
    public static void main(String[] args)
    {
        try{
            br=new BufferedReader(new FileReader("Test1.java"));
            bw=new BufferedWriter(new FileWriter("States.java"));
            String str;
            while ((str=br.readLine())!=null)
            {
                bw.write(str);
            }
            br.close();
            bw.close();
        }
        catch(Exception e)
        {
            System.out.println("getting Exception");
        }
    }
}
```

Ex:-

```
import java.io.*;
class Test
{
    static BufferedInputStream bis;
    static BufferedOutputStream bos;
    public static void main(String[] args)
    {
        try{
            bis=new BufferedInputStream(new FileInputStream("abc.txt"));
            bos=new BufferedOutputStream(new FileOutputStream("xyz.txt"));
            int str;
            while ((str=bis.read())!=-1)
            {
                bos.write(str);
            }
            bis.close();
        }
    }
}
```

```
        bos.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
        System.out.println("getting Exception");
    }
}
}
```

**Ex:-**

```
import java.io.*;
class Test
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new FileReader("abc.txt"));
        String str;
        while ((str=br.readLine())!=null)
        {
            System.out.println(str);
        }
    }
}
```

**Java.util.Scanner:-**

By using Scanner class we are able to divide the String into the number of tokens.

To get the integer value from the keyboard-----:s.nextInt()

To get the String value from the keyboard-----:s.next()

To get the floating values from the keyboard-----:s.nextFloat ();

```
import java.util.*;
```

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

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

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

        System.out.println("emp no----->" +eno);
        System.out.println("emp name----->" +ename);
        System.out.println("emp sal----->" +esal);

        System.out.println("do u want one more record(yes/no)");
        String option=s.next();
        if (option.equals("no"))
        {
            break;
        }
    }
}
```

Note :-

hasNext() method return true if the Scanner has another token as its input the return type is boolean.  
next() is return next complete token from the Scanner

Ex:-

```
import java.io.*;
import java.util.*;
class Test
{
    public static void main(String[] args)
    {

        try{
            Scanner s=new Scanner(new BufferedReader(new FileReader("abc.txt")));
            while (s.hasNext())
            {
                System.out.println(s.next());
            }
        }
        catch(Exception e)
        {
            System.out.println(e);
            System.out.println("getting Exception");
        }
    }
}
```



**Serialization:-**

The process of saving an object to a file (or) the process of sending an object across the network is called serialization.

But strictly speaking the process of converting the object from java supported form to the network supported form of file supported form.

To do the serialization we required following classes

1. FileOutputStream
2. ObjectOutputStream

**Deserialization:-**

The process of reading the object from file supported form or network supported form to the java supported form is called deserialization.

We can achieve the deserialization by using following classes.

1. FileInputStream
2. ObjectInputStream

Ex:-Student.java

```
import java.io.Serializable;
public class Student implements Serializable
{
    int id;
    String name;
    int marks;

    public Student(int id, String name,int marks)
    {
        this.id = id;
        this.name = name;
        this.marks=marks;
    }
}
```

**To perform serialization :- we are writing the object data to the file called abc.txt file we are transferring that file across the network.**

```
import java.io.*;
class Serializable1
{
    public static void main(String args[])throws Exception
    {
        Student s1 =new Student(211,"ravi",100);
        FileOutputStream fos=new FileOutputStream("abc.txt",true);
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(s1);
        oos.flush();
        System.out.println("Serializable process success");
    }
}
```

**To perform deserialization:- in the network the file is available with java data to read the data we have to go for deserialization.**

```
import java.io.*;
```

```

class Deserialization
{
    public static void main(String args[])throws Exception
    {
        //deserialization process
        FileInputStream fis=new FileInputStream("abc.txt");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Student s=(Student)ois.readObject();

        System.out.println("the student name is:"+s.name);
        System.out.println("the student id is:"+s.id);
        System.out.println("the student marks:"+s.marks);
        System.out.println("deserialization success");
    }
}

```

### Transient Modifiers

- Transient modifier is the modifier applicable for only variables and we can't apply for methods and classes.
- At the time of serialization, if we don't want to save the values of a particular variable to meet security constraints then we should go for transient modifier.
- At the time of serialization JVM ignores the original value of transient variable and default value will be serialized.

```

import java.io.*;
import java.io.Serializable;
class Student implements Serializable
{
    transient int id=100;
    transient String name="ravi";
}
class Serializable1
{
    public static void main(String args[])throws Exception
    {
        Student s1=new Student();
        System.out.println("the student id is:"+s1.id);
        System.out.println("the student name is:"+s1.name);
        FileOutputStream fos=new FileOutputStream("ratan.txt",true);
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(s1);
        FileInputStream fis=new FileInputStream("ratan.txt");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Student s=(Student)ois.readObject();
        System.out.println("the student id is:"+s.id);
        System.out.println("the student name is:"+s.name);
    }
}

```

## Exception Handling

### Information regarding Exception :-

- 1) Dictionary meaning of the exception is abnormal termination.
- 2) An exception is a problem occurred during execution time of the program.
- 3) An unwanted unexpected event that disturbs normal flow of execution called exception.
- 4) Exception is nothing but a object.
- 5) Exception is a class present in java.lang package.
- 6) All the exceptions are nothing but objects called classes.
- 7) Whenever user is entered invalid data then Exception is occur.
- 8) A file that needs to be opened can't found then Exception is occurred.
- 9) Exception is occurred when the network has disconnected at the middle of the communication.

### Types of Exceptions:-

As per sun micro systems standards The Exceptions are divided into three types

- 1) Checked Exception
- 2) Unchecked Exception
- 3) Error

### checkedException:-

The Exceptions which are checked by the compiler at compilation time for the proper execution of the program at runtime is called CheckedExceptions.

Ex:- IOException,SQLException etc.....

Some of the checked Exceptions in the java language

Exception	Description
ClassNotFoundException	If the loaded class is not available
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	If the requested method is not available.

**uncheckedException:-**

The exceptions which are not checked by the compiler at compilation time is called uncheckedException . These checking down at run time only.

Ex:-     ArithmeticException,NullPointerException, etc.....

Some of the unchecked exceptions in the java language:-

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.(out of range)
InputMismatchException	If we are giving input is not matched for storing input.
ClassCastException	If the conversion is Invalid.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.

**Error:-**

Errors are caused by lack of system resources . these are non recoverable.

Ex:-     StackOverflowError,OutOfMemoryError,AssertionError etc.....

**Goodpoint:-**

The Exception whether it is checked or unchecked the Exceptions are occurred at runtime.

**Difference between Exception and Error:-****Exception:-**

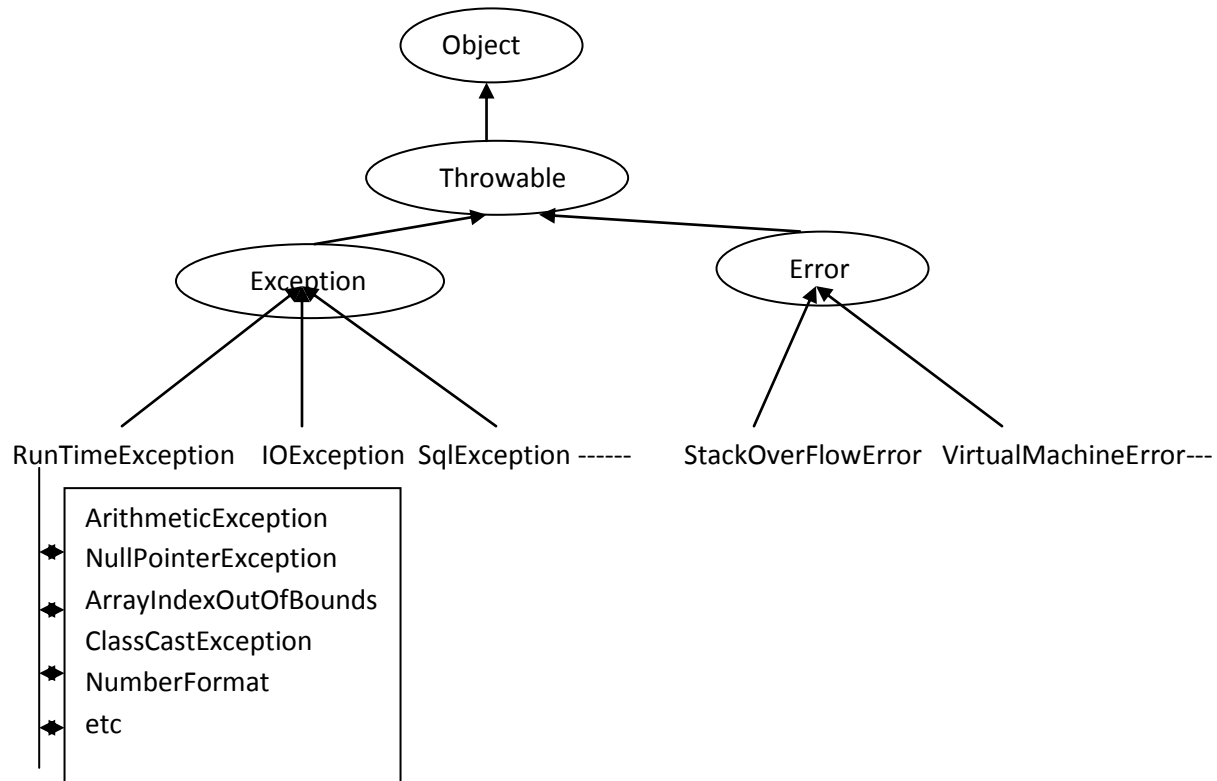
An exception is unwanted unexpected event these are caused by the programmer mistake.  
Exceptions are recoverable.

Ex:- IOException,SQLException,RuntimeException etc.....

**Error:-**

Errors are caused by lack of system resources . these are non recoverable.

Ex:- StackOverFlowError,AssertionError etc.....

**Exception Handling Tree Structure:-****Note:-**

- 1) RuntimeException and its child classes and Error and its child classes are Unchecked remaining all are checkedExceptions.
- 2) Root class for all Exception hierarchy is Throweable class.

**In java Exception handling we are having 5 key words:-**

- 1) try
- 2) catch
- 3) finally
- 4) throw
- 5) throws

**Exception Handling:-**

It is recommended to handle the Exception the main of the Exception Handling is normal Execution of the program or graceful termination of the program at runtime.

We are able to handle the exception in two ways.

1. By using try-catch blocks
2. By using throws keyword.

#### Exception handling by using Try –catch block:-

- 1) In java language we are handling the exceptions By using try and catch blocks. try block contains risky code of the program and catch block contains handling code of the program.
- 2) Catch block code is a alternative code for Exceptional code. If the exception is raised the alternative code is executed fine then rest of the code is executed normally.

#### Syntax:-

```
try
{
    Code to run [break;]
}
Catch(ExceptionName reference_variable)
{
    Code to run if an exception is raised
}
```

#### Before try and catch:-The program goes to abnormal termination .

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("durga");
        System.out.println("software");
        System.out.println(10/0);
        System.out.println("solutions");
    }
}
```

Output:-  
 durga  
 Software  
 Exception in Thread "main" :java.lang.ArithmeticException: / by zero

Note:- if we are not using try-catch it is always abnormal termination if an Exception raised.

Exception in Thread "main" java.lang.ArithmeticException: / by zero

Handled by JVM                      type of the Exception                      discription

#### After try catch:-

- 1) If we are taking try-catch the program goes to normal termination. Because the risky code we are taking inside the try block and handling code we are taking inside the catch block.
- 2) If the exception is raised in the try block the corresponding catch block is executed.
- 3) If the corresponding catch block is not there program goes to abnormal termination.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("durga");
        System.out.println("software");
        try
        {
            System.out.println(10/0);
        }
        catch (ArithmeticException e)
        {
            System.out.println("you are getting AE "+e);
        }
        System.out.println("solutions");
    }
}
```

Output:-  
durga  
Software  
You are getting AE: java.lang.ArithmeticException: / by zero  
Solutions.

**Ex 1:-**

Exception raised in try block the JVM will search for corresponding catch block if the catch block is matched, corresponding catch block will be executed and rest of the code is executed normally.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            int[] a={10,20,30};
            System.out.println(a[0]);
            System.out.println(a[1]);
            System.out.println(a[2]);
            System.out.println(a[3]);
        }
        catch(ArrayIndexOutOfBoundsException ae)
        {
            System.out.println("we are getting exception");
        }
    }
}
```

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

Ex 2:-

Exception raised in try block the JVM will search for corresponding catch block if the catch block is matched, corresponding catch block will be executed and rest of the code is executed normally. If the catch block is not matched the program is terminated abnormally.

class Test

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("program starts");  
        try  
        {  
            int[] a={10,20,30};  
            System.out.println(a[0]);  
            System.out.println(a[1]);  
            System.out.println(a[2]);  
            System.out.println(a[3]);  
        }  
        catch(ArithmeticException ae)  
        {  
            System.out.println("we are getting exception");  
        }  
        System.out.println("rest of the code");  
    }  
}
```

Ex 3:- if there is no exception in try block the catch blocks won't be executed.

Class Test

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("program starts");  
        try  
        {  
            System.out.println("ratan sir");  
            System.out.println("how r u");  
        }  
        catch(ArithmeticException ae)  
        {  
            System.out.println("we are getting exception");  
        }  
        System.out.println("rest of the code");  
    }  
}
```



Ex 4:- independent try blocks are not allowed in java language.(compilation error)

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            int a=10/0;
        }
        System.out.println("rest of the code is available");
    }
}
```

**Ex 5:-**

In between try and catch independent statements are not allowed. If we are providing independent statements the compiler will raise compilation error.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            int a=10/0;
        }
        System.out.println("in between try and catch");

        catch(ArithmeticException e)
        {
            int a=10/5;
            System.out.println(a);
        }
        System.out.println("rest of the code is available");
    }
}
```

**Ex 6:-**

- 1) Inside the try block once we are getting exception the JVM search for the corresponding catch block .
- 2) If the catch block is matched then it will executed that catch block the program is terminated normally the control never goes try block once again.
- 3) Once the control is out of the try block the control never entered into try block once again .

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            System.out.println("durgasoft");
            int a=10/0;
            (1) System.out.println("hi girls");
            (2)System.out.println("how are you boys");
        }
        catch(ArithmeticException e)
        {
            int a=10/5;
            System.out.println(a);
        }
        System.out.println("rest of the code ");
    }
}
```

**1 & 2 won't be executed**

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            System.out.println("hi girls");(1)
            System.out.println("how are you boys");(2)
            System.out.println("durgasoft");
            int a=10/0;
        }
        catch(ArithmeticException e)
        {
            int a=10/5;
            System.out.println(a);
        }
        System.out.println("rest of the code ");
    }
}
```

**1 & 2 will be executed.**

**Ex 7:-** The way of handling the exception is varied from exception to the exception so it is recommended to provide try with number of catch blocks.

```
import java.util.*;
```

```
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("provide the division value");
        int n=s.nextInt();
        try
        {
            System.out.println(10/n);
            String str=null;
            System.out.println("u r name is :"+str);
            System.out.println("u r name length is--->"+str.length());
        }
        catch (ArithmeticException ae)
```

```
        {
            System.out.println("good boy zero not allowed geting Exception"+ae);
        }
        catch (NullPointerException ne)
        {
            System.out.println("good girl getting Exception"+ne);
        }
        System.out.println("rest of the code");
    }
}
```

**Ex 8:-** By using root class (Exception) we are able to hold any type of exceptions.

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("provide the division value");
        int n=s.nextInt();
        try
        {
            System.out.println(10/n);
            String str=null;
            System.out.println("u r name is :"+str);
            System.out.println("u r name length is--->"+str.length());
        }

        catch (Exception e)
        {
            System.out.println("getting Exception"+e);
        }
        System.out.println("rest of the code");
    }
}
```

Ex 9:-in java class if we are declaring multiple catch blocks at that situation the catch block order should be child to parent shouldn't be parent to the child.

**(No compilation error)****Child-parent**

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("provide the division val");
        int n=s.nextInt();
        try
        {
            System.out.println(10/n);
            String str=null;
            System.out.println("u r name is :"+str);
            System.out.println("u r name length is--
->"+str.length());
        }
        catch (ArithmeticException ae)
        {
            System.out.println("Exception"+ae);
        }
        catch (Exception ne)
        {
            System.out.println("Exception"+ne);
        }
        System.out.println("rest of the code");
    }
}
```

**Compilation error**

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("provide the division val");
        int n=s.nextInt();
        try
        {
            System.out.println(10/n);
            String str=null;
            System.out.println("u r name is :"+str);
            System.out.println("u r name length is--
->"+str.length());
        }
        catch (Exception ae)
        {
            System.out.println("Exception"+ae);
        }
        catch (ArithmeticException ne)
        {
            System.out.println("Exception"+ne);
        }
        System.out.println("rest of the code");
    }
}
```

**Ex 10:-** The exception raised in catch block it is always abnormal termination.

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("program starts");
        try
        {
            int a=10/0;
        }
        catch(ArithmeticException e)
        {
            int a=10/0;
            System.out.println(a);
        }
        System.out.println("rest of the code is avilable");
    }
}
```

**Possibilities of try-catch blocks:-****1)single time try-catch:-**

```
try
{
}
catch ()
{
}
```

**2)multiple times try-catch:-**

```
try
{
}
catch ()
{
}
try
{
}
catch ()
{
}
```

**3)try with multiple catches:-**

```
try
{
}
catch ()
{
}
catch ()
{
}
```

**4)nested try-catch:-**

```
try
{
    try
    {
    }
    catch ()
    {
    }
}
```

```
}
catch ()
{
}
```

**5)catch with try-catch:-**

```
try
{
}
catch ()
{
    try
    {
    }
    catch ()
    {
    }
}
```

**6)**

```
try
{
    try
    {
    }
    catch ()
    {
    }
}
catch ()
{
    try
    {
    }
    catch ()
    {
    }
}
```

**1)single time try-catch:-**

- 1) if we provide try-catch whenever we are getting Exception the corresponding catch block is matched then the code executed good then we are getting normal flow of execution.
- 2) If the corresponding catch block is not matched then our program always goes to abnormal termination.

```
try
{
    Risky code
}
catch ()
{
    Alternative code
}
```

**Ex 1 :-**

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("hi girls");
            System.out.println(10/0);
            System.out.println("hi boys");
        }
        catch (ArithmeticException e)
        {
            System.out.println("we are getting ArithmeticException"+e);
        }
        System.out.println("if we provide try-catch");
        System.out.println("the rest of the code is executed");
    }
};
```

Note : the program goes to normal termination

**2)multiple times try-catch:-**

```
try
{
}
catch ()
{
}
try
{
}
catch ()
{
}
```

Ex:-

class Test

```
{
    static String str;//instance variable default value is :null
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);
        }
        catch (ArithmeticException ae)
        {
            System.out.println("we are getting ArithmeticException"+ae);
        }
        try
        {
            System.out.println(str.length());
        }
        catch (NullPointerException ne)
        {
            System.out.println("we are getting nullpointerException"+ne);
        }
        System.out.println("if we provide try-catch");
        System.out.println("the rest of the code is executed");
    }
};
```

**3)try with multiple catches:-**

```
try
{
}
catch ()
{
}
catch ()
{
}
```

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("provide the division value");
        int n=s.nextInt();
        try
        {
            System.out.println("*****good boy check the output*****");
            System.out.println(10/n);
            String str=null;
            System.out.println("u r name is :"+str);
            System.out.println("u r name length is--->"+str.length());
        }
        catch (ArithmeticException ae)
        {
            System.out.println("good boy zero not allowed geting Exception"+ae);
        }
        catch (NullPointerException ne)
        {
            System.out.println("good girl getting Exception"+ne);
        }
        System.out.println("rest of the code");
    }
}
```



**4)nested try-catch:-**

```
try
{
    try
    {
    }
    catch ()
    {
    }
}
catch ()
{
}
```

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            try
            {
                System.out.println("first try block");
                int a=10/0;
                System.out.println(a);
            }
            catch (ArithmeticException ae)
            {
                System.out.println("first catch block"+ae);
            }
            try
            {
                System.out.println("second try block");
                int[] a={10,20,30};
                System.out.println(a[5]);
            }
            catch (ArrayIndexOutOfBoundsException aaa)
            {
                System.out.println("second catch block"+aaa);
            }
        }
        catch (Exception e)
        {
            System.out.println("main catch"+e);
        }
        System.out.println("normal flow of execution");
    }
}
```

**5)catch with try-catch:-**

```
try
{
}
catch ()
{
    try
    {
    }
    catch ()
    {
    }
}
```

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);
        }
        catch (ArithmeticException ae)
        {
            try
            {
                System.out.println("hi Rattaiah");
            }
            catch (Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
```

**Ex :-**

```
statement 1
statement 2
try
{
    statement 3
    try
    {
        statement 4
        statement 5
    }
    catch ()
    {
    }
}
```

```

        statement 6
        statement 7
    }
}
catch ()
{
    statement 8
    statement 9
    try
    {
        statement 10
        statement 11
    }
    catch ()
    {
        statement 12
        statement 13
    }
}
statement 14
statement 15

```

**case 1:-****if there is no Exception in the above example**

1, 2, 3, 4, 5, 14, 15 Normal Termination

**Case 2:-****if the exception is raised in statement 2**

1, Abnrmal Termination

**Case 3:-****if the exception is raised in the statement 3 the corresponding catch block is matched.**

1,2,8,9,10,11,14,15 normal termination

**Case 4:-****if the exception is raise in the statement-4 the corresponding catch block is not matched and outer catch block is not matched.**

1,2,3 abnormal termination.

**Case 5:-****If the exception is raised in the statement 5 and corresponding catch block is not matched and outer catch block is matched.**

1,2,3,4,8,9,10,11,14,15 normal termination

**Case 6:-****If the exception is raised in the statement 5 and the corresponding catch block is not matched and outer catch block is matched while executing outer catch inside the try block the the exception is raised in the statement 10 and the corresponding catch is matched.**

1,2,3,4,8,9,12,13,14,15 normal termination.

**Case 7:-****If the exception raised in statement 14.**

1,2,3,4,5 abnormal termination.

**Finally block:-**

- 1) finally is a block it is always executed irrespective of try and catch.
- 2) Finally contains clean-up code.
- 3) It is not possible to write finally alone . we must take try-catch-finally otherwise take the try-finally these two are the possibilities. If we are taking any other we are getting compilation error saying finally without try block .

**Syntax:-**

```
try
{
    risky code;
}
catch (Exception obj)
{
    handling code;
}
finally
{
    free code;
}
```

**Ex :- Exception raised in try block and corresponding catch block is matched then rest of the code is executed normally.**

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("durga");
            System.out.println(10/0);
        }
        catch (ArithmeticException ae)
        {
            System.out.println("u r getting ae:"+ae);
        }
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code");
    }
}
```

**Output:-**           durga  
                  U r getting ae:ArithmeticException : /by zero  
                  Finally block is always executed

Ex:-

class Test

```
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("durga");
            System.out.println(10/0);
        }
        catch (NullPointerException ne)
        {
            System.out.println("u r getting ne"+ne);
        }

        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code"); //this line not executed
    }
}
```

Output:-  
 durga  
 Finally block is always executed  
 Exception in Thread "main" :java.lang.ArithmeticException: / by zero

**Ex:- finally block is not executed vs finally block is executed**

class Test

```
{
    public static void main(String[] args)
    {
        System.out.println("ratan1");
        System.out.println("ratan2");
        int a=10/0;
        try
        {
            System.out.println("ratan");
        }
        finally
        {
            System.out.println("finally block");
        }
        System.out.println("rest of the code");
    }
};
```

**The only one situation the finally block is wont be executed:-**

In your program whenever we are using System.exit(0) the JVM is shut downing hence the rest of the code won't be executed .

**Ex:-**

class Test

```
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("durga");
            System.exit(0);----->line 1
            System.out.println(10/0);
        }
        catch (ArithmeticException ae)
        {
            System.out.println("u r getting ae"+ae);
            System.exit(0);----->line 2
        }
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code");
    }
}
```

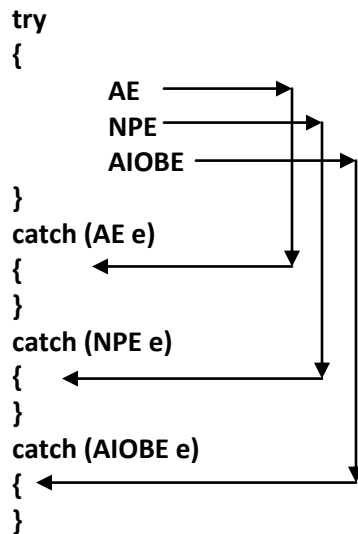
Note:- if we are commenting line 2

Output :-       durga

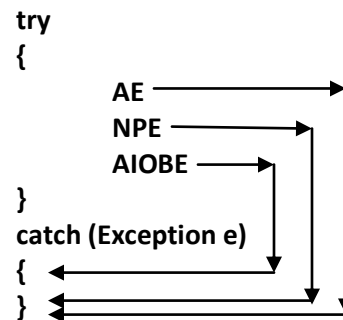
Note :- if we are commenting line 1

Output:-       durga  
              U r getting ae: ArithmeticException : /by zero

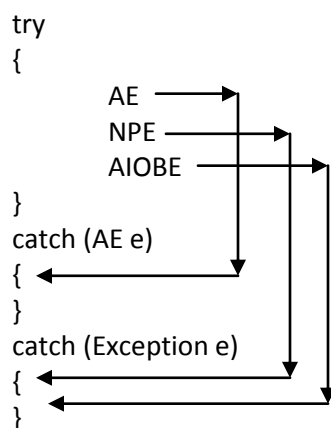
Ex 1:-



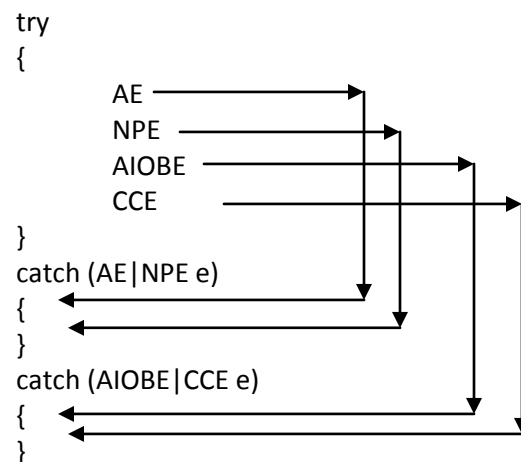
Ex 2:-



Ex 3:-



Ex 4:-introduced in 1.7 version

**Throw:-**

- 1) The main purpose of the throw keyword is to creation of Exception object explicitly either for predefined or user defined .
- 2) Throw keyword works like a try block. The difference is try block is automatically find the situation and creates a Exception object implicitly. Whereas throw keyword creates a Exception object explicitly.

Ex:-

```

class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);

```

```
        }
        catch (ArithmeticException ae)
        {
            System.out.println("we are getting Exception "+ae);
        }
    }
}
```

Output:-        we are getting Exception    ArtithmeticException: / by zero

**Note :-**

In the above program the main method is responsible to create a exception object. So the main method is creating exception object implicitly. The programmer is not responsible person to creates a exception object.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        throw new ArithmeticException("we are getting Exception / by zero man");
    }
}
```

Output:-

Exception in Thread "main" :java.lang.ArithmeticException:we are getting Exception/ by zero man

**Note:-**

In the above program the programmer is creating exception object explicitly. The main method is not responsible person to creates exception object.

Ex:-

```
import java.util.*;
class Test
{
    static void validate(int age)
    {
        if (age<18)
        {
            throw new ArithmeticException("not eligible for vote");
        }
        else
        {
            System.out.println("welcome to the voteing");
        }
    }
}
```



```
public static void main(String[] args)
{
    Scanner s=new Scanner(System.in);
    System.out.println("please enter your age ");
    int n=s.nextInt();
    validate(n);
    System.out.println("rest of the code");
}
}
```

**Throws :-**

- 1) Throw keyword is used to create exception object explicitly. But the main purpose of the throws keyword is bypassing the generated exception from present method to caller method.
- 2) Throw keyword is used in the method body. But throws keyword we have to use in the method declaration.
- 3) It is possible to throws any number of exceptions at a time based on the programmer requirement.

In the java language we are handling the Exception in two ways

- 1) By using try-catch blocks
- 2) By using throws keyword

**By using try-catch blocks:-**

Ex:-

```
import java.io.*;
class Student
{
    void studentDetails()
    {
        try
        {
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.println("please enter student name");
            String sname=br.readLine();
            System.out.println("u r name is:"+sname);
        }
        catch(IOException e)
        {
            System.out.println("we are getting Exception"+e);
        }
    }
    public static void main(String[] args)
    {
        Student s1=new Student();
        s1.studentDetails();
    }
}
```

**By using throws keyword:-**

Ex 1:-

```
import java.io.*;
class Student
{
    void studentDetails()throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("please enter student name");
        String sname=br.readLine();
        System.out.println("u r name is:"+sname);
    }
    public static void main(String[] args)throws IOException
    {
        Student s1=new Student();
        s1.studentDetails();
    }
}
```

Ex 2:-

```
import java.io.*;
class Student
{
    void studentDetails()throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("please enter student name");
        String sname=br.readLine();
        System.out.println("u r name is:"+sname);
    }
    public static void main(String[] args)
    {
        Student s1=new Student();
        try
        {
            s1.studentDetails();
        }
        catch (IOException ie)
        {
            System.out.println("this is my handling code");
        }
    }
}
```

Ex 3:-

```
import java.io.*;
class Student
{
    void studentDetails()throws IOException
```

```

    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("please enter student name");
        String sname=br.readLine();
        System.out.println("please enter student rollno");
        int sroll=Integer.parseInt(br.readLine());
        System.out.println("enter student address");
        String saddr=br.readLine();
        System.out.println("student name is:"+sname);
        System.out.println("student rollno is:"+sroll);
        System.out.println("student address is:"+saddr);
    }
    void principal() throws IOException
    {
        studentDetails();
    }
    void officeBoy()throws IOException
    {
        principal();
    }

    public static void main(String[] args) throws IOException
    {
        Student s1=new Student();
        s1.officeBoy();
    }
}

```

Exceptions:-

There are two types of exceptions present in the java language

- 1) Predefined Exceptions.
- 2) User defined Exceptions.

#### **Predefined Exception:-**

Predefined classes comes along with the software based on your requirement we have to create a objects.

Ex:- ArithmeticException,IOException,NullPointerException.....etc

#### **User defined Exceptions:-**

Based on the user requirement user can creates a Exception is called user defined Exception.

Ex:InvalidAgeException,BombBlostException.....etc

#### **To create user defined Exceptions:-**

- 1) To create user defined exception we have to take an user defined class that is a sub class to the RuntimeException(for creation of unchecked Exceptions) .
- 2) To create user defined exception we have to take user defined class that is subclass to the Exception(for creation of checked Exceptions)
- 3) Each and every Exception contains two constructors
  - a) default constructor
  - b) parameterized constructor

- 4) the naming conventions we have to follow
  - a. every exception suffix must be the word Exception.
  - b. Exception in a class so we have to follow class coding conventions.

**for the creation of UncheckedException:-****Default constructor approach**

Class InvalidAgeException extends RuntimeException

```
{  
}
```

**Parameterized constructor approach**

Class XXXException extends RuntimeException

```
{  
    XXXException(String str)  
    {  
        Super(str);  
    }  
}
```

**Note:-**

for these type of user defined Exceptions no need of handling the Exception . Hence try-catch [or] throws keyword is not required.

**For the creation of checkedException:-****Default constructor approach**

Class InvalidAgeException extends Exception

```
{  
}
```

**Parameterized constructor approach**

Class XXXException extends Exception

```
{  
    XXXException(String str)  
    {  
        Super(str);  
    }  
}
```

**Note:-**

For these type of user defined Exceptions we have to handle the Exception otherwise we are getting compilation error . Hence try-catch [or] throws keyword is not required for the handling of Exceptions.

**Ex:-preparation of custom checked exceptions:-**

//userdefined Exception class preparation

public class InvalidAgeException extends Exception

```
{  
    InvalidAgeException(String str)  
    {  
        super(str);  
    }  
}
```

```
}  
//program that uses user defined Exception class  
import java.util.*;  
class Test  
{  
    static void validate(int age) throws InvalidAgeException  
    {  
        if (age<18)  
        {  
            throw new InvalidAgeException("not eligible for vote");  
        }  
        else  
        {  
            System.out.println("welcome to the voteing");  
        }  
    }  
    public static void main(String[] args) throws InvalidAgeException  
    {  
        Scanner s=new Scanner(System.in);  
        System.out.println("please enter age");  
        int age=s.nextInt();  
        validate(age);  
    }  
}
```

**Ex:-preparation of custom unchecked exceptions:-**

```
public class InvalidAgeException extends RuntimeException  
{  
    InvalidAgeException(String str)  
    {  
        super(str);  
    }  
}  
  
import java.util.*;  
class Test  
{  
    static void validate(int age)  
    {  
        if (age<18)  
        {  
            throw new InvalidAgeException("not eligible for vote");  
        }  
        else  
        {  
            System.out.println("welcome to the voteing");  
        }  
    }  
}
```

```
public static void main(String[] args)
{
    Scanner s=new Scanner(System.in);
    System.out.println("please enter age");
    int age=s.nextInt();
    validate(age);
}
}
```

### **Different types of exceptions:-**

#### **ArrayIndexOutOfBoundsException:-**

whenever we are calling array with out of range at that moment we are getting ArrayIndexOutOfBoundsException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            int[] a={10,20,30};
            System.out.println(a[0]);//10
            System.out.println(a[1]);//20
            System.out.println(a[2]);//30
            System.out.println(a[3]);//ArrayIndexOutOfBoundsException
        }
        catch (ArrayIndexOutOfBoundsException ae)
        {
            System.out.println("boss u r geting ArrayIndexOutOfBoundsException");
            System.out.println("check u r code once");
        }
    }
}
```

#### **NumberFormatException:-**

At the time of converting String value into the integer value we are getting NumberFormatException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            String str="123";
            int a=Integer.parseInt(str);
            System.out.println(a);//conversion(string - int) is good
        }
    }
}
```

```
        String str1="abc";
        int b=Integer.parseInt(str1);
        System.out.println(b);//NumberFormatException
    }
    catch (NumberFormatException ae)
    {
        System.out.println("boss u r geting NumberFormatException");
        System.out.println("check once u r code");
    }
}
```

**NullPointerException:-**

If we are having 'null' in any variable in that variable we are performing any operation we are getting NummpointerException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            String str="rattaiah";
            System.out.println(str.length());//8

            String str1=null;
            System.out.println(str1.length());//NullPointerException

        }
        catch (NullPointerException ne)
        {
            System.out.println("boss u r geting nullpointerexception");
            System.out.println("check once u r code");
        }
    }
}
```

**ArithmeticException:-**

Whenever we are performing / by zero operation we are getting ArithmeticException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            int a=10/2;
            System.out.println(a);//5
        }
    }
}
```

```
        int b=10/0;
        System.out.println(b);//ArithmeticException
    }
    catch (ArithmeticException ne)
    {
        System.out.println("boss u r getting ArithmeticException");
        System.out.println("check once u r code");
    }
}
```

**IllegalArgumentException:-**

Thread priority range is 1-10

1---→low priority

10-→high priority

If we are giving priority out of range then we are getting IllegalArgumentException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        Thread t=new Thread();
        t.setPriority(11);//IllegalArgumentException
    }
}
```

**IllegalThreadStateException:-**

Whenever we are trying to restart the already start thread then we are getting  
IllegalThreadStateException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        Thread t=new Thread();
        t.start();
        t.start();//IllegalThreadStateException
    }
}
```

**StringIndexOutOfBoundsException:-**

Whenever we are trying to perform String based operations with out of range condition then we  
will get StringIndexOutOfBoundsException.

Ex:-

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



```
{
    String str="rattaiah";
    System.out.println(str.charAt(3));//t
    System.out.println(str.charAt(13));//StringIndexOutOfBoundsException
}
}
```

**NegativeArraySizeException:-**

If we are giving array size in negative values then we are getting NegativeArraySizeException.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        int[] a1=new int[100];
        System.out.println(a1.length);//100
        int[] a=new int[-9];
        System.out.println(a.length);//NegativeArraySizeException
    }
}
```

**InputMismatchException:-**

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        System.out.println("enter first number");
        int a=s.nextInt();
        System.out.println("enter second number");
        int b=s.nextInt();
        System.out.println(a+b);
    }
};
```

```
D:\>java Test
enter first number
123
enter second number
ratan
Exception in thread "main" java.util.InputMismatchException
```

**Different types of Errors:-****StackOverflowError:-**

Whenever we are calling method recursively then we are getting StackOverflowError.

Ex:-

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

**OutOfMemoryError:-**

If we are creating objects greater than the heap memory then we are getting OutOfMemoryError.

Ex:-

```
class Test
{
    public static void main(String[] args)
    {
        int[] a=new int[1000000000]; //OutOfMemoryError
    }
}
```

**NoClassDefFoundError:-**

Whatever the class if we want to execute if the class is not available at runtime we are getting NoClassDefFoundError.

Ex:-

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

Output:- javac Test.java  
Java Test

o/p:-Rattaiah

if we are executing class ABC (java ABC) if that class is not available then we are getting  
NoClassDefFoundError .

**java 7 features:-**

multi-catch:-

the multi-catch feature is allows two or more exceptions to be caught by the same catch clause.

import java.util.\*;

class Test

```
{
    public static void main(String[] args)
    {
        int a[]=new int[3];;
        Scanner s=new Scanner(System.in);
        System.out.println("enter n value");
        int n=s.nextInt();
        try
        {
            System.out.println(10/n);
            for (int i=0;i<a.length;i++ )
            {
                System.out.println("enter "+i+" value");
                a[i]=s.nextInt();
            }
            System.out.println(a[7]);
        }
        catch (ArithmeticException|ArrayIndexOutOfBoundsException e)
        {
            System.out.println(e);
        }
        System.out.println("program is ended");
    }
};
```

## Multi Threading

### Information about multithreading:-

- 2) The earlier days the computer's memory is occupied only one program after completion of one program it is possible to execute another program is called uni programming.
- 3) Whenever one program execution is completed then only second program execution will be started such type of execution is called co operative execution, this execution we are having lot of disadvantages.
  - a. Most of the times memory will be wasted.
  - b. CPU utilization will be reduced because only program allow executing at a time.
  - c. The program queue is developed on the basis co operative execution

**To overcome above problem a new programming style will be introduced is called multiprogramming.**

- 1) Multiprogramming means executing the more than one program at a time.
- 2) All these programs are controlled by the CPU scheduler.
- 3) CPU scheduler will allocate a particular time period for each and every program.
- 4) Executing several programs simultaneously is called multiprogramming.
- 5) In multiprogramming a program can be entered in different states.
  - a. Ready state.
  - b. Running state.
  - c. Waiting state.
- 6) Multiprogramming mainly focuses on the number of programs.

### Advantages of multiprogramming:-

1. CPU utilization will be increased.
2. Execution speed will be increased and response time will be decreased.
3. CPU resources are not wasted.

### Thread:-

- 1) Thread is nothing but separate path of sequential execution.
- 2) The independent execution technical name is called thread.
- 3) Whenever different parts of the program executed simultaneously that each and every part is called thread.
- 4) The thread is light weight process because whenever we are creating thread it is not occupying the separate memory it uses the same memory. Whenever the memory is shared means it is not consuming more memory.
- 5) Executing more than one thread a time is called multithreading.

**Single threaded model:-**

```
class Test
```

```
{
```

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

```
    {
```

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

```
        System.out.println("hi rattaiah");
```

```
        System.out.println("hello durgasoft");
```

```
    }
```

```
}
```

} begins

} body

} end

In the above program only one thread is available is called main thread to know the name of the thread we have to execute the following code.

```
class Test
```

```
{
```

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

```
    {
```

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

```
        Thread t=Thread.currentThread();
```

```
        System.out.println("current thread information is : "+t);//[main,5,main]
```

```
        System.out.println("current thread priority is      : "+t.getPriority());//5
```

```
        System.out.println("current thread name is          : "+t.getName());
```

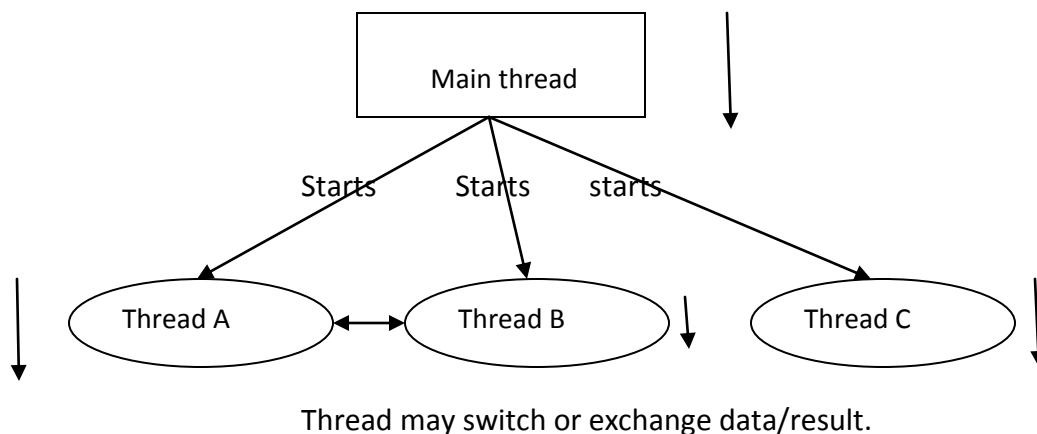
```
        System.out.println("hi rattaiah");
```

```
        System.out.println("hello durgasoft");
```

```
    }
```

```
}
```

In the above program only one thread is available name of that thread is main thread.

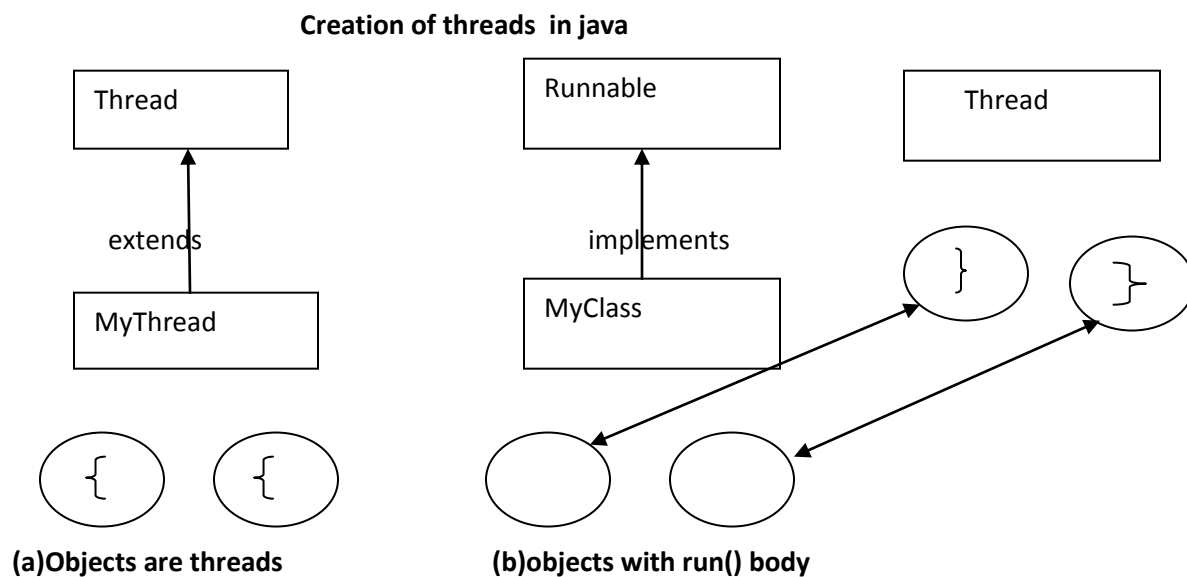
**Multithreaded model:-**

### The main important application areas of the multithreading are

1. Developing video games
2. Implementing multimedia graphics.
3. Developing animations

### There are two different ways to create a thread isavailable

- 1) Create class that extending standered java.lang.Thread Class
- 2) Create class that Implementing java.lang.Runnable interface



### First approach to create thread extending Thread class:-

#### Step 1:-

**Creates a class that is extend by Thread classes and override the run() method**

class MyThread extends Thread

```
{
    public void run()
    {
        System.out.println("business logic of the thread");
        System.out.println("body of the thread");
    }
};
```

#### Step 2:-

**Create a Thread object**

MyThread t=new MyThread();

**Step 3:-**

**Starts the execution of a thread.**

```
t.start();
```

**In this approach take one user defined class class that is extending Thread class .**

**Ex:-**

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Rattaiah from durgasoft");
        System.out.println("body of the thread");
    }
};
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
    }
}
```

**Note :-**

- 1) Whenever we are calling t.start() method the JVM search for the start() in the MyThread class but the start() method is not present in the MyThread class so JVM goes to parent class called Thread class and search for the start() method.
- 2) In the Thread class start() method is available hence JVM is executing start() method.
- 3) Whenever the thread class start() that start() is responsible person to call run() method.
- 4) Finally the run() automatically executed whenever we are calling start() method.
- 5) Whenever we are giving a chance to the Thread class start() method then only a new thread will be created.

**Life cycle stages are:-**

- 1) **New**
- 2) **Ready**
- 3) **Running state**
- 4) **Blocked / waiting / non-running mode**
- 5) **Dead state**

**New :-**

```
MyThread t=new MyThread();
```

**Ready :-**

```
t.start()
```

**Running state:-**

If thread scheduler allocates CPU for particular thread. Thread goes to running state

The Thread is running state means the run() is executed.

**Blocked State:-**

If the running thread got interrupted or goes to sleeping state at that moment it goes to the blocked state.

**Dead State:-**

If the business logic of the project is completed means run() over thread goes dead state.

**Second approach to create thread implementing Runnable interface:-****Step 1:-**

**Creates a class that implements Runnable interface.**

```
class MyClass extends Runnable
{
    public void run()
    {
        System.out.println("Rattaiah from durgasoft");
        System.out.println("body of the thread");
    }
};
```

**Step 2:-**

**Creating a object.**

```
MyClass obj=new MyClass();
```

**Step 3:-**

**Creates a Thread class object.**

```
Thread t=new Thread(obj);
```

**Step 4:-**

**Starts the execution of a thread.**

```
t.start();
```

**implementing Runnable interface**

```
class MyThread implements Runnable
{
    public void run()
    {
        System.out.println("Rattaiah from durgasoft");
        System.out.println("body of the thread");
    }
}
class ThreadDemo
```



```
{  
    public static void main(String[] args)  
    {  
        MyClass obj=new MyClass();  
        Thread t=new Thread(obj);  
        t.start();  
    }  
}
```

**Step 1:-**

the Class MyClass implements the Runnable interface and overriding run() method and contains the logic associates with the body of the thread.

**Step 2:-**

Creates the object of implementation class this is not like a first mechanism.

**Step 3 :-**

Creates a generic thread object then pass the MyClass reference variable as a parameter to that object.

**Step 4:-**

As a result of third step 3 a thread object is created in order to execute this thread method we need to call start() method. Then new thread is executed.

**We are having two approaches:-****First approach:-**

- 1) By extending the thread class, the derived class itself is a thread object and it gains full control over the thread life cycle.
- 2) Another important point is that when extending the Thread class, the sub class cannot extend any other base classes because Java allows only single inheritance.

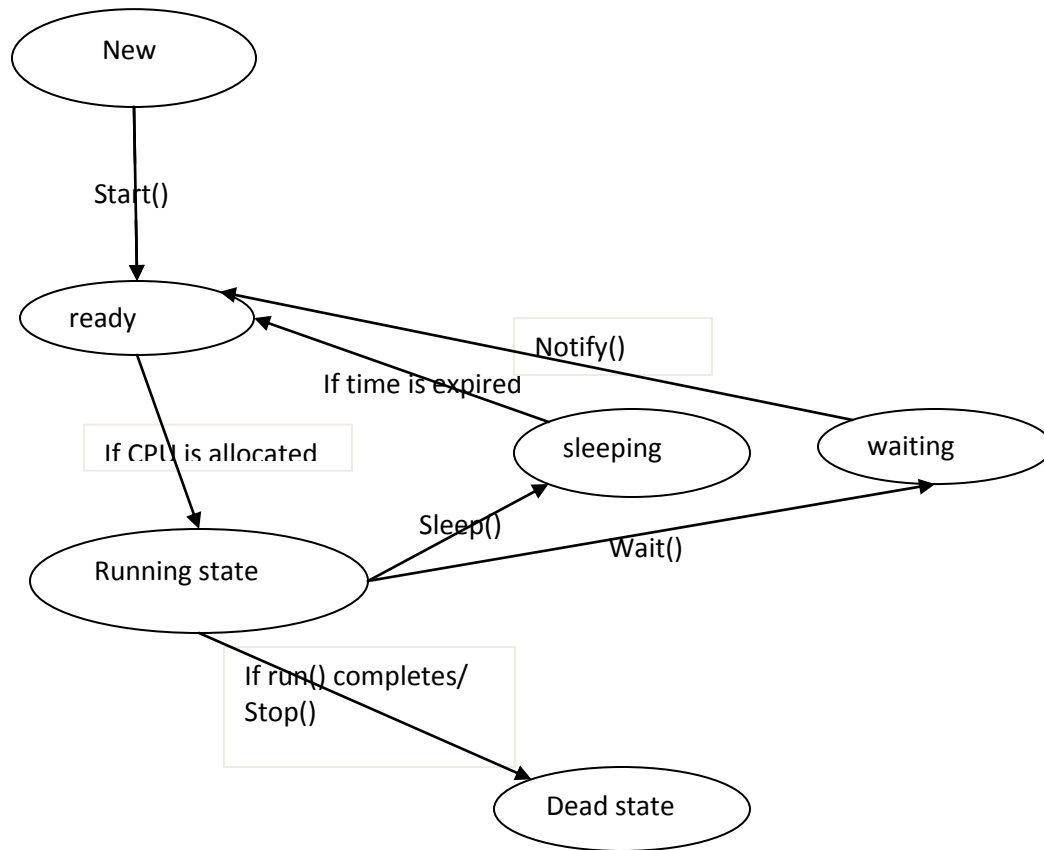
**if the program needs a full control over the thread life cycle, then extending the Thread class is a good choice.**

**Second approach:-**

- 1) Implementing the Runnable interface does not give developers any control over the thread itself, as it simply defines the unit of work that will be executed in a thread.
- 2) By implementing the Runnable interface, the class can still extend other base classes if necessary.

**if the program needs more flexibility of extending other base classes, implementing the Runnable interface would be preferable.**

We are having two approaches to create a thread use any approach based on application requirement.

**Thread life cycle:-**  
33333333**Thread life cycle**

**Internal Implementation of multiThreading:-**

Runnable-----→abstract method

Thread-----→empty implementation run() method

Mythread-----→based on our requirement we are providing implementation (overriding run() method)

**interface Runnable**

```
{  
    public abstract void run();  
}
```

**class Thread implements Runnable**

```
{  
    public void run()  
    {  
    }  
};
```

**class MyThread extends Thread**

```
{  
    public void run()  
    {  
        for (int i=0;i<5 ;i++ )  
        {  
            System.out.println("user implementation");  
        }  
    }  
};
```

**Thread Scheduler:-**

- Thread scheduler is a part of the JVM. It decides which thread is executed first and which thread is executed next.
- Only one thread is executed at a time.
- We can't expect exact behavior of the thread scheduler it is JVM vendor dependent. So we can't expect output of the multithreaded examples we can say the possible outputs.
- Thread Scheduler mainly uses preemptive (or) time slicing to schedule the threads.

**Preemptive scheduling:-**

In this highest priority task is executed first after this task enters into waiting state or dead state then only another higher priority task come to existence.

**Time Slicing Scheduling:-**

A task is executed predefined slice of time and then return pool of ready tasks. The scheduler determines which task is executed based on the priority and other factors.

**Difference between t.start() and t.run():-**

- In the case of t.start(), Thread class start() is executed a new thread will be created that is responsible for the execution of run() method.
- But in the case of t.run() method, no new thread will be created and the run() is executed like a normal method call by the main thread.

**Note :-**

**Here we are not overriding the run() method so thread class run method is executed which is having empty implementation so we are not getting any output.**

```
class MyThread extends Thread
{
}
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        for (int i=0;i<5;i++ )
        {
            System.out.println("main thread");
        }
    }
}
```

**Note :-**

**If we are overriding start() method then JVM is executes override start() method at this situation we are not giving chance to the thread class start() hence n new thread will be created only one thread is available the name of that thread is main thread.**

```
class MyThread extends Thread
{
    Public void start()
    {
        System.out.println("override start method");
    }
}
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        for (int i=0;i<5 ;i++ )
        {
            System.out.println("main thread");
        }
    }
}
```

**Particular task is performed by the number of threads:-**

- 1) Particular task is performed by the number of threads here number of threads(t1,t2,t3) are executing same method (functionality).
- 2) In the above scenario for each and every thread one stack is created. Each and every method called by particular Thread the every entry stored in the particular thread stack.

Here Four Stacks are created

Main -----stack1  
t1-----stack2  
t2-----stack3  
t3-----stack4

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("durgasoft task");
    }
}
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        MyThread t3=new MyThread();
        t1.start();
        t2.start();
        t3.start();
    }
}
```

**Ex5:-multiple threads are performing multiple operation.**

```
class MyThread1 extends Thread
{
    public void run()
    {
        System.out.println("mythread1 task");
    }
}

class MyThread2 extends Thread
{
    public void run()
    {
        System.out.println("mythread2 task");
    }
}
```

```
class MyThread3 extends Thread
{
    public void run()
    {
        System.out.println("Mythread3 task");
    }
}
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread1 t1=new MyThread1();
        MyThread2 t2=new MyThread2();
        MyThread3 t3=new MyThread3();
        t1.start();
        t2.start();
        t3.start();
    }
}
```

#### Getting and setting names of Thread:-

- 1) Every Thread in java has some name if may be default name provided by the jvm or customized name provided by the programmer.

The following methods are useful to set and get the name of a Thred.

- a. **Public final String getName()**
- b. **Public final void setName(String name)**

#### Ex:-

```
class MyThread extends Thread
{
}
class Test
{
    public static void main(String args[])
    {
        System.out.println(Thread.currentThread().getName());
        MyThread t=new MyThread();
        System.out.println(t.getName());
        Thread.currentThread().setName("meena");
        System.out.println(Thread.currentThread().getName());
    }
}
```

**Thread Priorities:-**

1. Every Thread in java has some property. It may be default priority provided by the JVM or customized priority provided by the programmer.
2. The valid range of thread priorities is 1 – 10. Where one is lowest priority and 10 is highest priority.
3. The default priority of main thread is 5. The priority of child thread is inherited from the parent.
4. Thread defines the following constants to represent some standard priorities.
5. Thread Scheduler will use priorities while allocating processor the thread which is having highest priority will get chance first and the thread which is having low priority.
6. If two threads having the same priority then we can't expect exact execution order it depends upon Thread Scheduler.
7. The thread which is having low priority has to wait until completion of high priority threads.
8. Three constant values for the thread priority.

- a. **MIN\_PRIORITY = 1**
- b. **NORM\_PRIORITY = 5**
- c. **MAX\_PRIORITY = 10**

Thread class defines the following methods to get and set priority of a Thread.

- a. **Public final int getPriority()**
- b. **Public final void setPriority(int priority)**

Here 'priority' indicates a number which is in the allowed range of 1 – 10. Otherwise we will get Runtime exception saying "IllegalArgumentException".

Ex:-

class RattaiahSirThread extends Thread

```
{
    public void run()
    {
        System.out.println("Enter into thread Rattaiah");
        System.out.println("thread Rattaiah is started");
        for (int i=0;i<10 ;i++ )
        {
            System.out.println("Rattaiah");
        }
        System.out.println("thread Rattaiah is ended");
    }
};
```

```
class NagoorSirThread extends Thread
{
    public void run()
    {
        System.out.println("Enter into thread Nagoor");
        System.out.println("thread Nagoor is started");
        for (int i=0;i<10 ;i++ )
        {
            System.out.println("Nagoor");
        }
        System.out.println("thread Nagoor is ended");
    }
};
class RamiReddySirThread extends Thread
{
    public void run()
    {
        System.out.println("Enter into thread Ramereddy");
        System.out.println("thread RamiReddy is started");
        for (int i=0;i<10 ;i++ )
        {
            System.out.println("RamiReddy");
        }
        System.out.println("thread RamiReeddy is ended");
    }
};
class ThreadDemo
{
    public static void main(String[] durga)
    {
        RattaiahSirThread thread1=new RattaiahSirThread();
        NagoorSirThread thread2=new NagoorSirThread();
        RamiReddySirThread thread3=new RamiReddySirThread();

        thread1.setPriority(Thread.MAX_PRIORITY);
        System.out.println(thread1.getPriority());

        Thread2.setPriority(Thread.MIN_PRIORITY);
        System.out.println(thread2.getPriority());

        Thread3.setPriority(thread2.getPriority()+1);
        System.out.println(thread3.getPriority());

        System.out.println("starting of Rattaiah Thread");
    }
}
```



```

        thread1.start();

        System.out.println("starting of Nagoor Thread");
        thread2.start();

        System.out.println("starting of RamiReddy Thread");
        thread3.start();
    }
};

```

### Some of the thread class methods:-

#### Sleep():-

The sleep() method causes the current thread to sleep for a specified amount of time in milliseconds.

**public static void sleep(long millis) throws InterruptedException**

**public static void sleep(long millis,int nanosec) throws InterruptedException**

For example, the code below puts the thread in sleep state for 5 minutes:

**Ex:-**

class Test

```

{
    public static void main(String[] args)
    {
        try
        {
            for (int i=0;i<10 ;i++)
            {
                System.out.println("Rattaiah");
                Thread.sleep(5*1000);//5 seconds
                Thread.sleep(5*60*1000);// 5 minits
            }
        }
        catch (InterruptedException ie)
        {
            System.out.println("the thread is got innterrupted");
        }
    }
}

```

**Ex :-**

class Test

```

{
    public static void main(String[] args)throws InterruptedException
    {
        System.out.println("Rattaiah");
        Thread.sleep(3*1000);
    }
}

```

**Java.lang.Thread.yield():-**

- ❖ Yield() method causes to pause current executing Thread for giving the chance for waiting threads of same priority.
- ❖ If there are no waiting threads or all threads are having low priority then the same thread will continue its execution once again.

**Syntax:-****Public static native void yield();****Ex:-**

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            Thread.yield();
            System.out.println("child thread");
        }
    }
}
class ThreadYieldDemo
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        t1.start();
        for(int i=0;i<10;i++)
        {
            System.out.println("main thread");
        }
    }
}
```

**Java.lang.Thread.join():-**

If a Thread wants to wait until completing some other thread then we should go for join() method.

1. Public final void join()throws InterruptedException
2. Public final void join(long ms)throws InterruptedException
3. Public final void join(long ms, int ns)throws InterruptedException

**Ex:-**

```
class MyThread extends Thread
{
    public void run()
    {
        for (int i=0;i<5;i++ )
        {
            try{
                System.out.println("rattaiah");
                Thread.sleep(3*1000);}
            catch(InterruptedException iee)
            {
                System.out.println("gettting innterrupted exctpion");
            }
        }
    }
}
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        try
        {
            t1.join();
        }
        catch (InterruptedException ie)
        {
            System.out.println("interrupted Exception");
        }
        t2.start();
    }
};
```

**Ex 2:-**

```
class MyThread extends Thread
{
    public void run()
    {
        for (int i=0;i<5;i++ )
        {
```

```

        try{
            System.out.println("rattaiah");
            Thread.sleep(3*1000);}
        catch(InterruptedException ie)
        {
            System.out.println("getting exception");
        }
    }
}
}
class ThreadDemo
{
    public static void main(String[] args)throws InterruptedException
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        t1.start();
        t1.join();
        t2.start();
    }
};

```

**isAlive():-**

used to check whether the thread is live or not.

```

    Public Boolean isAlive()
class MyThread extends Thread
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
};
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        System.out.println(t.isAlive());
        t.start();
        System.out.println(t.isAlive());
    }
};

```

**Java.lang.Thread.activeCount():-**

This method is used to find out the number of methods in active state.

Public static int activeCount();

**Ex:-**

class MyThread extends Thread

```
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
};
```

class ThreadDemo

```
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();
        MyThread t3=new MyThread();
        t1.start();
        t2.start();
        t3.start();
        System.out.println(Thread.activeCount());//4
    }
};
```

**Java.lang.currentThread():-**

This method is used to represent current thread class object.

Public static thread currentThread()

**Ex:-**

class MyThread extends Thread

```
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
};
```

class ThreadDemo

```
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
    }
```

```
        MyThread t2=new MyThread();
        t1.start();
        t2.start();
    }
};
```

**Java.lang.Thread.getId():-**

getId() is used to generate id value for each and every thread.

**Public long getId()**

**Ex:-**

```
class MyThread extends Thread
```

```
{
    public void run()
    {
        System.out.println(" rattaiah thread is running ");
    }
};
```

```
class ThreadDemo
```

```
{
    public static void main(String[] args)
    {
        MyThread t1=new MyThread();
        MyThread t2=new MyThread();

        t1.start();
        t2.start();

        System.out.println("the thread id :"+t1.getId());
        System.out.println("the thread name is      :"+t1.getName());
        System.out.println("the thread priority is "+t1.getPriority());
        System.out.println("the thread id :"+t2.getId());
        System.out.println("the thread name is      :"+t2.getName());
        System.out.println("the thread priority is "+t2.getPriority());
    }
};
```

**Interrupted():-**

- ❖ A thread can interrupt another sleeping or waiting thread.
- ❖ For this Thread class defines interrupt() method.

**Public void interrupt()**

**Effect of interrupt() method call:-**

```
class MyThread extends Thread
{
```

```
        public void run()
        {
            try
            {
                for (int i=0;i<10;i++ )
                {
                    System.out.println("i am sleeping ");
                    Thread.sleep(5000);
                }
            }
            catch (InterruptedException ie)
            {
                System.out.println("i got interrupted by interrupt() call");
            }
        }
    };
    class ThreadDemo
    {
        public static void main(String[] args)
        {
            MyThread t=new MyThread();
            t.start();
            t.interrupt();
        }
    };
};
```

**No effect of interrupt() call:-**

```
class MyThread extends Thread
{
    public void run()
    {
        for (int i=0;i<10;i++ )
        {
            System.out.println("i am sleeping ");
        }
    }
};
class ThreadDemo
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        t.interrupt();
    }
};
```

**NOTE:-**

The interrupt() is working good whenever our thread enters into waiting state or sleeping state.

The interrupted call will be wasted if our thread doesn't enters into the waiting/sleeping state.

```
class MyThread extends Thread
{
    public void run()
    {
        for (int i=0;i<10;i++)
        {
            Thread.sleep(2000);
            System.out.println("durgasoft task");
        }
    }
};

class ThreadDemo
{
    public static void main(String[] args)throws Exception
    {
        MyThread1 t1=new MyThread1();
        MyThread2 t2=new MyThread2();
        MyThread3 t3=new MyThread3();
        t1.start();
        t2.start();
        t3.start();//4-threads
        t1.join();
        System.out.println(t1.getName());//thread-0
        System.out.println(t2.getName());
        System.out.println(t3.getName());
        t1.setName("sneha");
        System.out.println(t1.getName());//sneha
        System.out.println(Thread.currentThread().getName());//main
        Thread.currentThread().setName("poornima");
        System.out.println(Thread.currentThread().getName());//poornima
        System.out.println(Thread.activeCount());
        System.out.println(t1.isAlive());
        System.out.println(t1.getId());
        System.out.println(t2.getId());
        System.out.println(Thread.currentThread().getPriority());
        System.out.println(t1.getPriority());
        Thread.currentThread().setPriority(10);
        System.out.println(Thread.currentThread().getPriority());
        for (int i=0;i<5;i++)
```



```

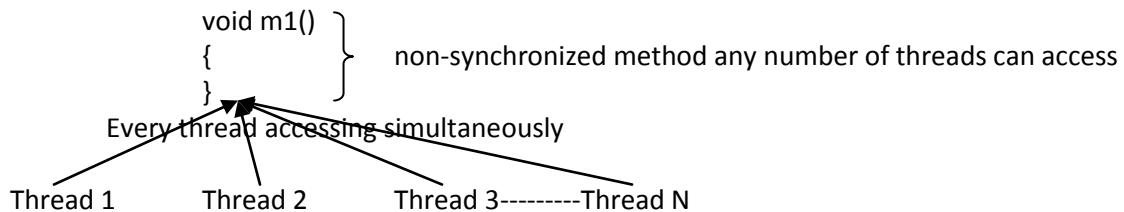
    {
        Thread.sleep(5000);
        Thread.yield();
        System.out.println("main thread");
    }
};

```

### Synchronized :-

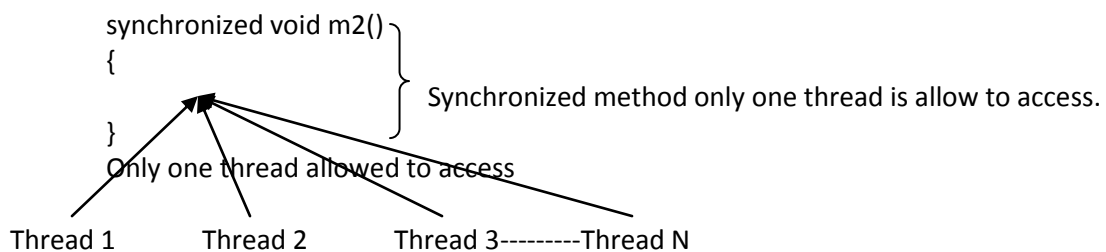
- Synchronized modifier is the modifier applicable for methods but not for classes and variables.
- If a method or a block declared as synchronized then at a time only one Thread is allowed to operate on the given object.
- The main advantage of synchronized modifier is we can resolve data inconsistency problems.
- But the main disadvantage of synchronized modifier is it increases the waiting time of the Thread and effects performance of the system .Hence if there is no specific requirement it is never recommended to use.
- The main purpose of this modifier is to reduce the data inconsistence problems.

### Non-synchronized methods



- 1) In the above case multiple threads are accessing the same methods hence we are getting data inconsistency problems. These methods are not thread safe methods.
- 2) But in this case multiple threads are executing so the performance of the application will be increased.

### Synchronized methods



- 1) In the above case only one thread is allow to operate on particular method so the data inconsistency problems will be reduced.
- 2) Only one thread is allowed to access so the performance of the application will be reduced.
- 3) If we are using above approach there is no multithreading concept.

Hence it is not recommended to use the synchronized modifier in the multithreading programming.

**Daemon threads:-**

The threads which are executed at background is called daemon threads.

Ex:- garbage collector, ThreadScheduler.default exceptional handler.

**Non-daemon threads:-**

The threads which are executed foreground is called non-daemon threads.

Ex:- normal java application.

**Volatile:-**

- Volatile modifier is also applicable only for variables but not for methods and classes.
- If the values of a variable keep on changing such type of variables we have to declare with volatile modifier.
- If a variable declared as a volatile then for every Thread a separate local copy will be created.
- Every intermediate modification performed by that Thread will take place in local copy instead of master copy.
- Once the value got finalized just before terminating the Thread the master copy value will be updated with the local stable value. The main advantage of volatile modifier is we can resolve the data inconsistency problem.
- But the main disadvantage is creating and maintaining a separate copy for every Thread
- Increases the complexity of the programming and affects performance of the system.

## Nested classes

Declaring the class inside another class is called nested classes. This concept is introduced in the 1.1 version.

The nested classes are two types

**Static nested classes:-** The nested classes declare as a static modifier is called static nested classes.

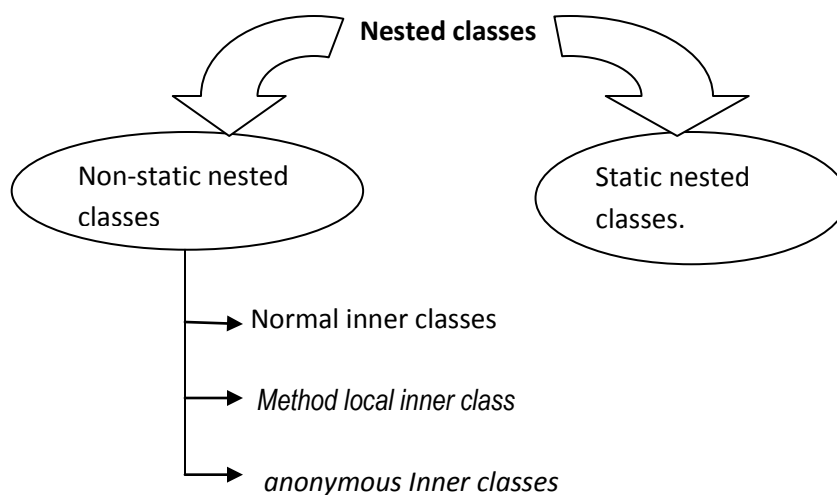
### 1. Static nested classes

**Non static nested classes:-** these are called inner classes.

### 2. Normal inner classes

### 3. Method local inner classes

### 4. Anonymous inner classes



```

Ex :-  class Outerclasses
        {
        static class staticnestedclass
        {
        };
        class Innerclass
        {
        };
        };
  
```

For the outer classes the compiler will provide the .class and for the inner classes also the compiler will provide the .class file.

The .class file name for the inner classes is **OuterclassName\$innerclassname.class**

<b>Outer class object creation</b>	<b>:-</b>	<b>Outer o=new Outer();</b>
<b>Inner class object creation</b>	<b>:-</b>	<b>Outer.Inner i=o.new Inner();</b>
<b>Outer class name</b>	<b>:-</b>	<b>Outer.class</b>
<b>Inner class name</b>	<b>:-</b>	<b>Outer\$Inner.class</b>

**Uses of nested classes:-****1. It is the way logically grouping classes that are only used in the one place.**

If a class is useful to only one other class then it is logically embedded into that classes make the two classes together.

**A is only one time usage in the B class  
(without using inner classes)**

```
class A
{
};
class B
{
    A a=new A();
};
```

**by using inner classes**

```
class B
{
    class A
    {
    };
};
```

**2. It increase the encapsulation**

If we are taking two top level classes A and B the B class need the members of A that members even we are declaring private modifier the B class can access the private numbers moreover the B is not visible for outside the world.

**3. It lead the more readability and maintainability of the code**

Nesting the classes within the top level classes at that situation placing the code is very closer to the top level class.

**Member inner classes:-**

1. If we are declaring any data in outer class then it is automatically available to inner classes.
2. If we are declaring any data in inner class then that data is should not have the scope of the outer class.

**Syntax:-**

```
class Outer
{
    class Inner
    {
    };
};
```

**Object creation syntax:-****Syntax 1:-**

```
OuterClassName o=new OuterClassName();
OuterClassName.InnerClassName oi=OuterObjectreference.new InnterClassName();
```

**Syntax 2:-**

```
OuterclassName.InnerClassName oi=new OuterClass().new InnerClass();
```

**Note:-** by using outer class name it is possible to call only outer class peroperties and methods and by using inner class object we are able to call only inner classes properties and methods.

Ex:-

```
class Outer
{
    private int a=100;
    class Inner
    {
        void data()
        {
            System.out.println("the value is :"+a);
        }
    }
}
class Test
{
    public static void main(String[] args)
    {
        Outer o=new Outer();
        Outer.Inner i=o.new Inner();
        i.data();
    }
};
```

**Ex :-**

```
class Outer
{
    int i=100;
    void m1()
    {
        //j=j+10;// compilation error
        //System.out.println(j);//compilation error
        System.out.println("m1 method");
    }
    class Inner
    {
        int j=200;
        void m2()
        {
            i=i+10;
            System.out.println(i);
        }
    }
};
class Test
{
    public static void main(String[] args)
    {
        A a=new A();
        System.out.println(a.i);
        a.m1();
    }
}
```

```

        A.B b=a.new B();
        System.out.println(b.j);
        b.m2();
        //b.m1(); compilation error
    }
};

```

**this keyword is not required :-**

```

class Outer
{
    int i=100;
    class Inner
    {
        int j=200;
        void m1(int k)
        {
            System.out.println(i);
            System.out.println(j);
            System.out.println(k);
        }
    };
};
class Test
{
    public static void main(String[] args)
    {
        Outer o=new Outer();
        Outer.Inner i=o.new Inner();
        i.m1(300);
    }
};

```

**this keyword is required:-**

```

class Outer
{
    int a=100;
    class Inner
    {
        int a=200;
        void m1(int a)
        {
            System.out.println(a);
            System.out.println(this.a);
            System.out.println(Outer.this.a);
        }
    };
};
class Test
{
    public static void main(String[] args)
    {
        Outer o=new Outer();
        Outer.Inner i=o.new Inner();
        i.m1(300);
    }
};

```

#### **Method local inner classes:-**

1. Declaring the class inside the method is called method local inner classes.
2. In the case of the method local inner classes the class has the scope up to the respective method.
3. Method local inner classes do not have the scope of the outside of the respective method.
4. whenever the method is completed
5. we are able to perform any operations of method local inner class only inside the respective method.

#### **Syntax:-**

```

class Outer
{
    void m1()
    {
        class inner
        {
        };
    }
};

```

Ex:-

```
class Outer
{
    private int a=100;
    void m1()
    {
        class Inner
        {
            void innerMethod()
            {
                System.out.println("inner class method");
                System.out.println(a);
            }
        };
        Inner i=new Inner();
        i.innerMethod();
    }
};
class Test
{
    public static void main(String[] args)
    {
        Outer o=new Outer();
        o.m1();
    }
};
```

Ex 2:-in method local inner classes it is not possible to call the non-final variables inside the inner classes hence we must declare that local variables must be final then only it is possible to access that members.

```
class Outer
{
    private int a=100;
    void m1()
    {
        final int b=1000;
        class Inner
        {
            void innerMethod()
            {
                System.out.println("inner class method");
                System.out.println(a);
                System.out.println(b);
            }
        };
        Inner i=new Inner();
        i.innerMethod();
    }
};
```

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

**Static inner classes:-**

In general in java classes it is not possible to declare any class as a abstract class but is possible to declare inner class as a static modifier.

Declaring the static class inside the another class is called static inner class.

Static inner classes can access only static variables and static methods it does not access the instance variables and instance methods.

**Syntax:-**

```
class Outer
{
    static class Inner
    {
    };
};

class Outer
{
    static int a=10;
    static int b=20;
    static class Inner
    {
        int c=30;
        void m1()
        {
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
        }
    };
    public static void main(String[] args)
    {
        Outer o=new Outer();
        Outer.Inner i=new Outer.Inner();
        i.m1();
    }
};
```

**Anonymous inner class:-**

1. The name less inner class is called anonymous inner class.
2. it can be used to provide the implementation of normal class or abstract class or interface



Ex:-

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

Ex:-

```
interface Test
{
    abstract public void m1();
};
class OuterClass
{
    void m2()
    {
        System.out.println("m2 method");
    }
    Test t=new Test()
    {
        public void m1()
        {
            System.out.println("annonymus inner class");
        }
    };
};
class Demo
```

```
{
    public static void main(String[] args)
    {
        OuterClass o=new OuterClass();
        o.m2();
        o.t.m1();
    }
};
```

**Anonymous inner classes for abstract classes:-**

**it is possible to provide abstract method implementations by taking inner classes.**

```
abstract class Animal
{
    abstract void eat();
};
class Test
{
    Animal a=new Animal()
    {
        void eat()
        {
            System.out.println("animals eating gross");
        }
    };
    public static void main(String[] args)
    {
        Test t=new Test();
        t.a.eat();
    }
}
```

Ex:-

```
abstract class Animal
{
    abstract void eat();
};
class Test
{
    public static void main(String[] args)
    {
        Animal a=new Animal()
        {
            void eat()
            {
                System.out.println("animals eating gross");
            }
        };
        a.eat();
    }
}
```

## ENUMARATION

1. This concept is introduced in 1.5 version
2. enumeration is used to declare group of named constant s.
3. we are declaring the enum by using enum keyword. For the enums the compiler will generate .classess
- 4.enum is a keyword and **Enum** is a class and every enum is directl child class of **java.lang.Enum** so it is not possible to inherit the some other class. Hence for the enum inheritance concept is not applicable
5. by default enum constants are **public static final**

```
enum Heroin      {
    Samantha,tara,ubanu ;
}
enum Week
{
    public static final smantha;
    public static final tara;
    Public static final ubanu;
}
```

### EX:-calling of enum constants individually

```
enum Heroin
{
    samantha,tara,anu;
}
class Test
{
    public static void main(String... ratan)
    {
        Heroin s=Heroin.samantha;
        System.out.println(s);
        Heroin t=Heroin.tara;
        System.out.println(t);
        Heroin a=Heroin.anu;
        System.out.println(a);
    }
};
```

### EX:-

1. printing the enumeration constants by using for-each loop.
2. values() methods are used to print all the enum constants.
3. ordinal() is used to print the index values of the enum constants.

```
enum Heroin
{
    samantha,tara,anu;
}
class Test
{
    public static void main(String... ratan)
    {
        Heroin[] s=Heroin.values();
        for (Heroin s1:s)
        {
            System.out.println(s1+"-----"+s1.ordinal());
        }
    }
};
```

1. inside the enum it is possible to declare constructors. That constructors will be executed for each and every constant. If we are declaring 5 constants then 5 times constructor will be executed.
2. Inside the enum if we are declaring only constants the semicolon is optional.
3. Inside the enum if we are declaring group of constants and constructors at that situation the group of constants must be first line of the enum must ends with semicolon.

**Ex :-Semicolon optional**

```
enum Heroin
{
    samantha,tara,anu,ubanu
}
class Test
{
    public static void main(String... ratan)
    {
        Heroin s=Heroin.samantha;
    }
};
```

**Ex:- semicolon mandatory**

```
enum Heroin
{
    samantha,tara,anu,ubanu;
    Heroin()
    {
        System.out.println("ratan sir");
    }
}
class Test
{
    public static void main(String... ratan)
    {
        Heroin s=Heroin.samantha;
    }
};
```

**Ex:- constructors with arguments**

```
enum Heroin
{
    samantha,tara(100),ubanu(100,200);
    Heroin()
    {
        System.out.println("smantha constructor");
    }
    Heroin(int a)
    {
        System.out.println(a);
        System.out.println(" tara constructor");
    }
    Heroin(int a,int b)
    {
        System.out.println(a+b);
        System.out.println(" ubanu constructor");
    }
}
class Test
{
    public static void main(String[] args)
    {
        Heroin[] s=Heroin.values();
        for (Heroin s1:s)
        {
            System.out.println(s1+"-----"+s1.ordinal());
        }
    }
};
```

**Ex:-inside the enum it is possible to provide main method.**

```
enum Heroin
{
    samantha,tara,anu;
    public static void main(String[] args)
    {
        System.out.println("enum main method");
    }
}
class Test
{
    public static void main(String... ratan)
    {
        Heroin[] s=Heroin.values();
        for (Heroin s1:s)
        {
            System.out.println(s1+"-----"+s1.ordinal());
        }
    }
};
```

**Ex:- inside the enums it is possible to declare group of constants and constructors and main method**

```
enum Heroin
{
    samantha,tara,ubanu;
    Heroin()
    {
        System.out.println("constructor");
    }
    public static void main(String[] args)
    {
        System.out.println("enum main method");
    }
}
class Test
{
    public static void main(String[] args)
    {
        Heroin[] s=Heroin.values();
        for (Heroin s1:s)
        {
            System.out.println(s1+"-----"+s1.ordinal());
        }
    }
};
```

**Garbage collector:-**

garbage collector is destroying the useless object and it is a part of the jvm.

To make eligible objects to the garbage collector

**Approach -1:-whenever we are assigning null constants to our objects then objects are eligible for gc(garbage collector)**

class Test

```
{
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test();
        System.out.println(t1);
        System.out.println(t2);
        ;
        ;
        ;
        t1=null;//t1 object is eligible for Garbage collector
        t2=null;//t2 object is eligible for Garbage Collector
        System.out.println(t1);
        System.out.println(t2);
    }
};
```

**Approach-2:-whenever we reassign the reference variable the objects are automatically eligible for garbage collector.**

class Test

```
{
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test();
        System.out.println(t1);
        System.out.println(t2);
        t1=t2;
        System.out.println(t1);
        System.out.println(t2);
    }
};
```

**gc():-**

- 1) internally the garbage collector is running to destroy the useless objects.
- 2) by using gc() method we are able to call Garbage Collector explicitly by the developer.
- 3) gc() present in System class and it is a static method.

Syntax:- System.gc();

- 3) Whenever garbage collector is destroying useless objects just before destroying the objects the garbage collector is calling finalize() method on that object to perform final operation of particular object.

Ex:-

class Test

```
{
```

```
        public void finalize()
        {
            System.out.println("ratan sir destroyed");
        }
        public static void main(String[] args)
        {
            Test t1=new Test();
            Test t2=new Test();
            System.out.println(t1);
            System.out.println(t2);
            t1=null;
            t2=null;
            System.gc();
        }
    };
```

**Approach-3 :-whenever we are creating objects inside the methods one method is completed the objects are eligible for garbage collector.**

```
class Test
{
    public void finalize()
    {
        System.out.println("object destroyed");
    }
    void m1()
    {
        Test t1=new Test();
        Test t2=new Test();
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
        System.gc();
    }
};
```

**Ex:- if the garbage collector is calling finalize method at that situation exception is raised such type of exception are ignored.**

```
class Test
{
    public void finalize()
    {
        System.out.println("ratan sir destroyed");
        int a=10/0;
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test();
        t1=t2;
        System.gc();
    }
};
```

**Ex:- If user is calling finalize() method explicitly at that situation exception is raised.**

```
class Test
{
    public void finalize()
    {
        System.out.println("ratan sir destroyed");
        int a=10/0;
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test();
        t1=t2;
        t2.finalize();
    }
};
```

**Instanceof operator:-**

**it is used check the type of the object.**

**Instanceof operator return Boolean value as a return value.**

**Syntax:- Reference-variable instanceof class-name**

**Ex:- Test t=new Test();  
t instanceof Test**

**while using instanceof operator the reference variable and class Name must have some relationship (Parent-child or child-parent) otherwise compiler will raise compilation error.**

if the reference variable is child of the specified classes at that situation instanceof return true

if the reference variable is parent of the specified classes at that situation instanceof return false.

```
class Fruit
{
};
class Apple extends Fruit
{
    public static void main(String[] args)
    {
        Apple a=new Apple();
        Object o=new Object();
        String str="ratan";
        Throwable t=new Throwable();
        System.out.println(a instanceof Fruit);
        System.out.println(a instanceof Object);
        System.out.println(str instanceof Object);
        System.out.println(o instanceof Throwable);
        System.out.println(t instanceof Throwable);
        //System.out.println(t instanceof Apple);//compilation error
    }
};
```



## **Collections(java.util)**

### **Limitations of array:-**

- 1) Array is indexed collection o fixed number of homogeneous data elements
- 2) Arrays can hold homogeneous data only
- 3) Once we created an array no chance of increasing o decreasing size of array

Ex:-

```
Student[ ] s=new Student[100];  
S[0]=new Student();  
S[1]=new Student();  
S[2]=new Customer();-----compilation error
```

To overcome the above limitations of array the sun peoples are introduced collections concept

### **Collections:-**

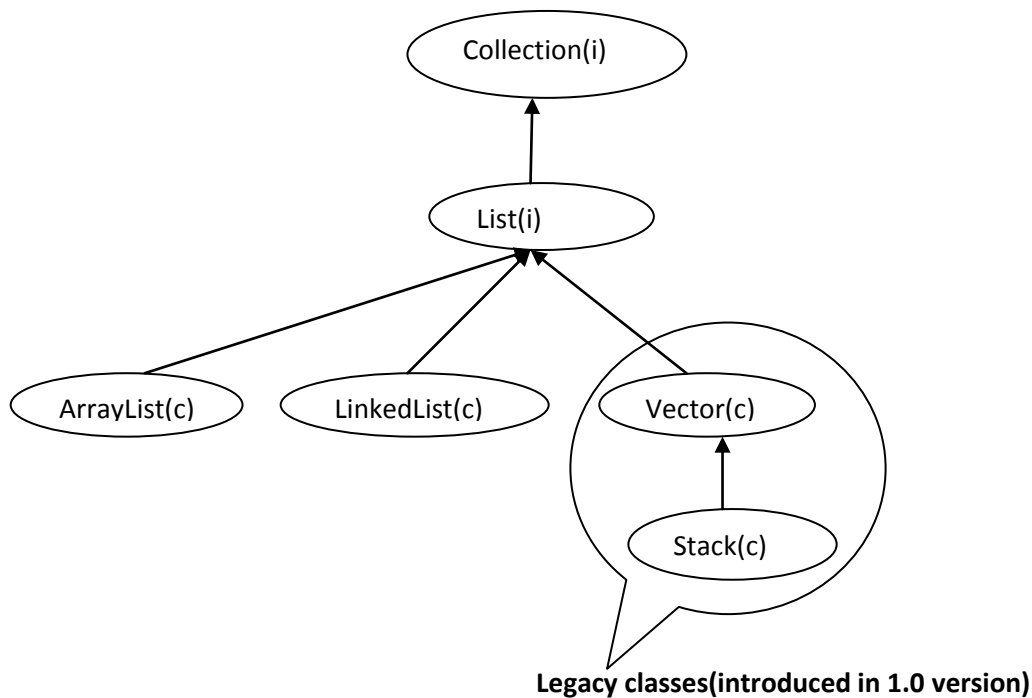
- 1) collection can hold both homogeneous data and heterogeneous data
- 2) collections are growable in nature
- 3) Memory wise collections are good. Recommended to use.
- 4) Performance wise collections are not recommended to use .

### **Collections:-**

If we want to represent group of as a single entity then we should go for collection.

### **In the collection framework we having 9 key interfaces:-**

1. Collection
2. List
3. Set
4. SortedSet
5. NavigableSet
6. Queue
7. Map
8. SortedMap
9. NavigableMap



I ----->Interface

c----->class

#### ArrayList:-

**class ArrayList extends AbstractList implements List**

the collection classes stores only objects but we are passing primitives these primitives are automatically converts into objects is called autoboxing.

- 1) Introduced in 1.2 version.
- 2) ArrayList supports dynamic array that can be grow as needed.it can dynamically increase and decrease the size.
- 3) Duplicate objects are allowed.
- 4) Null insertion is possible.
- 5) Heterogeneous objects are allowed.
- 6) The under laying data structure is growable array.
- 7) Insertion order is preserved.

Ex:-

```

import java.util.*;
class ArrayListDemo
{
    public static void main(String[] args)
    {
        //creation of ArrayList
        ArrayList al=new ArrayList();
        System.out.println("initial size of the arraylist:"+al.size());
        //adding elements to the ArrayList
        al.add("a");
    }
}

```

```

        al.add("A");
        al.add("a");
        al.add(null);
        al.add(10);
        al.add(1,"ratan");
        //print the ArrayList elements
        System.out.println(al);
        System.out.println("ArrayList size:"+al.size());
        //remove the elements of ArrayList
        al.remove("a");
        System.out.println("ArrayList size:"+al.size());
        System.out.println(al);
    }
}

```

Ex:- ArrayList with generics

```

import java.util.*;
class ArrayListDemo
{
    public static void main(String[] args)
    {
        ArrayList<Integer> al=new ArrayList<Integer>();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        System.out.println(al);
        Integer[] a=new Integer[al.size()];
        al.toArray(a);
        int sum=0;
        for (Integer a1:a)
        {
            sum=sum+a1;
            System.out.println(a1);
        }
        System.out.println(sum);
    }
}

```

**LinkedList:-****Class LinkedList extends AbstractSequentialList implements List,Deque,Queue**

- 1) Introduced in 1.2 v
- 2) Duplicate objects are allowed
- 3) Null insertion is possible
- 4) Heterogeneous objects are allowed
- 5) The under laying data structure is double linked list.
- 6) Insertion ode is preserved.

**Ex:-LinkedList with generics.**

```

import java.util.*;
class Test

```

```
{
    public static void main(String[] args)
    {
        LinkedList<String> l=new LinkedList<String>();
        l.add("a");
        l.add("ratan");
        l.add("anu");
        l.add("aaa");
        System.out.println(l);
        System.out.println(l.size());
    }
}
```

Ex:-

```
import java.util.*;
class Demo
{
    public static void main(String[] args)
    {
        LinkedList ll=new LinkedList();
        System.out.println(ll.size());
        //add the elements to the LinkedList
        ll.add("a");
        ll.add(10);
        ll.add(10.6);
        ll.addFirst("ratan");
        ll.addLast("anu");
        System.out.println("original content :"+ll);
        System.out.println(ll.size());
        //remove elements from LinkedList
        ll.remove(10.6);
        ll.remove(0);
        System.out.println("after deletion content :"+ll);
        System.out.println(ll.size());
        //remove first and last elements
        ll.removeFirst();
        ll.removeLast();
        System.out.println("ll after deletion of first and last :"+ll);
        //get and set a value
        int a=(Integer)ll.get(0);
        ll.set(0,a+"ratan");
        System.out.println("ll after change:"+ll);
    }
};
```

```
D:\app>java Demo
0
original content :[ratan, a, 10, 10.6, anu]
5
after deletion content :[a, 10, anu]
3
11 after deletion of first and last :[10]
11 after change:[10ratan]
```

#### Vector:- (legacy class introduced in 1.0 version)

- 1) Introduced in 1.0 v legacy classes.
- 2) Duplicate objects are allowed
- 3) Null insertion is possible
- 4) Heterogeneous objects are allowed
- 5) The under laying data structure is growable array.
- 6) Insertion order is preserved.
- 7) Every method present in the Vector is synchronized and hence vector object is Thread safe.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        for (int i=0;i<10;i++ )
        {
            v.addElement(i);
        }
        System.out.println(v);
        v.addElement("rattaiah");
        System.out.println(v);
        v.removeElement(0);
        System.out.println(v);
        v.clear();
        System.out.println(v);
    }
}
```

#### Stack:- (legacy class introduced in 1.0 version)

- 1) It is a child class of vector
- 2) Introduce in 1.0 v legacy class
- 3) It is designed for LIFO(last in first order )

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Stack s=new Stack();
        s.push("A");
        s.push(10);
        s.push("aaa");
    }
}
```

```
        System.out.println(s);
        s.pop();
        System.out.println(s);
        System.out.println(s.search("A"));
    }
}
```

**Cursors:-**

The main purpose of the constructors is to retrieve the data from the collection objects.

There are three types of cursors present in the java language.

1. Enumeration
2. Iterator
3. ListIterator

**Enumeration:-**

1. It is used for only legacy classes(Vector,Stack)
2. Based on above reason the enumeration cursor is not a universal cursor
3. By using this cursor it is possible to read the data only it not possible to update the data and not possible to delete the data.
4. By using elements method we are getting enumeration object.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        for (int i=0;i<10 ;i++ )
        {
            v.addElement(i);
        }
        System.out.println(v);
        Enumeration e=v.elements();
        while (e.hasMoreElements())
```

```

        {
            Integer i=(Integer)e.nextElement();
            if (i%2==0)
            {
                System.out.println(i);
            }
        }
        System.out.println(v);
    }
}

```

**Iterator:-**

1. it is universal cursor we can apply any type of collection class.
2. By using this it is possible to read the data and remove the data.
3. We can use iterator() method to get the iterator object.

Ex:-

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        for (int i=0;i<10 ;i++ )
        {
            v.addElement(i);
        }
        System.out.println(v);
        Iterator itr=v.iterator();
        while (itr.hasNext())
        {
            Integer i=(Integer)itr.next();
            if (i%2==0)
            {
                System.out.println(i);
            }
            else
                itr.remove();
        }
        System.out.println(v);
    }
}

```

**ListIterator:-**

1. It is applicable for only list type of objects.
2. By using this it is possible to read the data update the data and delete data also.
3. By using listiterator() method we are getting ListIterator object

Ex:-

```

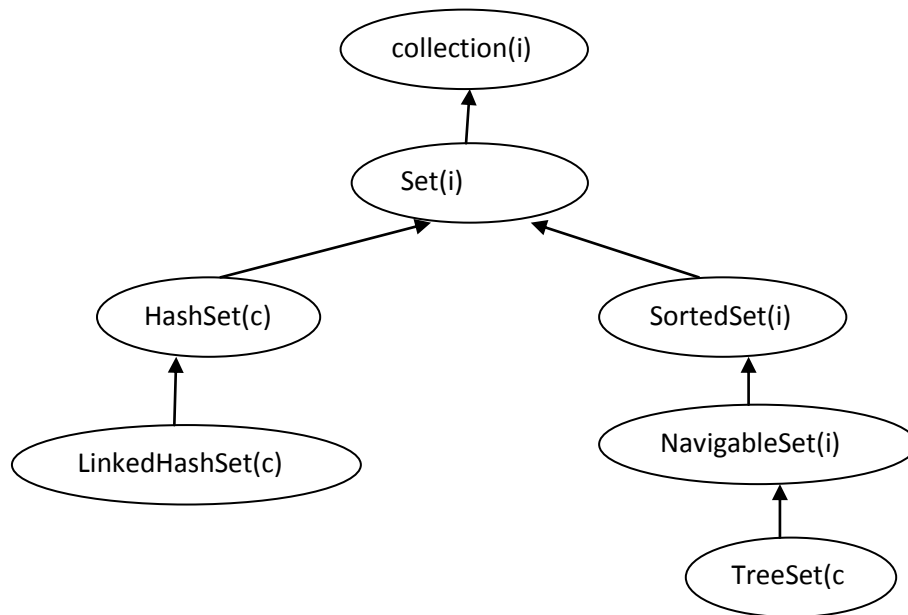
import java.util.*;

```

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

        ListIterator litr=v.listIterator();
        while (litr.hasNext())
        {
            Integer i=(Integer)litr.next();
            if (i==0)
            {
                litr.add("veeru");
            }
            if (i==5)
            {
                litr.set("sambha");
            }
            if (i==9)
            {
                litr.remove();
            }
        }
        System.out.println(v);
    }
}
```



**HashSet:-**

1. Introduced in 1.2 v
2. Duplicate objects are not allowed if we are trying to insert duplicate values then we won't get any compilation errors and won't get any Execution errors simply add method return false.
3. Null insertion is possible
4. Heterogeneous objects are allowed
5. The underlying data structure is hashTable.
6. Insertion order is not preserved.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        HashSet h=new HashSet();
        h.add("a");
        h.add("a");
        h.add("aaaa");
        h.add(10);
        h.add(null);
        System.out.println(h);
    }
}
```

**LinkedHashSet:-**

1. Introduced in 1.4 v
2. Duplicate objects are not allowed if we are trying to insert duplicate values then we won't get any compilation errors and won't get any Execution errors simply add method return false.

3. Null insertion is possible
4. Heterogeneous objects are allowed
5. The underlying data structure is linkedList & hashTable.
6. Insertion order is preserved.
7. It is a child class of HashSet.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        LinkedHashSet h=new LinkedHashSet();
        h.add("a");
        h.add("a");
        h.add("aaaa");
        h.add(10);
        h.add(null);
        System.out.println(h);
    }
}
```

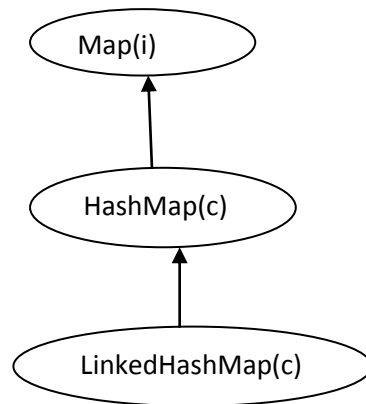
**TreeSet:-**

1. The underlying data Structure is BalancedTree.
2. Insertion order is not preserved it is based some sorting order.
3. Heterogeneous data is not allowed.
4. Duplicate objects are not allowed
5. Null insertion is possible only once.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        TreeSet t=new TreeSet();
        t.add(50);
        t.add(20);
        t.add(40);
        t.add(10);
        t.add(30);
        System.out.println(t);
        SortedSet s1=t.headSet(50);
        System.out.println(s1);//[10,20,30,40]
        SortedSet s2=t.tailSet(30);
        System.out.println(s2);//[30,40,50]
        SortedSet s3=t.subSet(20,50);
        System.out.println(s3);//[20,30,40]
    }
}
```

```
}
```

**Map interface:-****Map:-**

1. Map is a child interface of collection.
2. Up to now we are working with single object and single value whereas in the map collections we are working with two objects and two elements.
3. The main purpose of the collection is to compare the key value pairs and to perform necessary operation.
4. The key and value pairs we can call it as map Entry.
5. Both keys and values are objects only.
6. In entire collection keys can't be duplicated but values can be duplicate.

**HashMap:-**

1. It is used to hold key value pairs.
2. Underlying data structure is HashTable.
3. Duplicate keys are not allowed but values can be duplicated.
4. Insertion order is not preserved.
5. Null is allowed for key (only once) and allows for values any number of times.
6. Every method is non-synchronized so multiple threads can operate at a time hence performance is high.

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        HashMap h=new HashMap();
        h.put("sambha",100);
        h.put("veeru",100);
        h.put("durga",100);
        System.out.println(h);
    }
}
```

```
        Set s=h.keySet();
        System.out.println(s);
        Collection c=h.values();
        System.out.println(c);
        Set s1=h.entrySet();
        System.out.println(s1);

        Iterator itr=s1.iterator();
        while (itr.hasNext())
        {
            Map.Entry m1=(Map.Entry)itr.next();
            System.out.println(m1.getKey()+"-----"+m1.getValue());
            if (m1.getKey().equals("sambha"))
            {
                m1.setValue("gayan TeamLead");
            }
        }
        System.out.println(s1);
    }
}
```

**HashTable:-**

1. It is a legacy class introduced in the 1.0 version.
2. Every method is synchronized hence only one thread is allow to access.
3. The performance of the application is low.
4. Null insertion is not possible if we are trying to insert null values we are getting NullPointerException.

Ex:-

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Hashtable h=new Hashtable();
        h.put("hyd",100);
        h.put("bang",200);
        h.put("pune",300);
        System.out.println(h);
        System.out.println(h.contains(300));//true
        System.out.println(h.containsValue(500));//false
        Collection c=h.values();
    }
}
```

```
        System.out.println(c);
        Set c1=h.keySet();
        System.out.println(c1);
    }
}
```

**LinkedHashMap:-**

1. It used to hold key value pairs
2. Underlying data Structure is HashTable & LinkedList.
3. Duplicate keys are not allowed but values can be duplicated.
4. Insertion order is preserved.

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        LinkedHashMap h=new LinkedHashMap();
        h.put("sambha",100);
        h.put("veeru",100);
        h.put("durga",100);
        System.out.println(h);
        Set s=h.keySet();
        System.out.println(s);
        Collection c=h.values();
        System.out.println(c);
        Set s1=h.entrySet();
        System.out.println(s1);

        Iterator itr=s1.iterator();
        while (itr.hasNext())
        {
            Map.Entry m1=(Map.Entry)itr.next();
            System.out.println(m1.getKey()+"-----"+m1.getValue());
            if (m1.getKey().equals("sambha"))
            {
                m1.setValue("gayan TeamLead");
            }
        }
        System.out.println(s1);
    }
}
```

## Networking

### Introduction to networking:-

- 1) The process of connecting the resources (computers) together to share the data is called networking.
- 2) Java.net is package it contains number of classes by using that classes we are able to connection between the devices (computers) to share the information.
- 3) Java.net package provide support for the TCP (Transmission Control Protocol),UDP(user data gram protocol) protocols.

- 4) In the network we are having to components
  - a. Sender
  - b. Receiver

**Sender/source:** - the person who is sending the data is called sender.

**Receiver/destination:-** the person who is receiving the data is called receiver.

In the network one system can acts as a sender as well as receiver.

- 5) In the networking terminology everyone says client and server.
  - I. Client
  - II. Server

**Client:-** the person who is sending the request and taking the response is called client.

**Server:-** the person who is taking the request and sending the response is called server.

### Categories of network:-

We are having two types of networks

- 1) Per-to-peer network.
- 2) Client-server network.

### Client-server:-

In the client server architecture always client system behaves as a client and server system behaves as a server.

### Peer-to-peer:-

In the peer to peer client system sometimes behaves as a server, server system sometimes behaves like a client the roles are not fixed.

### Types of networks:-

#### Intranet:-

It is also known as a private network. To share the information in limited area range(within the organization) then we should go for intranet.

#### Internet:-

It is also known as public networks. Where the data maintained in a centralized server hence we are having more sharability. And we can access the data from anywhere else.

#### Extranet:-

This is extension to the private network means other than the organization , authorized persons able to access.

**The frequently used terms in the networking:-**

- 1) IP Address
- 2) URL(Uniform Resource Locator)
- 3) Protocol
- 4) Port Number
- 5) MAC address.
- 6) Connection oriented and connection less protocol
- 7) Socket.

**Protocol:-**

Protocol is a set of rules followed by the every computer present in the network this is useful to send the data physically from one place to another place in the network.

- TCP(Transmission Control Protocol)(connection oriented protocol)
- UDP (User Data Gram Protocol)(connection less protocol)
- Telnet
- SMTP(Simple Mail Transfer Protocol)
- IP (Internet Protocol)

**IP Address:-**

- 1) IP Address is a unique identification number given to the computer to indentify it uniquely in the network.
- 2) The IP Address is uniquely assigned to the computer it is not duplicated.
- 3) The IP Address range is 0-255 if we are giving the other than this range that is not allowed.
- 4) We can identify the particular computer in the network with the help of IP Address.
- 5) The IP Address contains four digit number
  - a. 125.0.4.255----good
  - b. 124.654.5.6-----bad
  - c. 1.2.3.4.5.6-----bad
- 6) Each and every website contains its own IP Address we can access the sites through the names otherwise IP Address.

Site Name       :-       www.google.com

IP Address       :-       74.125.224.72

Ex:-

```
import java.net.*;
import java.io.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("please enter site name");
        String sitename=br.readLine();
```

```

        InetAddress in=InetAddress.getByName(sitename);
        System.out.println("the ip address is:"+in);

    }
}

```

Compilation :- javac Test.java

Execution :- java Test

[www.google.com](http://www.google.com)

The IP Address is:www.google.com/74.125.236.176

java Test

[www.yahoo.com](http://www.yahoo.com)

The IP Address is: [www.yahoo.com/ 106.10.139.246](http://www.yahoo.com/106.10.139.246)

Java Test

Please press enter key then we will get IP Address of the system.

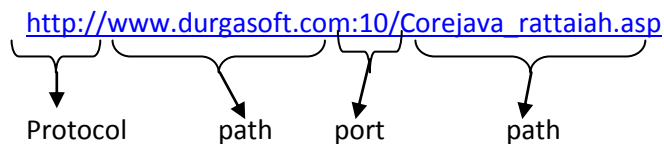
The IP Address is : local host/we are getting IP Address of the system

#### Note:-

If the internet is not available we are getting java.net.UnKnownHostException.

#### URL(Uniform Resource Locator):-

- 1) URL is a class present in the java.net package.
- 2) By using the URL we are accessing some information present in the world wide web.
- 3) Example of URL is:-



The URL contains information like

- a. Protocol to use **http://**
  - b. Server name/IP address [www.durgasoft.com](http://www.durgasoft.com)
  - c. Port number of the particular application and it is optional(:10)
  - d. File name or directory name **Corejava\_rattaiah.asp**
- 4) To create the object for URL we have to use the following syntax
- a. URL obj=new URL(String protocol, String host, int port, String path);
  - b. URL obj=new URL(String protocol, String host, String path);

#### Ex:-

```

import java.net.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        URL url=new URL("http://www.durgasoft.com:10/index.html");
        System.out.println("protocol is:"+url.getProtocol());
        System.out.println("host name is:"+url.getHost());
    }
}

```



```
        System.out.println("port number is:"+url.getPort());
        System.out.println("path is:"+url.getPath());
        System.out.println(url);
    }
}
```

**Communication using networking :-**

In the networking it is possible to do two types of communications.

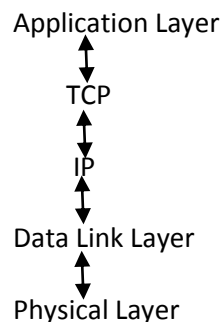
- 1) Connection oriented(TCP/IP communication)
- 2) Connection less(UDP Communication)

**Connection Oriented:-**

- a) In this type of communication we are using combination of two protocols TCP,IP.
- b) In this type of communication the main purpose of the TCP is transferred in the form of packets between the source and destination. And the main purpose of the IP is finding address of a particular system.

To achieve the following communication the java peoples are provided the following classes.

- a. Socket
- b. ServerSocket

**Layers of the TCP/IP connection.****Application Layer:-**

Takes the data from the application and sends it to the TCP layer.

**TCP Protocol:-**

it will take the data which is coming from Application Layer and divides in to small units called Packets. Then transfer those packets to the next layer called IP. The packet contains group of bytes of data.

**IP:-**

It will take the packets which is coming from TCP and prepare envelop called 'frames' hence the frame contains the group of packets. Then it will identify the particular target machine on the basis of the IP address and sent that frames to the physical layer.

**Physical Layer:-**

Based on the physical medium it will transfer the data to the target machine.

**Connection Less :- (UDP)**

- 1) UDP is a protocol by using this protocol we are able to send data without using Physical Connection.
- 2) This is a light weight protocol because no need of the connection between the client and server .
- 3) This is very fast communication compare to the TCP/IP communication.
- 4) This protocol not sending the data inn proper order there may be chance of missing the data.
- 5) This communication used to send the Audio and Video data if some bits are lost but we are able to see the video and images we are getting any problems.

To achieve the UDP communication the java peoples are provided the fallowing classes.

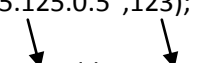
1. DataGramPacket.
2. DataGramSocket.

**Socket:-**

- 1) Socket is used to create the connection between the client and server.
- 2) Socket is nothing but a combination of IP Address and port number.
- 3) The socket is created at client side.
- 4) Socket is class present in the java.net package
- 5) It is acting as a communicator between the client and server.
- 6) Whenever if we want to send the data first we have to create a socket that is acts as a medium.

Create the socket

- 1) Socket s=new Socket(int IPAddress, int portNumber);
  - a. Socket s=new Socket("125.125.0.5",123);



- 2) Socket s=new Socket(String HostName, int PortNumber);
  - a. Socket s=new Socket(Durgasoft,123);

**Client.java:-**

```
import java.net.*;
import java.io.*;
class Client
{
    public static void main(String[] args)throws Exception
    {
        Socket s=new Socket("localhost",5555);
        String str="ratan from client";
        OutputStream os=s.getOutputStream();
        PrintStream ps=new PrintStream(os);
        ps.println(str);

        InputStream is=s.getInputStream();
```

```
        BufferedReader br=new BufferedReader(new InputStreamReader(is));
        String str1=br.readLine();
        System.out.println(str1);
    }
}

Server.java:-
import java.io.*;
import java.net.*;
class Server
{
    public static void main(String[] args) throws Exception
    {
        //to read the data from client
        ServerSocket ss=new ServerSocket(5555);
        Socket s=ss.accept();

        System.out.println("connection is created ");
        InputStream is=s.getInputStream();

        BufferedReader br=new BufferedReader(new InputStreamReader(is));
        String data=br.readLine();
        System.out.println(data);

        //write the data to the client
        data=data+"this is from server";

        OutputStream os=s.getOutputStream();
        PrintStream ps=new PrintStream(os);
        ps.println(data);
    }
}
```

**Java.awt package**

1. Java.awt is a package it will provide very good environment to develop graphical user interface applications.
2. AWT means (Abstract Window Toolkit). AWT is used to prepare the components but it is not providing any life to that components means by using AWT it is possible to create a static components.
3. To provide the life to the static components we need to depends upon some other package is called java.awt.event package.
4. This application not providing very good look and feel hence the normal users facing problem with these types of applications.
5. By using AWT we are preparing application these applications are called console based or CUI application.

**Note**

Java.awt package is used to prepare static components.

Java.awt.event package is used to provide the life to the static components.

**GUI(graphical user interface):-**

1. It is a mediator between end user and the program.
2. AWT is a package it will provide very good predefined support to design GUI applications.

**component :-**

Component is an object which is displayed pictorially on the screen.

Ex:-

Button,Label,TextField.....etc

**Container:-**

Container is a GUI component, it is able to accommodate all other GUI components.

Ex:-

Frame,Applet.

**Event:-**

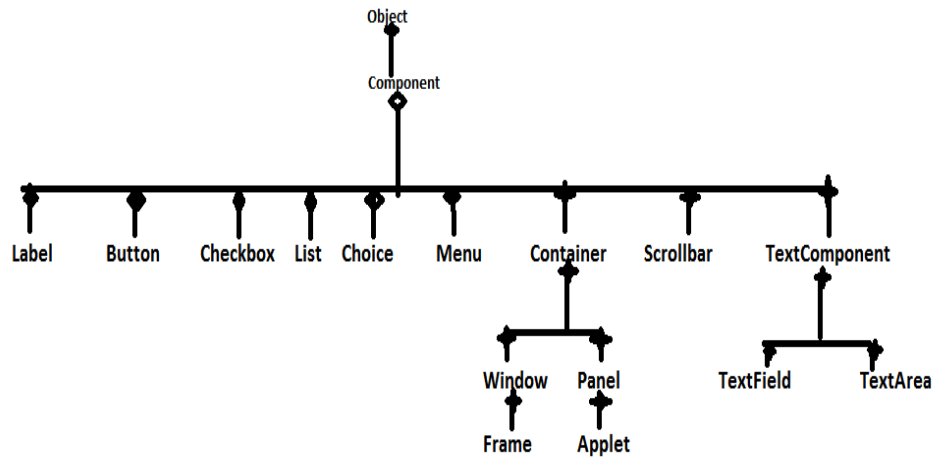
The event nothing but a action generated on the component or the change is made on the state of the object.

Ex:-

Button clicked, Checkboxchecked, Itemselected in the list, Scrollbar scrolled horizontal/vertically.

**Classes of AWT:-**

The classes present in the AWT package.

**Frame:-**

- 1) Frame is a class which is present in java.awt package.
- 2) Frame is a Basic component in AWT, because all the components displayed in a Frame.
- 3) We are displaying pictures on the Frame.
- 4) It is possible to display some text on the Frame.

Based on the above reasons the frame will become basic component in AWT.

**Constructors:-**

- \* create a Frame class object.  
`Frame f=new Frame();`
- \* create a Frame class object and pass file  
`Frame f=new Frame("MyFrame");`
- \* Take a subclass to the Frame and create object to the subclass.  
`class MyFrame extends Frame`  
`MyFrame f=new MyFrame();`

**Characteristics of the Frame:-**

- 1) When we create a Frame class object the Frame will be created automatically with the invisible mode. To provide visible mode to following method.

**public void setVisible(boolean b)**

where b==true means visible mode.

where b==false means invisible mode.

Ex: f.setVisible(true);

- 2) When we created a Frame the initial size of the Frame is :  
0 pixel height  
0 pixel width  
So it is not visible to use.
  - To provide particular size to the Frame we have to use following method.

**public void setSize(int width,int height)**

Ex: f.setSize(400,500);

- 3) To provide title to the Frame explicitly we have to use the following method

**public void setTitle(String Title)**

Ex: f.setTitle("MyFrame");

- 4) When we create a Frame, the default background color of the Frame is white. If you want to provide particular color to the Frame we have to use the following method.

**public void setBackground(color c)**

Ex: f.setBackground(Color.red);

\*\*\*\*\*CREATION OF FRMAE\*\*\*\*\*

```
import java.awt.*;
class Demo
{
    public static void main(String[] args)
    {
        //frame creation
        Frame f=new Frame();
        //set visibility
        f.setVisible(true);
        //set the size of the frame
        f.setSize(400,400);
        //set the background
        f.setBackground(Color.red);
        //set the title of the frame
        f.setTitle("myframe");
    }
}
```

```
};
```

\*\*\*CRATION OF FRAME BY TAKING USER DEFINED CLASS\*\*\*\*

```
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        setVisible(true);
        setSize(500,500);
        setTitle("myframe");
        setBackground(Color.red);
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

#### To display text on the screen:-

1. If you want to display some textual message or some graphical shapes on the Frame then we have to override paint(), which is present in the Frame class.

```
public void paint(Graphics g)
```

2. To set a particular font to the text, we have to use Font class present in java.awt package

```
Font f=new Font(String type,int style,int size);
```

```
Ex: Font f= new Font("arial",Font.Bold,30);
```

Ex :-

```
import java.awt.*;
class Test extends Frame
{
    public static void main(String[] args)
    {
        Test t=new Test();
        t.setVisible(true);
        t.setSize(500,500);
        t.setTitle("myframe");
        t.setBackground(Color.red);
    }
    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.ITALIC,25);
```

```

        g.setFont(f);
        g.drawString("hi ratan how r u",100,100);
    }
}

```

**Note:-**

1. When we create a MyFrame class constructor, jvm executes MyFrame class constructor just before this JVM has to execute Frame class zero argument constructor.
2. In Frame class zero argument constructor repaint() method will be executed, it will access predefined Frame class paint() method. But as per the requirement overriding paint() method will be executed.
3. Therefore the paint() will be executed automatically at the time of Frame creation.

**Preparation of the components:-****Label: -**

- 1) Label is a constant text which is displayed along with a TextField or TextArea.
- 2) Label is a class which is present in java.awt package.
- 3) To display the label we have to add that label into the frame for that purpose we have to use add() method present in the Frame class.

**Constructor:-**

```

Label l=new Label();
Label l=new Label("user name");

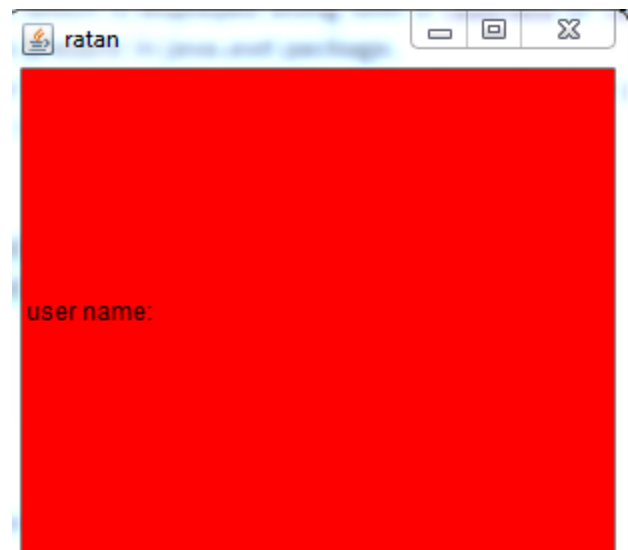
```

**Ex :-**

```

import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);
        Label l=new Label("user name:");
        f.add(l);
    }
}

```

**TextField:-**

- 1) TextField is an editable area.
  - 2) In TextField we are able to provide single line of text.
  - 3) Enter Button doesn't work on TextField. To add TextField into the Frame we have to use add() method.
1. To set Text to the textarea we have to use the following method.



```
t.setText("Durga");
```

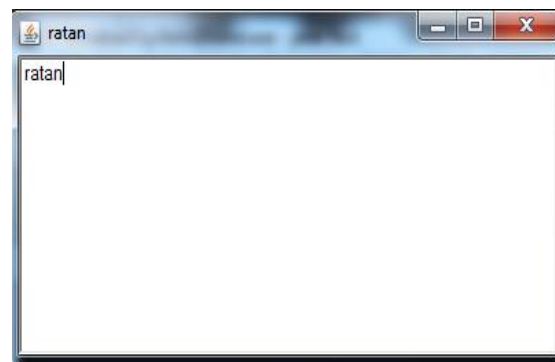
2. To get the text form the TextArea we have to use following method.  
String s=t.getText();
3. To append the text into the TextArea.  
t.appendText("ratan");

**Constructor:-**

```
TextFiled tx=new TextFiled();  
TextField tx=new TextField("ratan");
```

**Ex :-**

```
import java.awt.*;  
class Test  
{  
    public static void main(String[] args)  
    {  
        Frame f=new Frame();  
        f.setVisible(true);  
        f.setTitle("ratan");  
        f.setBackground(Color.red);  
        f.setSize(400,500);  
        //TextField tx=new TextField(); empty TextField  
        TextField tx=new TextField("ratan");  
        //TextField with data  
        f.add(tx);  
    }  
}
```

**TextArea:-**

- 1) TextArea is a class present in java.awt.package.
- 2) TextArea is a Editable Area. Enter button will work on TextArea.
- 3) To add the TextArea into the frame we have to use the add()

**Construction:-**

```
TextArea t=new TextArea();  
TextArea t=new TextArea(int rows,int columns);
```

4. To set Text to the textarea we have to use the following method.  
t.setText("Durga");
5. To get the text form the TextArea we have to use following method.  
String s=t.getText();

6. To append the text into the TextArea.  
t.appendText("ratan");

```
import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);
        f.setLayout(new FlowLayout());
        Label l=new Label("user name:");
        TextArea tx=new TextArea(4,10);//4 character height 10 character width
        tx.appendText("ratan");
        tx.setText("aruna");
        System.out.println(tx.getText());
        f.add(l);
        f.add(tx);
    }
}
```



**Choice:-**

- 1) Choice is a class present in java.awt package.
- 2) List is allows to select multiple items but choice is allow to select single Item.

**Constructor:-**

Choice ch=new Choice();

**Methods :-**

1. To add items to the choice we have to use following method.  
ch.add("HYD");  
ch.add("Chennai");  
ch.add("BANGALORE");
2. To remove item from the choice based on the string.  
ch.remove("HYD");  
ch.remove("BANGALORE");

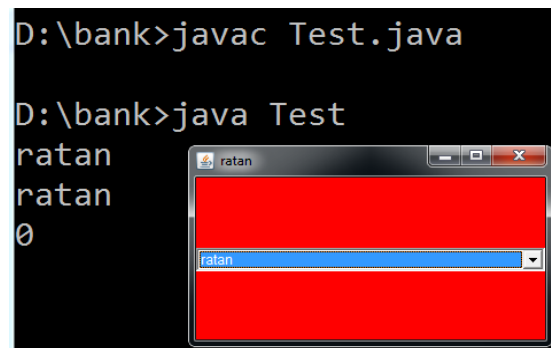
3. To remove the item based on the index position  
`ch.remove(2);`
4. To remove the all elements  
`ch.removeAll();`
5. To inset the data into the choice based on the particular position.  
`ch.insert(2,"ratan");`
6. To get selected item from the choice we have to use following method.  
`String s=ch.getSelectedItem();`
7. To get the selected item index number we have to use following method  
`int a=ch.getSelectedIndex();`

ex:-

```
import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);

        Choice ch=new Choice();
        ch.add("c");
        ch.add("cpp");
        ch.add("java");
        ch.add(".net");
        ch.remove(".net");
        ch.remove(0);
        ch.insert("ratan",0);
        f.add(ch);

        System.out.println(ch.getItem(0));
        System.out.println(ch.getSelectedItem());
        System.out.println(ch.getSelectedIndex());
        //ch.removeAll();
    }
}
```



#### List:-

- 1) List is a class it is present in java.awt.package
- 2) List is providing list of options to select. Based on your requirement we can select any number of elements. To add the List to the frame we have to use add() method.

#### CONSTRUCTOR:-

- 1) List l=new List();

It will creates the list by default size is four elements. And it is allow selecting the only one item at a time.

2) List l=new List(3);

It will display the three items size and it is allow selecting the only single item.

3) List l=new List(5,true);

It will display the five items and it is allow selecting the multiple items.

#### Methods:-

1. To add the elements to the List we have to use following method.

```
l.add("c");  
l.add("cpp");  
l.add("java");  
l.add("ratan",0);
```

2. To remove element from the List we have to use following method.

```
l.remove("c");  
l.remove(2);
```

3. To get selected item from the List we have to use following method.

```
String x=l.getSelectedItem();
```

4. To get selected items from the List we have to use following method.

```
String[] x=s.getSelectedItems()
```

Ex:-

```
import java.awt.*;  
class Test  
{  
    public static void main(String[] args)  
    {  
        Frame f=new Frame();  
        f.setVisible(true);  
        f.setTitle("ratan");  
        f.setBackground(Color.red);  
        f.setSize(400,500);  
        f.setLayout(new FlowLayout());  
  
        List l=new List(4,true);  
        l.add("c");  
        l.add("cpp");  
        l.add("java");  
        l.add(".net");  
        l.add("ratan");  
        l.add("arun",0);  
        l.remove(0);  
        f.add(l);  
        System.out.println(l.getSelectedItem());  
    }  
}
```

**Checkbox:-**

- 1) Checkbox is a class present in java.awt package.
- 2) The user can select more than one checkbox at a time. To add the checkbox to the frame we have to use add() method.

**Constructor:-**

- 1) `Checkbox cb1=new Checkbox();`  
`cb1.setLabel("BTECH");`
- 2) `Checkbox cb1=new Checkbox("MCA");`
- 3) `Checkbox cb3=new Checkbox("BSC",true);`

**Methods:-**

1. To set a label to the CheckBox explicitly and to get label from the CheckBox we have to use the following method.  
`cb.setLabel("BSC");`
2. To get the label of the checkbox we have to use following method.  
`String str=cb.getLabel();`
3. To get state of the CheckBox and to set state to the CheckBox we have to use following method.  
`Boolean b=ch.getState();`

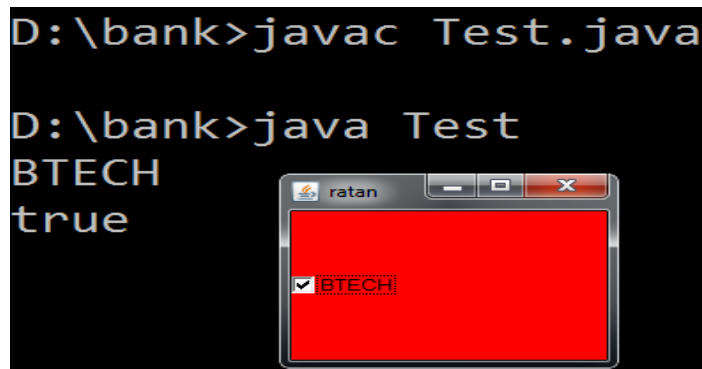
Ex:-

```
import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);
        Checkbox cb1=new Checkbox("BTECH",true);
        f.add(cb1);
        System.out.println(cb1.getLabel());
    }
}
```

```

        System.out.println(cb1.getState());
    }
}

```



### RADIO BUTTON:-

- 1) AWT does not provide any predefined support to create RadioButtons.
- 2) It is possible to select Only item is selected from group of item. To add the RadioButton to the frame we have to use add() method.

By using two classes we create Radio Button those are

- a)CheckBoxgroup
- b)CheckBox

step 1:- Create CheckBox group object.

```
CheckBoxGroup cg=new CheckBoxGroup();
```

step 2:- pass Checkbox object to the CheckboxGroup class then the radio buttons are created.

```

CheckBox cb1=new CheckBox("male",cg,false);
CheckBox cb2=new CheckBox("female",cg,false);

```

### Methods:-

- 1) To set status and to get status we have to use setState() and getState() methods.

```

String str=Cb.getState();
Cb.setState();

```

- 2) To get Label and to set Label we have to use following methods.

```

String str=getLabel()
setLabel("female").

```

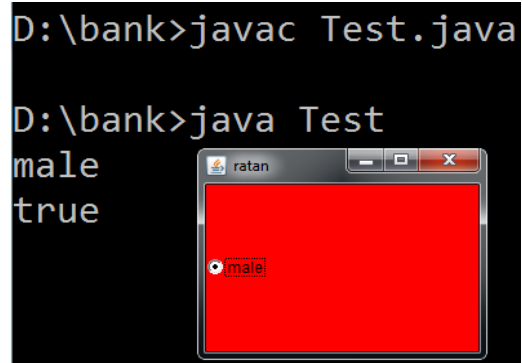
Ex:-

```

import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);
    }
}

```

```
CheckboxGroup cg=new CheckboxGroup();
Checkbox cb1=new Checkbox("male",cg,true);
f.add(cb1);
System.out.println(cb1.getLabel());
System.out.println(cb1.getState());
}
```

**Layout Managers:-**

```
import java.awt.*;
class Test
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setTitle("ratan");
        f.setBackground(Color.red);
        f.setSize(400,500);
        Label l1=new Label("user name:");
        TextField tx1=new TextField();
        Label l2=new Label("password:");
        TextField tx2=new TextField();
        Button b=new Button("login");
        f.add(l1);
        f.add(tx1);
        f.add(l2);
        f.add(tx1);
        f.add(b);
    }
}
```

**Event delegation model:-**

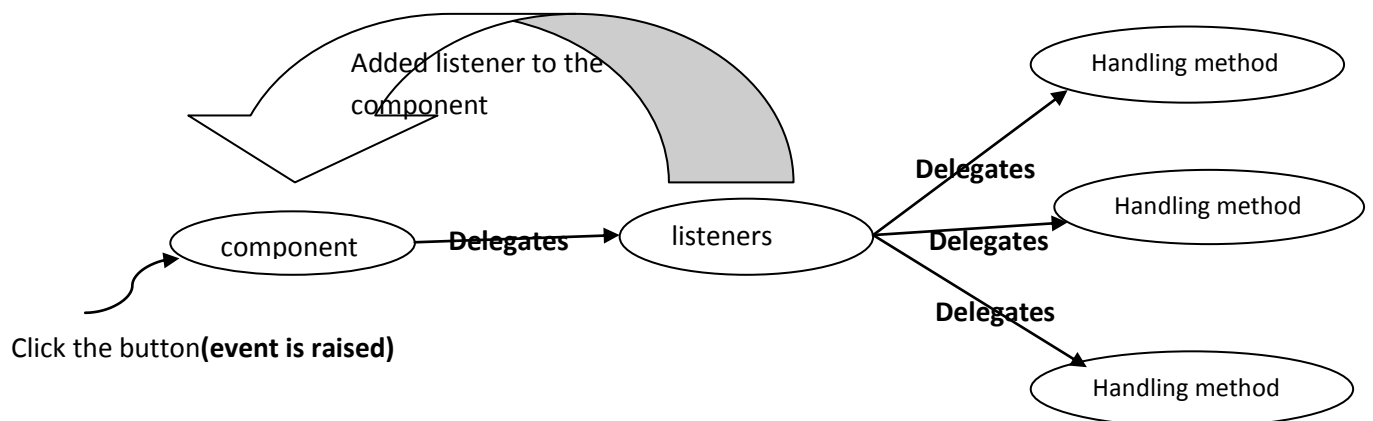
1. When we create a component the components visible on the screen but it is not possible to perform any action for example button.
2. Whenever we create a Frame it can be minimized and maximized and resized but it is not possible to close the Frame even if we click on Frame close Button.
3. The Frame is a static component so it is not possible to perform actions on the Frame.
4. To make static component into dynamic component we have to add some actions to the Frame.
5. To attach actions to the Frame component we need event delegation model.

**Whenever we click on button no action will be performed clicking like this is called event.**

**Event:** - Event is nothing but a particular action generated on the particular component.

1. When an event generates on the component the component is unable to respond because component can't listen the event.
2. To make the component listen the event we have to add listeners to the component.
3. Wherever we are adding listeners to the component the component is able to respond based on the generated event.
4. A listener is a interface which contain abstract methods and it is present in java.awt.event package
5. The listeners are different from component to component.

A component delegate event to the listener and listener is designates the event to appropriate method by executing that method only the event is handled. This is called Event Delegation Model.

**Note: -**

To attach a particular listener to the Frame we have to use following method

**Public void AddxxxListener(xxxListener e)**

Where xxx may be ActionListener,windowListener

The Appropriate Listener for the Frame is "windowListener"



**ScrollBar:-**

1. ScrollBar is a class present in the java.qwt.package
2. By using ScrollBar we can move the Frame up and down.

```
ScrollBar s=new ScrollBar(int type)
```

Type of scrollbar

1. VERTICAL ScrollBar
2. HORIZONTAL ScrollBar

To create a HORIZONTAL ScrollBar:-

```
ScrollBar sb=new ScrollBar(ScrollBar.HORIZONTAL);
```

To get the current position of the scrollbar we have to use the following method.

```
public int getValue()
```

To create a VERTICAL ScrollBar:-

```
ScrollBar sb=new ScrollBar(ScrollBar.VERTICAL);
```

**Appropriate Listeners for Components:-**

GUI Component	Event Name	Listner Name	Lisener Methods
1.Frame	Window Event	Window Listener	1.Public Void WindowOpened(WindowEvent e) 2.Public Void WindowActivated(WindowEvent e) 3.Public Void WindowDeactivated(WindowEvent e) 4.Public Void WindowClosing(WindowEvent e) 5.Public Void WindowClosed(WindowEvent e) 6.Public Void WindowIconified(WindowEvent e) 7.Public Void WindowDeiconified(WindowEvent e)
2.Textfield	ActionEvent	ActionListener	1.Public Void Actionperformed(ActionEvent ae)
3.TextArea	ActionEvent	ActionListener	1.Public Void Actionperformed(ActionEvent ae)
4.Menu	ActionEvent	ActionListener	1.Public Void Actionperformed(ActionEvent ae)
5.Button	ActionEvent	ActionListener	1.Public Void Actionperformed(ActionEvent ae)
6.Checkbox	ItemEvent	ItemListener	1.Public Void ItemStateChanged(ItemEvent e)
7.Radio	ItemEvent	ItemListener	1.Public Void ItemStateChanged(ItemEvent e)
8.List	ItemEvent	ItemListener	1.Public Void ItemStateChanged(ItemEvent e)
9.Choice	ItemEvent	ItemListener	1.Public Void ItemStateChanged(ItemEvent e)
10.Scrollbar	AdjustmentEvent	AdjustmentListener	1.Public Void AdjustementValueChanged

(AdjustementEvent e)

11.Mouse	MouseEvent	MouseListener	1.Public Void MouseEntered(MouseEvent e) 2.Public Void MouseExited(MouseEvent e) 3.Public Void MousePressed(MouseEvent e) 4.Public Void MouseReleased(MouseEvent e) 5.Public Void MouseClicked(MouseEvent e)
12.Keyboard	KeyEvent	KeyListener	1.Public Void KeyTyped(KeyEvent e) 2.Public Void KeyPressed(KeyEvent e) 3.Public Void KeyReleased(KeyEvent e)

\*\*\*PROVIDING CLOSING OPTION TO THE FRAME\*\*\*\*

```

import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setSize(400,500);
        this.setVisible(true);
        this.setTitle("myframe");
        this.addWindowListener(new myclassimpl());
    }
}

class myclassimpl implements WindowListener
{
    public void windowActivated(WindowEvent e)
    {
        System.out.println("window activated");
    }
    public void windowDeactivated(WindowEvent e)
    {
        System.out.println("window deactivated");
    }
    public void windowIconified(WindowEvent e)
    {
        System.out.println("window iconified");
    }
    public void windowDeiconified(WindowEvent e)
    {
        System.out.println("window deiconified");
    }
    public void windowClosed(WindowEvent e)
    {
        System.out.println("window closed");
    }
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}

```

```
        public void windowOpened(WindowEvent e)
        {
            System.out.println("window Opened");
        }
    };
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
**PROVIDING CLOSEING OPTION TO THE FRAME**

import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setBackground(Color.red);
        this.setTitle("rattaiah");
        this.addWindowListener(new Listenerimpl());
    }
};

class Listenerimpl extends WindowAdapter
{
    public void windowClosing(WindowEvent we)
    {
        System.exit(0);
    }
};
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

Note :- by using WindowAdaptor class we can close the frame. Internally WindowAdaptor class implements WindowListener interface. Hence WindowAdaptor class contains empty implementation of abstract methods.

```
****PROVIDING CLOSEING OPTION THE FRAME****

import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setBackground(Color.red);
        this.setTitle("rattaiah");
        this.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
}
class FrameEx
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};

***WRITE SOME TEXT INTO THE FRAME*****

import java.awt.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setBackground(Color.red);
```

```

        this.setTitle("rattaiah");
    }
    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,20);
        g.setFont(f);
        this.setForeground(Color.green);
        g.drawString("HI BTECH ",100,100);
        g.drawString("good boys &",200,200);
        g.drawString("good girls",300,300);
    }
}
class FrameEx
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
-----
*****LAYOUT MACHANISUMS  FLOWLAYOUT*****
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame
{
    Label l1,l2;
    TextField tx1,tx2;
    Button b;
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(340,500);
        this.setBackground(Color.green);
        this.setTitle("rattaiah");

        l1=new Label("user name:");
        l2=new Label("password:");
        tx1=new TextField(25);
        tx2=new TextField(25);

        b=new Button("login");
        tx2.setEchoChar('*');
        this.setLayout(new FlowLayout());

        this.add(l1);
        this.add(tx1);
        this.add(l2);
        this.add(tx2);
    }
}

```

```
        this.add(b);
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

-----

\*\*\*\*\*BORDERLAYOUT\*\*\*\*\*

```
import java.awt.*;
class MyFrame extends Frame
{
    Button b1,b2,b3,b4,b5;
    MyFrame()
    {
        this.setBackground(Color.green);
        this.setSize(400,400);
        this.setVisible(true);
        this.setLayout(new BorderLayout());

        b1=new Button("Boys");
        b2=new Button("Girls");
        b3=new Button("management");
        b4=new Button("Teaching Staff");
        b5=new Button("non-teaching staff");

        this.add("North",b1);
        this.add("Center",b2);
        this.add("South",b3);
        this.add("East",b4);
        this.add("West",b5);
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

-----

\*\*\*\*\*CardLayout\*\*\*\*\*

```
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setSize(400,400);
        this.setVisible(true);
        this.setLayout(new CardLayout());
        Button b1=new Button("button1");
        Button b2=new Button("button2");
        Button b3=new Button("button3");
        Button b4=new Button("button4");
        Button b5=new Button("button5");

        this.add("First Card",b1);
        this.add("Second Card",b2);
        this.add("Thrid Card",b3);
        this.add("Fourth Card",b4);
        this.add("Fifth Card",b5);
    }
}

class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};

*****GRIDLAYOUT*****
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setTitle("rattaiah");
        this.setBackground(Color.red);
        this.setLayout(new GridLayout(4,4));
        for (int i=0;i<10;i++)
        {
            Button b=new Button(""+i);
            this.add(b);
        }
    }
}
```

```
};  
class Demo  
{  
    public static void main(String[] args)  
    {  
        MyFrame f=new MyFrame();  
    }  
};
```

\*\*\*\*\*ACTIONLISTENER\*\*\*\*\*

```
import java.awt.*;  
import java.awt.event.*;  
class myframe extends Frame implements ActionListener  
{  
    TextField tx1,tx2,tx3;  
    Label l1,l2,l3;  
    Button b1,b2;  
    int result;  
    myframe()  
    {  
        this.setSize(250,400);  
        this.setVisible(true);  
        this.setLayout(new FlowLayout());  
        l1=new Label("first value");  
        l2=new Label("second value");  
        l3=new Label("result");  
  
        tx1=new TextField(25);  
        tx2=new TextField(25);  
        tx3=new TextField(25);  
  
        b1=new Button("add");  
        b2=new Button("mul");  
  
        b1.addActionListener(this);  
        b2.addActionListener(this);  
        this.add(l1);  
        this.add(tx1);  
        this.add(l2);  
        this.add(tx2);  
        this.add(l3);  
        this.add(tx3);  
        this.add(b1);  
        this.add(b2);  
    }  
}
```



```
public void actionPerformed(ActionEvent e)
{
    try{
        int fval=Integer.parseInt(tx1.getText());
        int sval=Integer.parseInt(tx2.getText());

        String label=e.getActionCommand();

        if (label.equals("add"))
        {
            result=fval+sval;
        }

        if (label.equals("mul"))
        {
            result=fval*sval;
        }
        tx3.setText(""+result);
    }
    catch(Exception ee)
    {
        ee.printStackTrace();
    }
}

};

class Demo
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
```

---

\*\*\*\*\* LOGIN STATUS\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;

class MyFrame extends Frame implements ActionListener
{
    Label l1,l2;
    TextField tx1,tx2;
    Button b;
    String status="";
    MyFrame()
    {
```

```
        setVisible(true);
        setSize(400,400);
        setTitle("girls");
        setBackground(Color.red);

        l1=new Label("user name:");
        l2=new Label("password:");
        tx1=new TextField(25);
        tx2=new TextField(25);

        b=new Button("login");
        b.addActionListener(this);
        tx2.setEchoChar('*');

        this.setLayout(new FlowLayout());

        this.add(l1);
        this.add(tx1);
        this.add(l2);
        this.add(tx2);
        this.add(b);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String  uname=tx1.getText();
        String upwd=tx2.getText();

        if (uname.equals("durga")&&upwd.equals("dss"))
        {
            status="login success";
        }
        else
        {
            status="login failure";
        }
        repaint();
    }
    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,30);
        g.setFont(f);
        this.setForeground(Color.green);
        g.drawString("Status:----"+status,50,300);
    }
}
```

```
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

---

\*\*\*\*\*MENUITEMS\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;
class MyFrame extends Frame implements ActionListener
{
    String label="";
    MenuBar mb;
    Menu m1,m2,m3;
    MenuItem mi1,mi2,mi3;
    MyFrame()
    {
        this.setSize(300,300);
        this.setVisible(true);
        this.setTitle("myFrame");
        this.setBackground(Color.green);

        mb=new MenuBar();
        this.setMenuBar(mb);

        m1=new Menu("new");
        m2=new Menu("option");
        m3=new Menu("edit");
        mb.add(m1);
        mb.add(m2);
        mb.add(m3);

        mi1=new MenuItem("open");
        mi2=new MenuItem("save");
        mi3=new MenuItem("saveas");

        m1.addActionListener(this);
        m2.addActionListener(this);
        m3.addActionListener(this);

        m1.add(mi1);
        m1.add(mi2);
```

```
        m1.add(mi3);
    }

    public void actionPerformed(ActionEvent ae)
    {
        label=ae.getActionCommand();
        repaint();
    }

    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,25);
        g.setFont(f);
        g.drawString("Selected item....."+label,50,200);
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

---

#### \*\*\*\*\*MOUSELISTENER INTERFACE\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;
class myframe extends Frame implements MouseListener
{
    String[] msg=new String[5];
    myframe()
    {
        this.setSize(500,500);
        this.setVisible(true);
        this.addMouseListener(this);
    }
    public void mouseClicked(MouseEvent e)
    {
        msg[0]="mouse clicked.....("+e.getX()+","+e.getY()+")";
        repaint();
    }

    public void mousePressed(MouseEvent e)
    {
        msg[1]="mouse pressed.....("+e.getX()+","+e.getY()+")";
```

```
        repaint();
    }

    public void mouseReleased(MouseEvent e)
    {
        msg[2]="mouse released.....("+e.getX()+","+e.getY()+")";
        repaint();
    }

    public void mouseEntered(MouseEvent e)
    {
        msg[3]="mouse entered.....("+e.getX()+","+e.getY()+")";
        repaint();
    }

    public void mouseExited(MouseEvent e)
    {
        msg[4]="mouse exited.....("+e.getX()+","+e.getY()+")";
        repaint();
    }

    public void paint(Graphics g)
    {
        int X=50;
        int Y=100;
        for(int i=0;i<msg.length;i++)
        {
            if (msg[i]!=null)
            {
                g.drawString(msg[i],X,Y);
                Y=Y+50;
            }
        }
    }
};

class Demo
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
```

---

\*\*\*\*\*ITEMLISTENER INTERFACE\*\*\*\*\*

import java.awt.\*;

```
import java.awt.event.*;

class myframe extends Frame implements ItemListener
{
    String qual="",gen="";
    Label l1,l2;
    CheckboxGroup cg;
    Checkbox c1,c2,c3,c4,c5;
    Font f;
    myframe()
    {
        this.setSize(300,400);
        this.setVisible(true);
        this.setLayout(new FlowLayout());

        l1=new Label("Qualification: ");
        l2=new Label("Gender: ");

        c1=new Checkbox("BSC");
        c2=new Checkbox("BTECH");
        c3=new Checkbox("MCA");

        cg=new CheckboxGroup();
        c4=new Checkbox("Male",cg,false);
        c5=new Checkbox("Female",cg,true);

        c1.addItemListener(this);
        c2.addItemListener(this);
        c3.addItemListener(this);
        c4.addItemListener(this);
        c5.addItemListener(this);

        this.add(l1);
        this.add(c1);
        this.add(c2);
        this.add(c3);
        this.add(l2);
        this.add(c4);
        this.add(c5);
    }

    public void itemStateChanged(ItemEvent ie)
    {
        if(c1.getState()==true)
        {
            qual=qual+c1.getLabel()+" ";
        }
    }
}
```

```
        if(c2.getState()==true)
        {
            qual=qual+c2.getLabel()+",";
        }
        if(c3.getState()==true)
        {
            qual=qual+c3.getLabel()+",";
        }

        if(c4.getState()==true)
        {
            gen=c4.getLabel();
        }
        if(c5.getState()==true)
        {
            gen=c5.getLabel();
        }
        repaint();
    }

    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,20);
        g.setFont(f);
        this.setForeground(Color.green);
        g.drawString("qualification----->" +qual,50,100);
        g.drawString("gender----->" +gen,50,150);
        qual="";
        gen="";
    }
}

class rc
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
```

---

\*\*\*\*\*KEYLISTENER INTERFACE\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;
class myframe extends Frame
{
    myframe()
```

```
        {
            this.setSize(400,400);
            this.setVisible(true);
            this.setBackground(Color.green);
            this.addKeyListener(new keyboardimpl());
        }
    };
```

class keyboardimpl implements KeyListener

```
{
    public void keyTyped(KeyEvent e)
    {
        System.out.println("key typed "+e.getKeyChar());
    }

    public void keyPressed(KeyEvent e)
    {
        System.out.println("key pressed "+e.getKeyChar());
    }

    public void keyReleased(KeyEvent e)
    {
        System.out.println("key released "+e.getKeyChar());
    }
}
```

class Demo

```
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
```

---

\*\*\*\*\*CHECK LIST AND CHOICE\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;
class myframe extends Frame implements ItemListener
{
    Label l1,l2;
```



```
List l;  
Choice ch;  
String[] tech;  
String city="";  
myframe()  
{  
    this.setSize(300,400);  
    this.setVisible(true);  
    this.setLayout(new FlowLayout());  
  
    l1=new Label("Technologies: ");  
    l2=new Label("City: ");  
  
    l=new List(3,true);  
    l.add("c");  
    l.add("c++");  
    l.add("java");  
    l.addItemListener(this);  
  
    ch=new Choice();  
    ch.add("hyd");  
    ch.add("chennai");  
    ch.add("Banglore");  
    ch.addItemListener(this);  
  
    this.add(l1);  
    this.add(l);  
    this.add(l2);  
    this.add(ch);  
}  
public void itemStateChanged(ItemEvent ie)  
{  
    tech=l.getSelectedItems();  
    city=ch.getSelectedItem();  
    repaint();  
}  
public void paint(Graphics g)  
{  
    Font f=new Font("arial",Font.BOLD,20);  
    g.setFont(f);  
    String utech="";  
    for(int i=0;i<tech.length ;i++ )  
    {  
        utech=utech+tech[i]+" ";  
    }  
    g.drawString("tech:-----"+utech,50,200);  
    g.drawString("city-----"+city,50,300);  
    utech="";  
}
```

```
    }
}
class Demo
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
-----
*****AdjustmentListener*****

import java.awt.*;
import java.awt.event.*;
class myframe extends Frame implements AdjustmentListener
{
    Scrollbar sb;
    int position;

    myframe()
    {
        this.setSize(400,400);
        this.setVisible(true);
        this.setLayout(new BorderLayout());

        sb=new Scrollbar(Scrollbar.VERTICAL);
        this.add("East",sb);

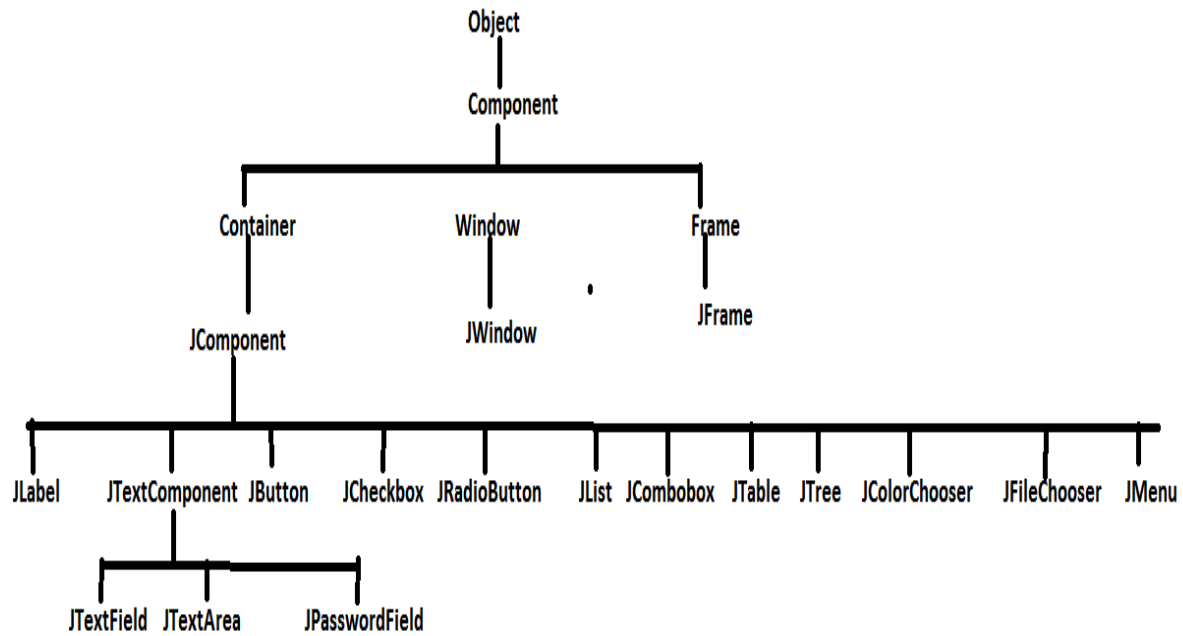
        sb.addAdjustmentListener(this);
    }
    public void adjustmentValueChanged(AdjustmentEvent e)
    {
        position=sb.getValue();
    }
    public void paint(Graphics g)
    {
        g.drawString("position:"+position,100,200);
        repaint();
    }
}
class scrollbarex
{
    public static void main(String[] args)
    {
        myframe f=new myframe();
    }
};
```

## SWINGS

1. Sun Micro Systems introduced AWT to prepare GUI applications.
2. awt components not satisfy the client requirement.
3. An alternative to AWT Netscape Communication has provided set of GUI components in the form of IFC(Internet Foundation Class).
4. IFC also provide less performance and it is not satisfy the client requirement.
5. In the above contest[sun+Netscape] combine and introduced common product to design GUI applications.

### Differences between awt and Swings:

1. AWT components are heavyweight component but swing components are light weight component.
2. AWT components consume more number of system resources Swings consume less number of system resources.
3. AWT components are platform dependent but Swings are platform independent.
4. AWT is provided less number of components where as swings provides more number of components.
5. AWT doesn't provide Tooltip Test support but swing components have provided Tooltip test support.
6. in awt for only window closing :      windowListener  
   windowAdaptor  
In case of swing use small piece of code.
  - i. f.setDefaultCloseOperation(JFrame.EXIT-ON-CLOSE);
7. AWT will not follow MVC but swing follows MVC Model View Controller It is a design pattern to provide clear separation b/w controller part,model part,view part.
  - a. Controller is a normal java class it will provide controlling.
  - b. View part provides presentation
  - c. Model part provides required logic.
8. In case of AWT we will add the GUI components in the Frame directly but Swing we will add all GUI components to panes to accommodate GUI components.

**Classes of swing:-**

\*\*\*\*\*SWING\*\*\*\*\*

```
import java.awt.*;
import javax.swing.*;

class MyFrame extends JFrame
{
    JLabel l1,l2,l3,l4,l5,l6,l7;
    JTextField tf;
    JPasswordField pf;
    JCheckBox cb1,cb2,cb3;
    JRadioButton rb1,rb2;
    JList l;
    JComboBox cb;
    JTextArea ta;
    JButton b;
    Container c;

    MyFrame()
    {
        this.setVisible(true);
        this.setSize(150,500);
        this.setTitle("SWING GUI COMPONENTS EXAMPLE");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c=this.getContentPane();
        c.setLayout(new FlowLayout());
        c.setBackground(Color.green);

        l1=new JLabel("User Name");
        l2= new JLabel("password");
        l3= new JLabel("Qualification");
        l4= new JLabel("User Gender");
        l5= new JLabel("Technologies");
        l6= new JLabel("UserAddress");
        l7= new JLabel("comments");

        tf=new JTextField(15);
        tf.setToolTipText("TextField");
        pf=new JPasswordField(15);
        pf.setToolTipText("PasswordField");

        cb1=new JCheckBox("BSC",false);
        cb2=new JCheckBox("MCA",false);
        cb3=new JCheckBox("PHD",false);
        rb1=new JRadioButton("Male",false);
        rb2=new JRadioButton("Female",false);
    }
}
```

```
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);

        String[] listitems={"cpp","c","java"};
        l=new JList(listitems);

        String[] cbitems={"hyd","pune","bangalore"};
        cb=new JComboBox(cbitems);

        ta=new JTextArea(5,20);

        b=new JButton("submit");

        c.add(l1);
        c.add(tf);
        c.add(l2);
        c.add(pf);
        c.add(l3);
        c.add(cb1);
        c.add(cb2);
        c.add(cb3);
        c.add(l4);
        c.add(rb1);
        c.add(rb2);
        c.add(l5);
        c.add(l);
        c.add(l6);
        c.add(cb);
        c.add(l7);
        c.add(ta);
        c.add(b);
    }
}
class SwingDemo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

---

\*\*\*\*\*JCOLORCHOOSER\*\*\*\*\*

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
class MyFrame extends JFrame implements ChangeListener
{
    JColorChooser cc;
    Container c;
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setTitle("SWING GUI COMPONENTS EXAMPLE");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c=getContentPane();
        cc=new JColorChooser();
        cc.getSelectionModel().addChangeListener(this);
        c.add(cc);
    }

    public void stateChanged(ChangeEvent c)
    {
        Color color=cc.getColor();
        JFrame f=new JFrame();
        f.setSize(400,400);
        f.setVisible(true);
        f.getContentPane().setBackground(color);
    }
}
class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

\*\*\*\*\*JFILECHOOSER\*\*\*\*\*

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
class MyFrame extends JFrame implements ActionListener
{
    JFileChooser fc;
    Container c;
    JLabel l;
    JTextField tf;
    JButton b;

    MyFrame()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setTitle("SWING GUI COMPONENTS EXAMPLE");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c=getContentPane();

        l=new JLabel("Select File:");
        tf=new JTextField(25);
        b=new JButton("BROWSE");
        this.setLayout(new FlowLayout());
        b.addActionListener(this);
        c.add(l);
        c.add(tf);
        c.add(b);
    }
    public void actionPerformed(ActionEvent ae)
    {
        class FileChooserDemo extends JFrame implements ActionListener
        {
            FileChooserDemo()
            {
                Container c=getContentPane();
                this.setVisible(true);
                this.setSize(500,500);

                fc=new JFileChooser();
                fc.addActionListener(this);
                fc.setLayout(new FlowLayout());
                c.add(fc);
            }
        }
    }
}
```



```

        public void actionPerformed(ActionEvent ae)
        {
            File f=fc.getSelectedFile();
            String path=f.getAbsolutePath();
            tf.setText(path);
            this.setVisible(false);
        }
    }
    new FileChooserDemo();
}

class Demo
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
-----

*****JTABLE*****
import javax.swing.*;
import java.awt.*;
import javax.swing.table.*;
class Demo1
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame();
        f.setVisible(true);
        f.setSize(300,300);
        Container c=f.getContentPane();

        String[] header={"ENO","ENAME","ESAL"};
        Object[][]
        body={{{"111","aaa",5000},{"222","bbb",6000},{"333","ccc",7000},{"444","ddd",8000}};

        JTable t=new JTable(body,header);
        JTableHeader th=t.getTableHeader();
        c.setLayout(new BorderLayout());
        c.add("North",th);
        c.add("Center",t);
    }
}
-----

```

\*\*\*\*\*APPLET\*\*\*\*\*

```
import java.awt.*;
import java.applet.*;
public class Demo2 extends Applet
{
    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,20);
        g.setFont(f);

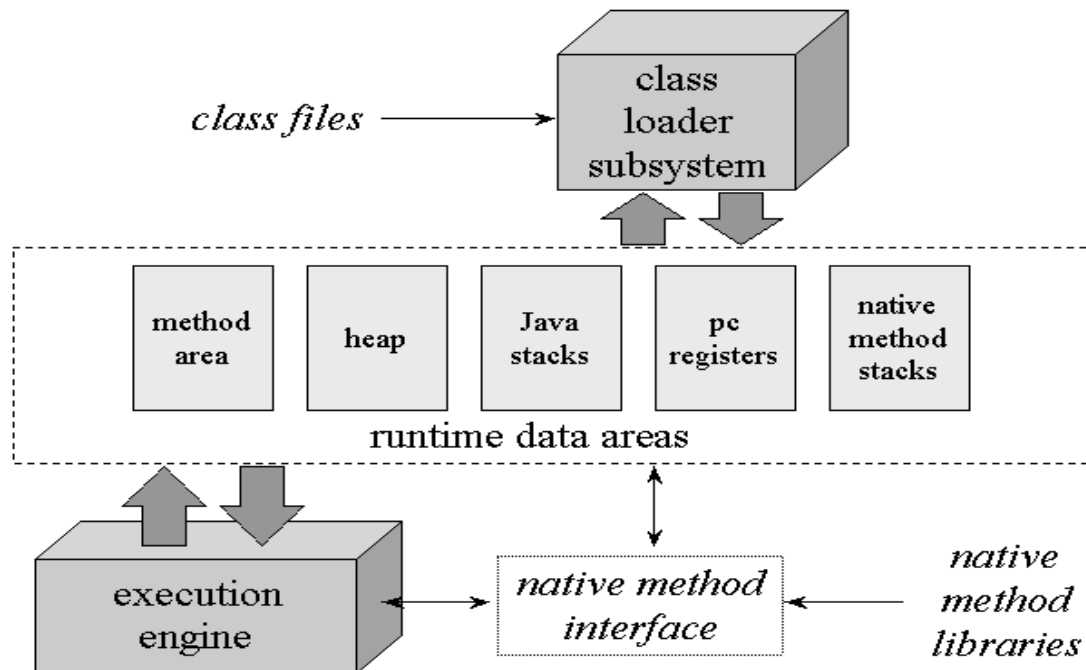
        g.drawString("Durga Software Solutions",100,200);
    }
};
<html>
<applet code="Demo2.class" width="500" height="500">
</applet>
```

</html>

-----  
\*\*\*\*\*INIT() START() STOP() DESTROY()\*\*\*\*\*

```
import java.awt.*;
import java.applet.*;
public class Demo3 extends Applet
{
    String msg="";
    public void paint(Graphics g)
    {
        Font f=new Font("arial",Font.BOLD,20);
        g.setFont(f);
        g.drawString("Durga Software Solutions "+msg,100,200);
    }
    public void init()
    {
        msg=msg+"initialization"+" ";
    }
    public void start()
    {
        msg=msg+"starting"+" ";
    }
    public void stop()
    {
        msg=msg+"stoping";
    }
    public void destroyed()
    {
        msg=msg+"destroyed";
    }
};
```

```
<html>  
<applet code="Demo3.class" width="500" height="500">  
</applet>  
</html>
```

**Internal architecture of the Java virtual machine(JVM):-****Jvm:-**

1. jvm is used to execute byte code instructions(.class files).
2. Jvm is platform dependent means for different operating systems different Jvm's are available

**Class loader subsystem:-**

1. It is used to load the classes and interfaces.
2. It verifies the byte code instructions.
3. It allots the memory required for the program.

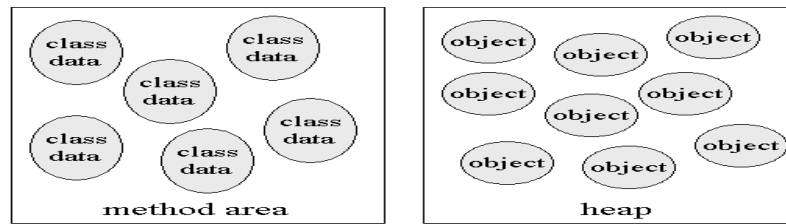
**Runtime data area:-** this is the memory resource used by the JVM and it is 5 types

**Method Area:-**

It is used to store the class data and method data.

**Heap area:-**

It is used to store the Objects.



**Runtime data areas shared among all threads.**

#### Java stacks:-

Whenever new thread is created for each and every new thread the JVM will creates PC(program counter) register and stack.

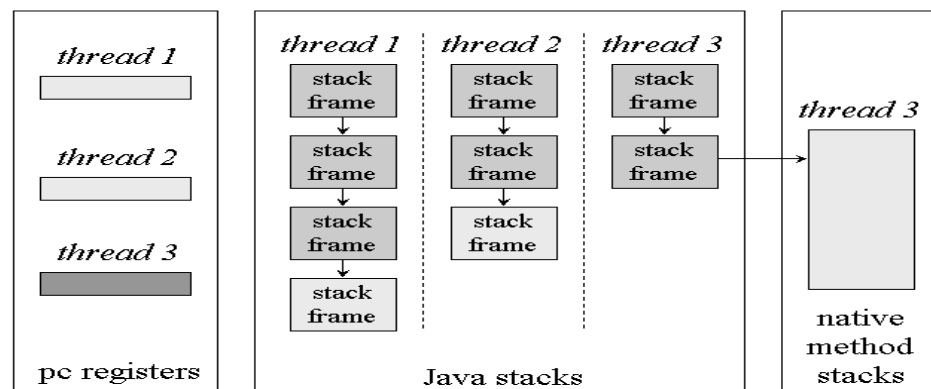
If a thread executing java method the value of pc register indicates the next instruction to execute.

Stack will stores method invocations of every thread. The java method invocation includes local variables and return values and intermediate calculations.

The each and every method entry will be stored in stack. And the stack contains group of entries and each and every entry stored in one **stack frame** hence stack is group of stack frames.

Whenever the method completes the entry is automatically deleted from the stack so whatever the functionalities declared in method it is applicable only for respective methods.

Java **native method** stack is used to store the native methods invocations.



**Runtime data areas exclusive to each thread.**

one and two are executing Java methods. Thread three is executing a native method  
Stack frames for currently executing methods are shown in a lighter shade

**Native method interface:-**

Native method interface is a program that connects native methods libraries (C header files) with JVM for executing native methods.

**Native method library:**

It contains native libraries information.

**Execution engine:-**

It is used to execute the instructions are available in the methods of loaded classes.

It contains JIT(just in time compiler) and interpreter used to convert byte code instructions into machine understandable code.

<b>modifier</b>	<b>classes</b>	<b>methods</b>	<b>variables</b>
public	yes	yes	yes
private	no	yes	yes
default	yes	yes	yes
protected	no	yes	yes
final	yes	yes	yes
abstract	yes	yes	no
strictfp	yes	yes	no
transient	no	no	yes
native	no	yes	no
static	no	yes	yes
synchronized	no	yes	no
volatile	no	no	yes