

CORE JAVA

Programming Language

Program: -

- Program is the nothing but it is the plan that contains the seat of instructions for fulfilling any specific job.
- Programming language are the languages that are used for communication between the programmer and computer system in the written way.
- Programming language is the language use to write the program.

There are two types of programming language:

1. Low Level Programming Language
2. High Level Programming Language

1. **Low Level Programming Language:** Low Level Programming Language can directly interact with the hardware.
2. **High Level Programming Language:** High Level Programming Languages never interact with hardware. Directly rather interacts with the low level language.

Low Level Language: ----- System Software

High Level Language: ----- Application Software

Application Software can be of two types:

1. Platform dependent
2. Platform independent

1. Platform Dependent: -

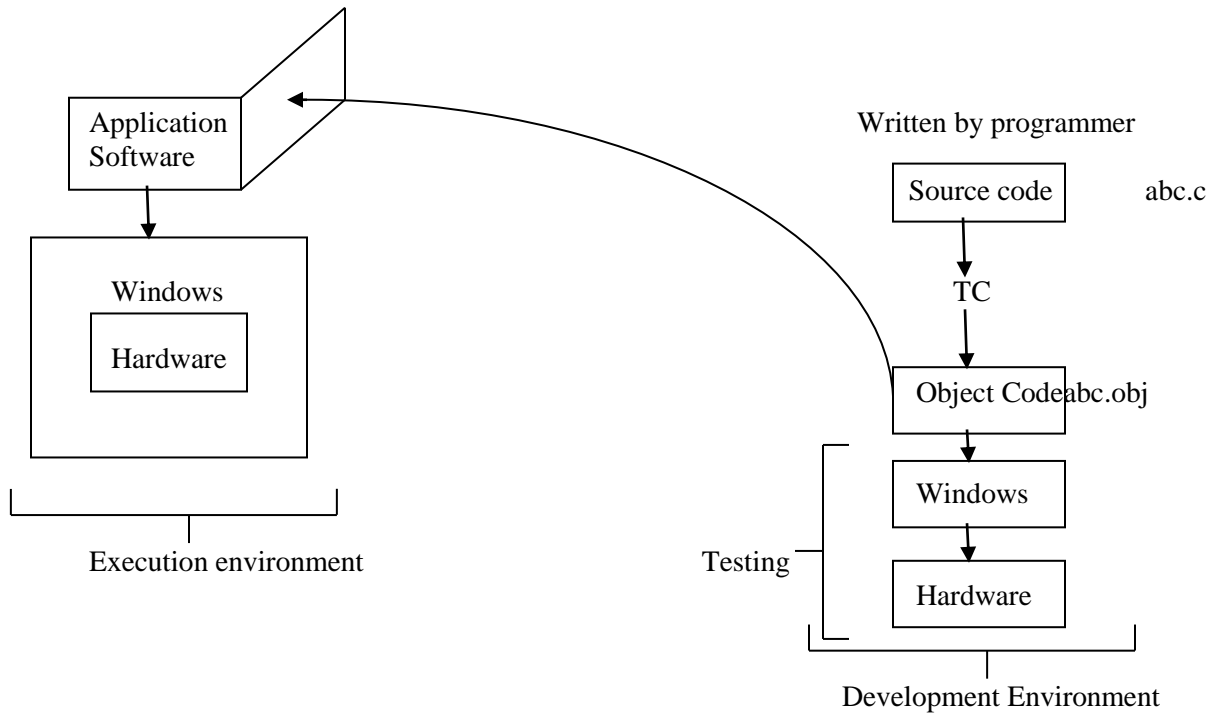
- If any Application Software is dependent then it can only be onto the Platform for which it was developed.
- If any software is developed for the windows operating system then can't be run onto any other operating system.

Question: - How any software becomes the platform dependent?

- The compiler of the technology is either platform dependent or independent.

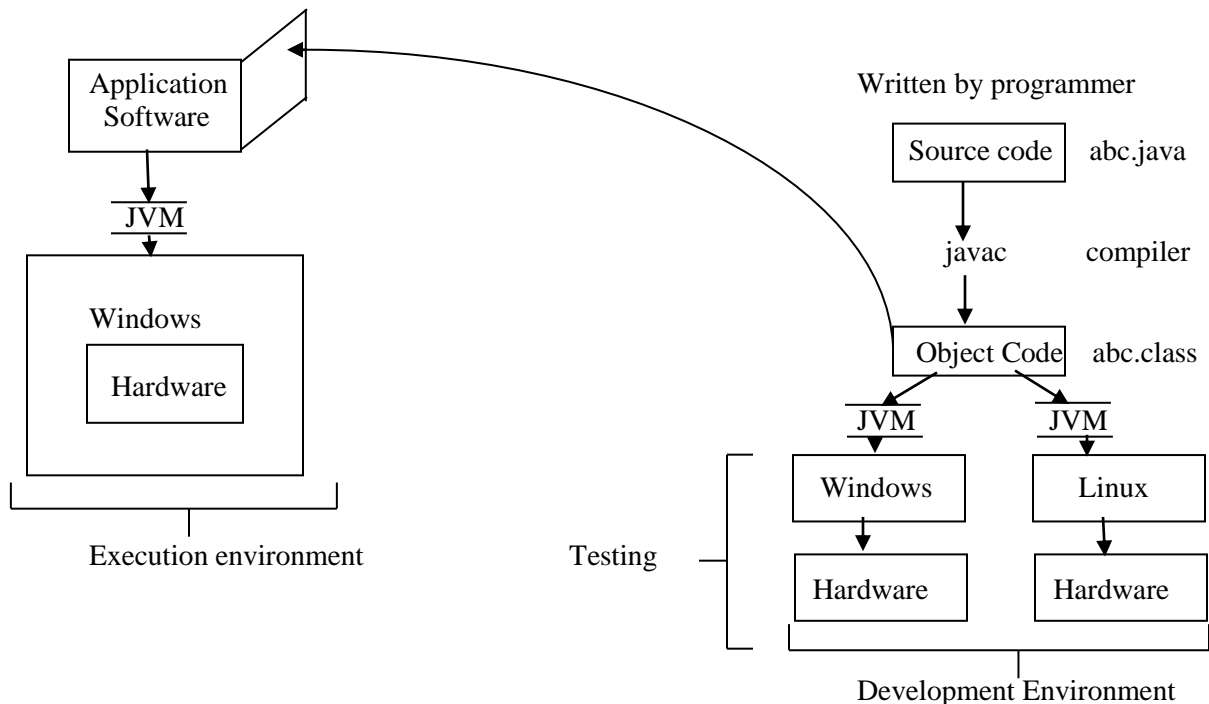
The Role of Compiler

- In any technology is to prepare the codes for the execution corresponding to the platform. The compiler of C language always prepares the code for the any specific Operating System.
- Turbo C always prepares the code for the windows Operating System.



2. Platform Independent: -

- Platform Independent application software is the software that can be run on the any platform. When the compiler on the technology does not generate the code specifically for any platform then the code is consider as the platform independent.
- In case of java the compiler never reads the code for any Operating System the java compilers create the codes for JVM.
- Java compiler generates the **.class** file that contains the write byte code. Byte code is not native to any Operating System rather it is native to JVM.



- There are the different JVM for the different operating system. JVM can be installed or uninstalled separately.

JDK (JAVA DEVELOPMENT KIT):

- JDK is the setup of java that contains a development tools and JRE (Java Runtime Environment). There are the different versions of JDK.
- First version was JDK 1.0 launched in the November 1995 and the latest version is JDK 8 in the March 2014.
- JDK is used by the developers to develop the application in java.

JDK = TOOL + JRE

JRE (JAVA RUNTIME ENVIRONMENT):

- JRE is also the setup separated from the JDK that means JRE can also be installed and uninstalled separately.
- The software applications developed in java can never be executed without the JRE. JRE contains the libraries and JVM.

JRE = LIBRARIES + JVM

VERSION OF JDK

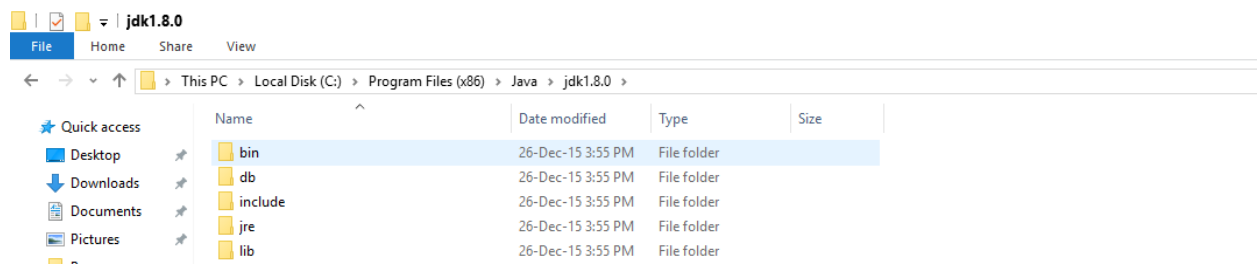
Version	Year	Packages	Classes	New features
JDK 1.0	Nov-1995	8 packages	212 classes	
JDK 1.1	Feb-1997	23 packages	504 classes	Inner Class, Windows Programming, JDBC, RMI
JDK 1.2	Starting of 1999	59 packages	1520 classes	Collection API, Struts, Java Foundation Classes, Keywords etc.
J2SE 1.3	May-2000	76 packages	1842 classes	Doesn't contain any popular new feature.
J2SE 1.4	Feb-2002	135 packages	Approximately 3000 classes	Assert Keyword, Exception Changing, XML support.
J2SE 5	Sep-2004		Approximately 3500 classes	Var-args, for-each loop, static, annotation, generic, import, etc.
J2SE 6 (JSE 6)	Dec-2006	Approximately 200 packages	Approximately 4000 classes	Java Compiler API, PRI main method.
J2SE 7 (JSE 7)	July-2011		Approximately 4500 classes	String in switch case, Automatic Null Handling, Multiple Exception Handling, Binary Literals, etc.
J2SE 8 (JSE 8)	March-2014		Approximately 5000 classes	Private class, Static Method in the Interface, etc.

Different between public JRE AND private JRE

- Private JRE is always being available in the JDK and Public JRE is available outside the JDK.
- By using private JRE we can run any **.class** file directly but executable file can't be run in the private JRE.

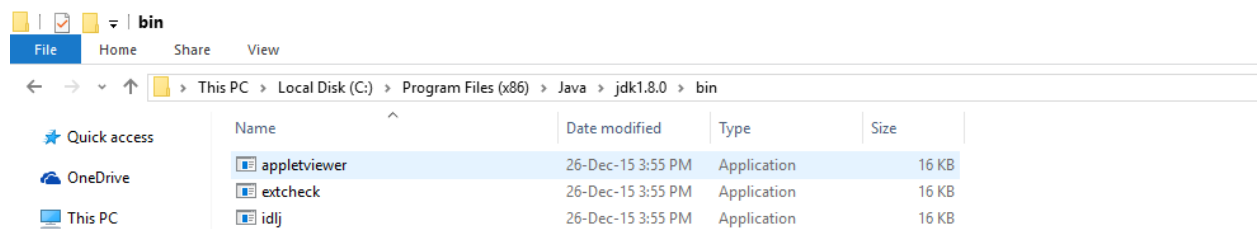
After the installation of JDK the home folder will be C drive program file.

C:\Program Files\Java\jdk 1.8.0



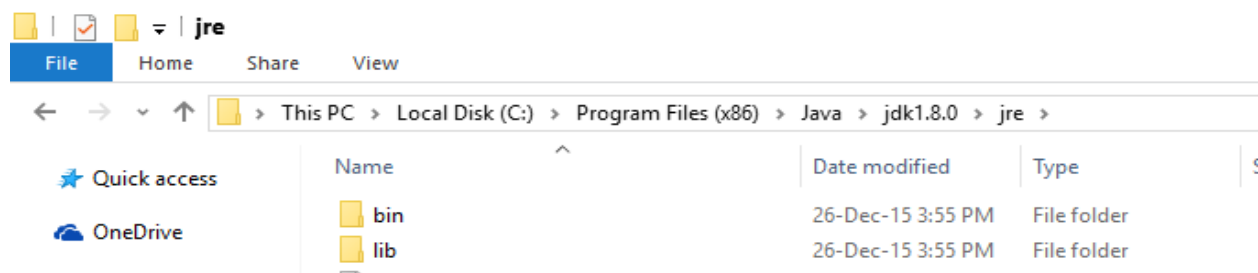
➤ Tools under the JDK

C:\Program Files\Java\jdk 1.8.0\bin



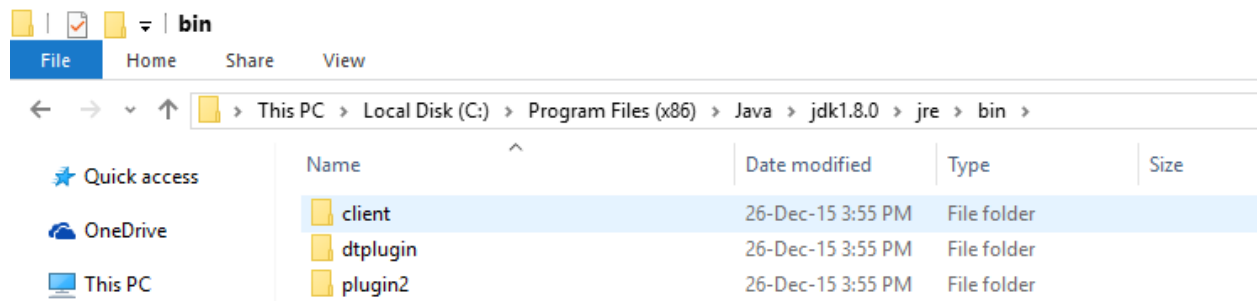
➤ Private JRE under the JDK

C:\Program Files\Java\jdk 1.8.0\jre



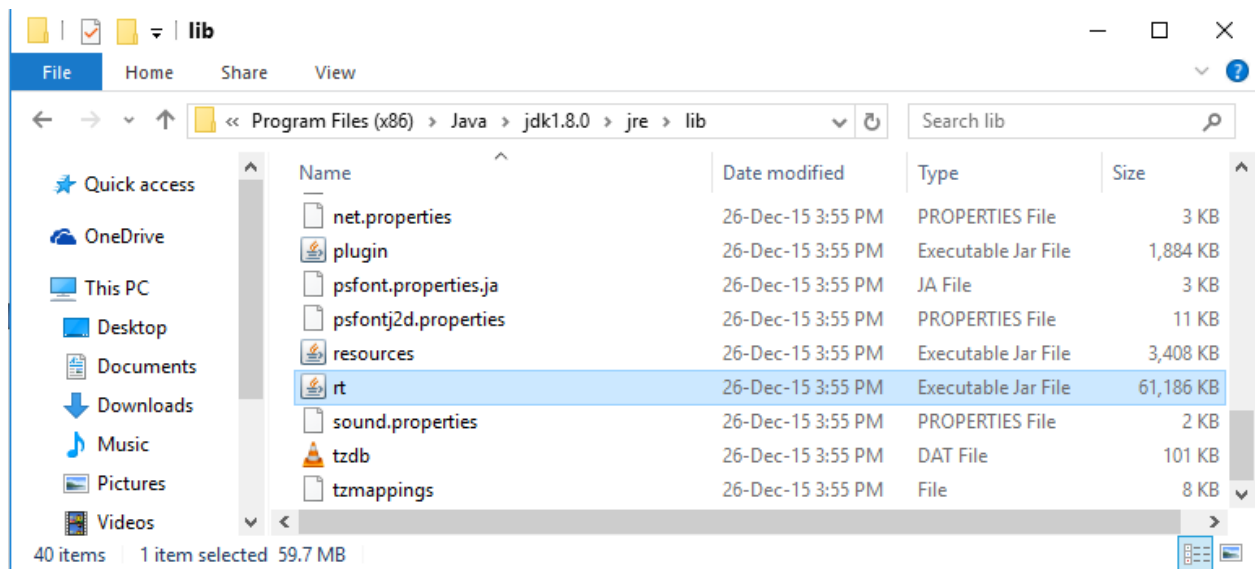
➤ JVM in the JRE

C:\Program Files\Java\jdk 1.8.0\jre\bin



➤ Library in the JRE

C:\Program Files\Java\jdk 1.8.0\jre\lib\rt.jar



- In JDK is 32 bit version then will be installed in the program files(x86).
- In JDK is 64 bit version then will be installed in the program files.
- ❖ In the JRE\bin there is exe named **java.exe** that it use to start the JVM.
- ❖ JVM is not the single files but it is the collective from of the multiple **.dll** files and **.exe** files exist in the JRE\bin.
- ❖ JAR is the tool in the JDK which is used to compress the contents of the folder and makes the **.jar** file.
- ❖ **rt.jar** is the compressed file that contains the java library class.

Development Process:-

- Development always perform three task in their development process
 - Writing the source code
 - Compilation of the source code
 - Testing (execution) of the compiled code.

[**Note:** - in the JDK doesn't provide any IDE (integrated development environment) we have to install them separately. IDE's of java Ex: - netbeans, Eclipse, Myeclipse.]

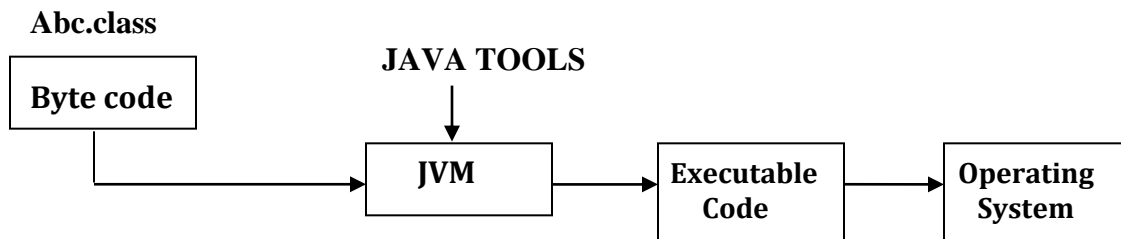
Now we can write the source code in the notepad or any other text editor and we can use the development tools in the MSDOS.

Development Tool in the JDK

1. **JAVA TOOLS:** - This tool is the java compiler that is used to compile the source code and generate the byte code.



2. **JAVA TOOL:** - This is use to start the JVM to run the byte code.



- JVM started
- .class file loaded into the JVM
- Executable code prepared
- Executable code to provide to the operating system by the JVM for the execution.

3. **JAVAP:** - This tool is use to list out the details of any class.
4. **JAVADOC:** - This tool is use to create the technical documentation of the java classes.
5. **JARTOOL:** - This tool is use to make the .jar file.

Setting the Path of the Tools

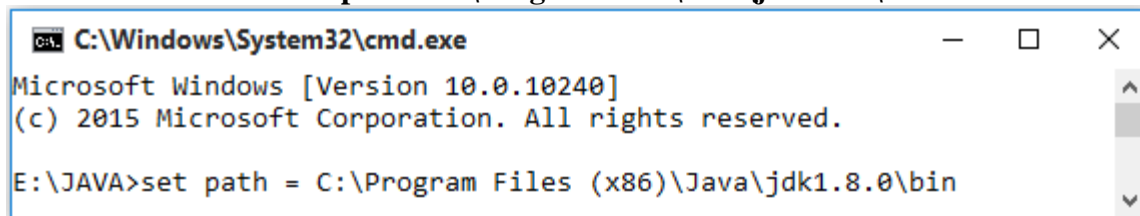
In order to use the java tools in the MSDOS and in order to use the java tools by the other software such as web server, IDE's etc.

They are two ways to set the path

- Temporary path setting
- Permanent path setting

1. **Temporary path setting:** -

Set path = C:\Program Files\Java\jdk 1.8.0\bin

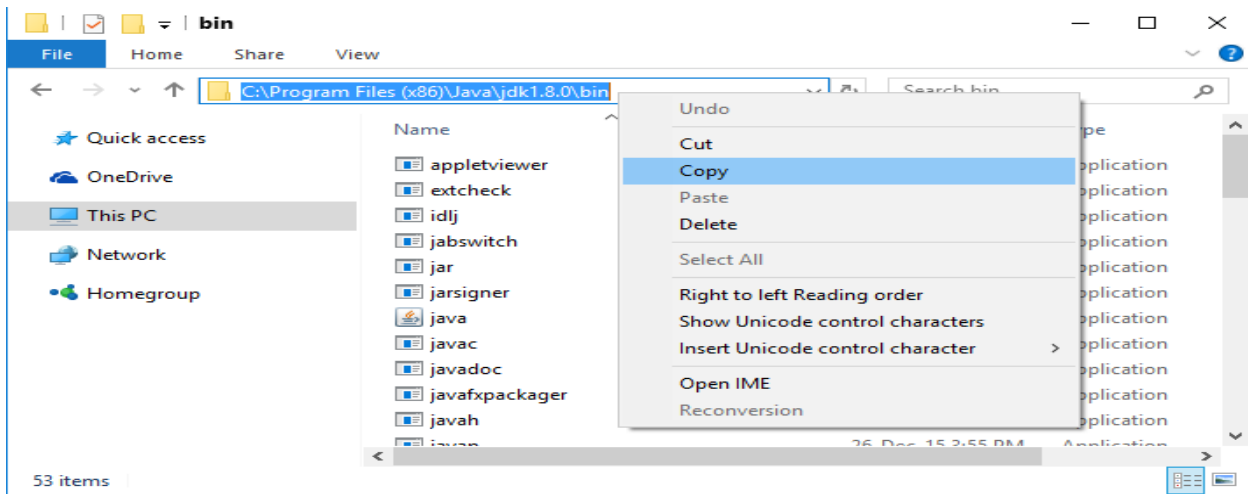


(Note: - Temporary path work only for the current session of the DOS)

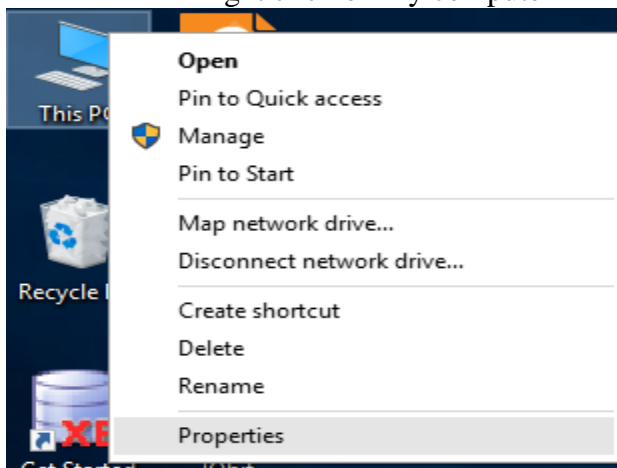
To view the entire path those are already set in MSDOS prompt

Set path

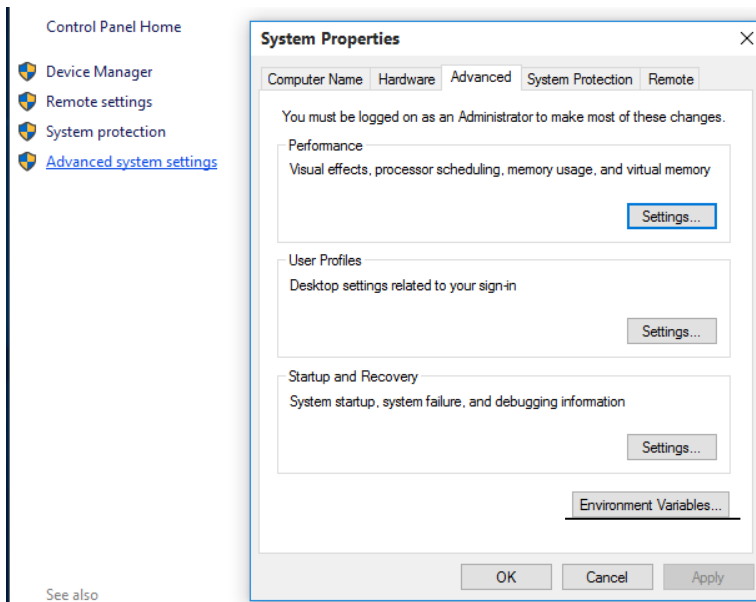
2. Permanent path setting: -
Copy the java\bin path.



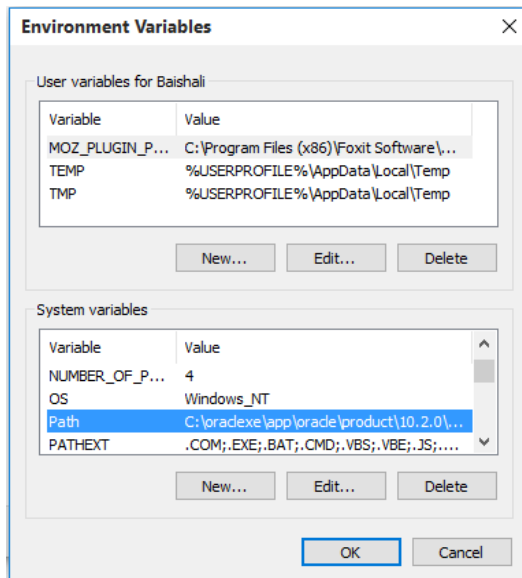
➤ Right click on my computer -----> Select properties



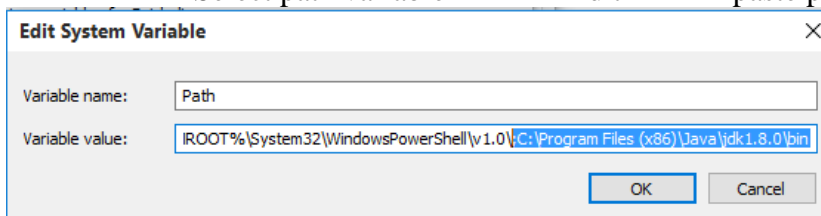
➤ Advance System Setting ----->



➤ Select Environment variables.



➤ Select path variable -----> Edit -----> paste path ----> Ok.



First java programTest.java

class Demo

{

public static void main(String []args)


```

    {
        System.out.println("Hello, Welcome in the first Java Program");
    }
}

```

Output

The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user is in the directory "E:\JAVA\Programs\Core Java Program\First Program". They run the command "javac Demo.java" to compile the program. Then, they run "java Demo" to execute it. The output of the program is "Hello, Welcome in the first Java Program".

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program>javac Demo.java
E:\JAVA\Programs\Core Java Program\First Program>java Demo
Hello, Welcome in the first Java Program
E:\JAVA\Programs\Core Java Program\First Program>

```

[**Note:** - Byte code- it contains 1byte, basically its shorthand style code]

All the programming in java always is performed within the class. Except the declaration, package import and the class level annotation.

Syntax of valid .java file

```

package p1;           → package Declaration
import my package.*;  → package importing
@TestAnnotation       → use the annotation
class Test
{
    Variable declaration
    Method declaration;
}

```

Components of the our 1st program **Demo.java**

class Demo

```

class    → keyword
Test     → identifier (user defined name)

```

public static void main(String []args)

```

public    → Access specifier           → keyword
static    → Access modifier           → keyword
void      → Return type of main method → keyword
main      → Method name
String    → Library class of java
[]args    → Identifier(name of Array)

```

System.out.println();

```

System    → Library class
out       → Object of PrintStream class
println   → Method of PrintStream

```

Println: - after println the statement println place the cursor on the new line.

System.out.print();

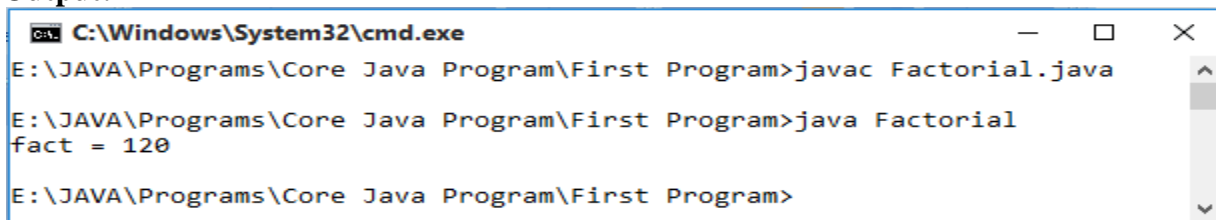
Print: - It wills the cursor in same line after printing the same line.

// To find the Factorial no

```
class Factorial
{
    public static void main(String []s)
    {
        int a, f=1;
        int temp=5;
        for(a=1;a<=temp;a++)
        {
            f=f*a;
        }
        System.out.println("fact = "+f);
    }
}
```

└─ Concatenation operator

Output: -



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program>javac Factorial.java
E:\JAVA\Programs\Core Java Program\First Program>java Factorial
fact = 120
E:\JAVA\Programs\Core Java Program\First Program>
```

// To find the no is prime

```
class Prime
{
    public static void main(String []s)
    {
        int a=2,p=0;
        int temp=118;
        while(a<=temp-1)
        {
            if(temp%a==0)
            {
                System.out.println("not Prime");
                break;
            }
            a++;
        }
        if(a==temp)
            System.out.println("Prime");
    }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program>javac Prime.java
E:\JAVA\Programs\Core Java Program\First Program>java Prime
not Prime
E:\JAVA\Programs\Core Java Program\First Program>
```

// To find a power b no.

```
class Power
{
    public static void main(String []s)
    {
        int a=2, b=4, pow=1, i;
        for(i=1; i<=b; i++)
        {
            pow= pow * i ;
        }
        System.out.print("Power= "+pow);
    }
}
```

// To print this Series 0 1 1 2 3 5 8 13.....n.

```
class Series
{
    public static void main(String []s)
    {
        int n = 60;
        int a = 0, b = 1, i;
        for(i = 1; i <= n; i++)
        {
            i = a + b;
            System.out.print(a + " ");
            a = b;
            b = i;
        }
    }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program>javac Series.java
E:\JAVA\Programs\Core Java Program\First Program>java Series
0 1 1 2 3 5 8 13 21 34
E:\JAVA\Programs\Core Java Program\First Program>
```

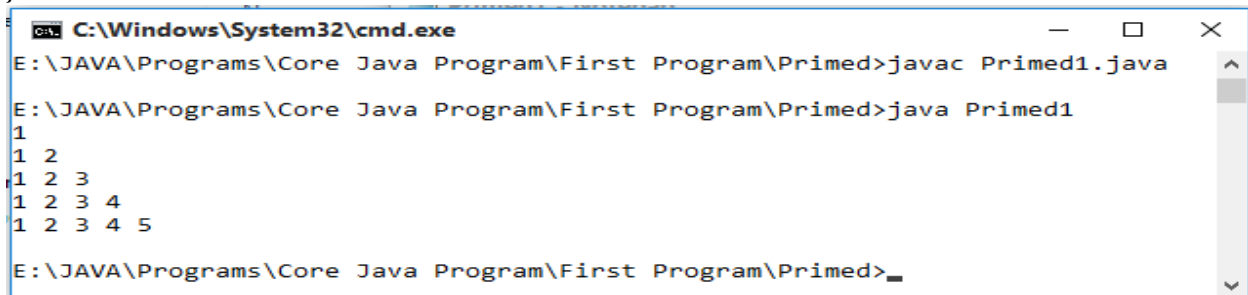
// To print primed.

```
class Primed1
{
```

```

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

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program\Primed>javac Primed1.java
E:\JAVA\Programs\Core Java Program\First Program\Primed>java Primed1
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
E:\JAVA\Programs\Core Java Program\First Program\Primed>

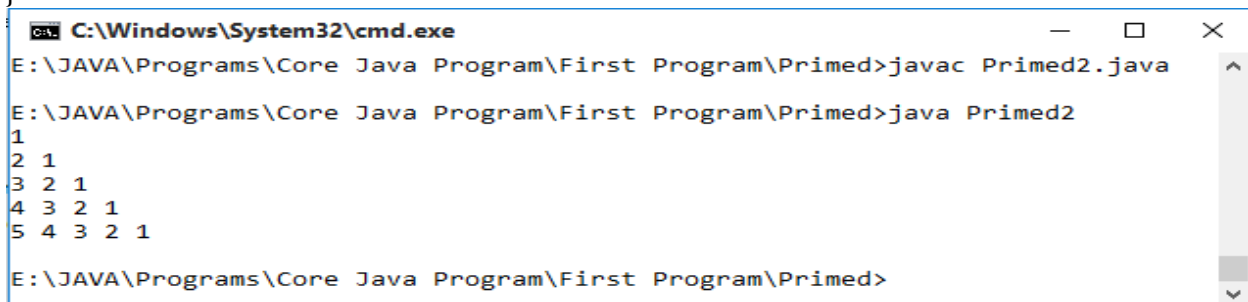
```

// To print primed.

```

class Primed2
{
    public static void main(String []s)
    {
        int i, j;
        for(i=1; i<=5; i++)
        {
            for(j=i; j>=1; j--)
            {
                System.out.print(j+" ");
            }
            System.out.println();
        }
    }
}

```



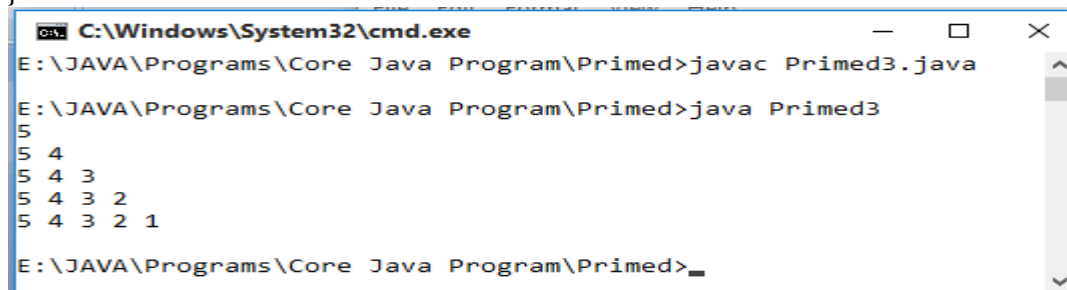
```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\First Program\Primed>javac Primed2.java
E:\JAVA\Programs\Core Java Program\First Program\Primed>java Primed2
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
E:\JAVA\Programs\Core Java Program\First Program\Primed>

```

// To print primed.

```
class Primed3
{
    public static void main(String []s)
    {
        int i, j;
        for(i =5; i >=1; i--)
        {
            for(j=5; j>=i; j--)
            {
                System.out.print(j+" ");
            }
            System.out.println();
        }
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user has navigated to the directory "E:\JAVA\Programs\Core Java Program\Primed" and executed the following commands:

```
E:\JAVA\Programs\Core Java Program\Primed>javac Primed3.java
E:\JAVA\Programs\Core Java Program\Primed>java Primed3
```

The output of the program is displayed as follows:

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

The prompt then returns to "E:\JAVA\Programs\Core Java Program\Primed>".

// To print primed.

```
class Primed4
{
    public static void main(String []s)
    {
        int i, j;
        for(i =5; i >=1; i--)
        {
            for(j=i; j<=5; j++)
            {
                System.out.print(j+" ");
            }
            System.out.println();
        }
    }
}
```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Primed>javac Primed4.java
E:\JAVA\Programs\Core Java Program\Primed>java Primed4
5
4 5
3 4 5
2 3 4 5
1 2 3 4 5
E:\JAVA\Programs\Core Java Program\Primed>_

```

// To print primed.

class Primed5

```

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

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Primed>javac Primed5.java
E:\JAVA\Programs\Core Java Program\Primed>java Primed5
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
E:\JAVA\Programs\Core Java Program\Primed>_

```

Command Line Argument

- Command line argument is the way by which we can provide the values to the program from outside the program.
- The java tool provide the syntax to pass the command line argument

Java **Test** **abc xyz** ←
 ↓ ↓ ↓
 Tool class command

Ex:

class Test

```

{
    public static void main(String []s)
    {

```

command line argument

```

        Variable;
    }
}

```

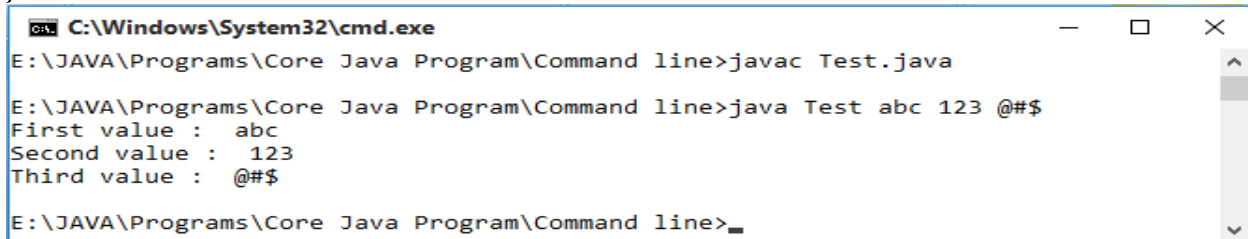
- All the command line argument is automatically stored into the array and that array is automatically passed into the argument of main method by the JVM.

Example: -A java program to get the command line values.

```

class Test
{
    public static void main(String s[])
    {
        System.out.println("First value: "+s[0]);
        System.out.println("Second value: "+s[1]);
        System.out.println("Third value: "+s[2]);
    }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Command line>javac Test.java
E:\JAVA\Programs\Core Java Program\Command line>java Test abc 123 @#$
First value : abc
Second value : 123
Third value : @#$
E:\JAVA\Programs\Core Java Program\Command line>_

```

All the command line argument separated by the space

[Note: -in the java each array has the length properties that specify the no of elements content by the array.]

Example: - A java program to get the command line value using loop.

```

class Test
{
    public static void main(String s[])
    {
        int i;
        for(i =0; i <s.length; i++)
            System.out.println("Value: "+s[i]);
    }
}

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Command line>javac Test1.java

E:\JAVA\Programs\Core Java Program\Command line>java Test1 abc 123 @$
Value : abc
Value : 123
Value : @$

E:\JAVA\Programs\Core Java Program\Command line>
```

Converting the string type numeric value into “int” type

If numeric values are contained in the string then we can't perform any arithmetic operation on that value rather we have to convert that string type numeric value into int type.

Integer.parseInt()

Class method

Integer is the java library class.

parseInt method is used to convert the string type numeric value into the int primitive type.

Example: Creating a java program to calculate the sum of values entered from the command line.
class Sum

```
{
    public static void main(String s[])
    {
        int i, sum, x;
        sum=0;
        for(i=0;i <s.length;i++)
        {
            x = Integer.parseInt(s[i]);
            sum=sum + x;
        }
        System.out.println("Value : "+sum);
    }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Command line>javac Sum.java

E:\JAVA\Programs\Core Java Program\Command line>java Sum 2 5 4 1
Value : 12

E:\JAVA\Programs\Core Java Program\Command line>
```

Example: -Except the value from command line and check how many values are even or odd?

// To find the value is even or odd

class Even

```
{
    public static void main(String s[])
    {
        int i, x;
```



```

        for(i =0;i <s.length;i++)
        {
            x = Integer.parseInt(s[i]);
            if(i%2==0)
            {
                System.out.println("given no is even: "+s[i]);
            }
            else
                System.out.println("given no is odd : "+s[i]);
        }
    }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Command line>javac Even.java

E:\JAVA\Programs\Core Java Program\Command line>java Even 5 4 9 6 3
given no is odd : 5
given no is even: 4
given no is odd : 9
given no is even: 6
given no is odd : 3

```

Example: - Except the value from command line and find out the minimum and maximum values.

//To find the value is maximum and minimum value

class MaxMin

```

{
    public static void main(String s[])
    {
        int i, x;
        int max=Integer.parseInt(s[0]);
        int min=Integer.parseInt(s[0]);
        for(i =0; i <s.length; i++)
        {
            x = Integer.parseInt(s[i]);
            if(x>max)
                max=x;
            if(x<min)
                min=x;
        }
        System.out.println("Maximum value : "+max);
        System.out.println("Minimum value : "+min);
    }
}

```

```

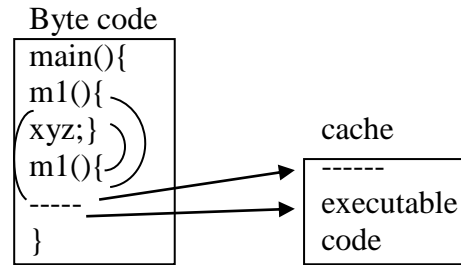
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Command line>javac Maxmin.java

E:\JAVA\Programs\Core Java Program\Command line>java Maxmin 5 4 9 6 3
Maximum value : 9
Minimum value : 3

E:\JAVA\Programs\Core Java Program\Command line>

```

JDK --[JRE --[JVM --[JIT



Two steps conversion in java.

In the java from write in the source code to the execution of the code there are two layers of the conversation (Translator).

1. Source Code to Byte Code (by the javac(compiler))
2. Byte Code to Executable Code(by the JVM)

[Note: - Executable Code can never be seen.]

The high level languages there are two types of conversions

1. Compilation
2. Interpreter

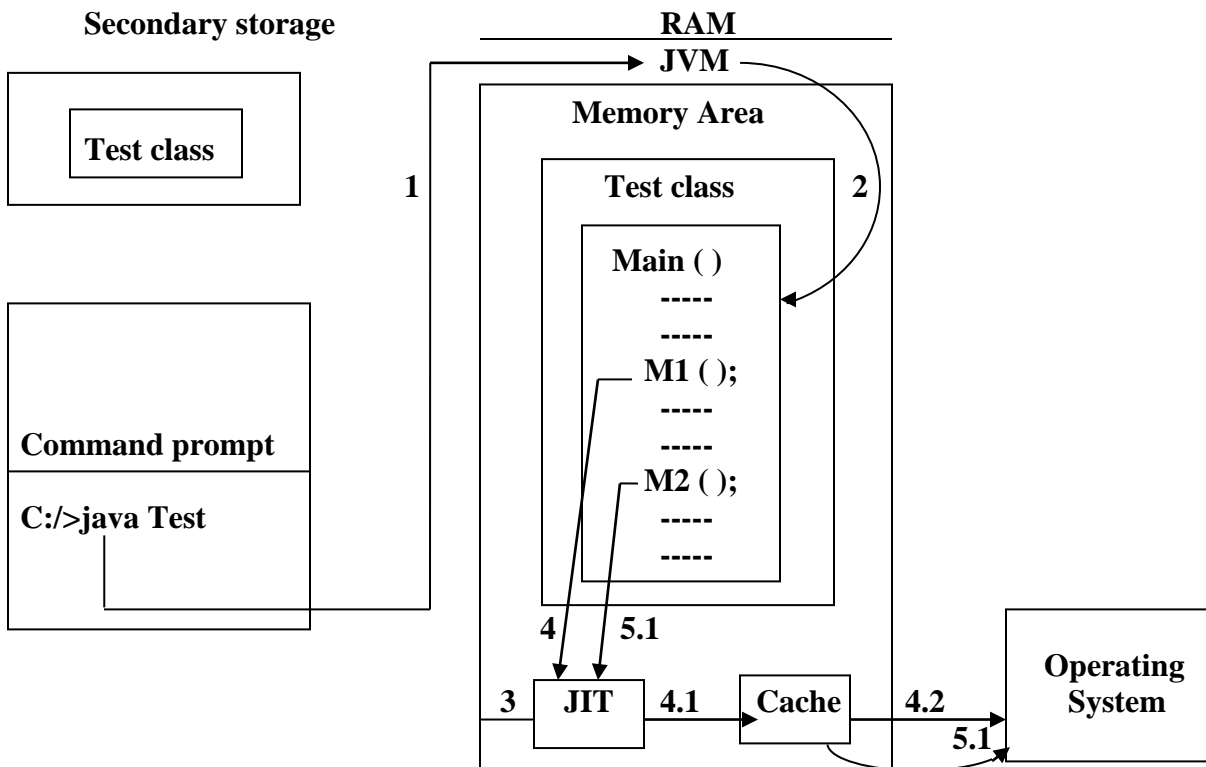
Difference between Compiler and Interpreter

Compiler	Interpreter
It performs the static conversion.	It performs a conversion directly.
It converts and store into separate file in the hard disk.	It converts the code at the execution time. (So converted code never be executed)
Execution of the program is fast if that is prepared by the compiler.	Execution of the program becomes slow if it is prepared by the interpreter.
It takes the more memory because he converts the full code.	It converts in run time as per the requirement.

- Initially JVM in java was the interpreter but now the JVM is the compiler come interpreter both.
- As the conversion tool JVM has JIT (**Just In Time Compiler**).

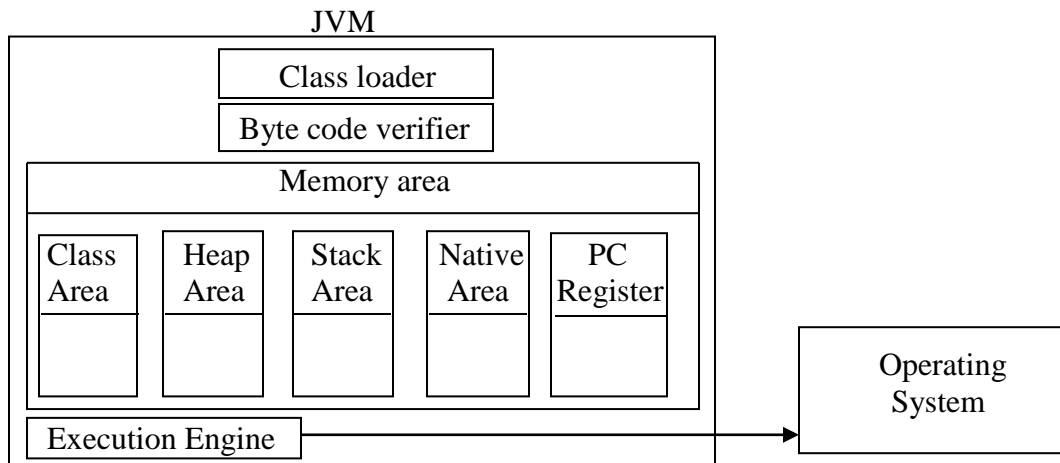
JIT (Just In Time): -

- **JIT** instance for just in time compiler which is responsible to convert the byte code into the executable code in the following manner.
- When a method calling is encountered first time, the JIT converts there instructions and provided then to the operating system for the execution and at that same time also stored the converted instruction in the temporary cache.
- When the method calling this encountered again then the JIT fix up the converted instructions from the temporary cache and provided to the operating system.



1. Java command issued to run the Test class and immediately JVM gets loaded into the RAM.
2. Test class loaded into JVM that means byte code of the total class loaded.
3. JIT started to convert the instruction of main method.
4. M1 () method calling encountered.
 - 1.1 Instruction of M1 () method converted by the JIT and stored into the cache.
 - 1.2 Instruction provide to the operating system for the execution.
- 5.1 Calling of M1 () method again.
- 5.2 JIT pickup the instruction from the cache and provide to the operating system.

THE ARCHITECTURE OF THE JVM



1. Class loader
2. Byte code verifier
3. Memory area
 - i. Class area
 - ii. Heap area
 - iii. Stack area
 - iv. Native area
 - v. Pc register
4. Execution engine

1. Class loader

Class loader is the utility in the JVM which is responsible to load the **.class** file from hard disk to JVM.

There are Three Types of class Loader

1. **Boot Strap class loader**
2. **System class loader**
3. **Extended class loader**

1. Boot Strap class loader

- Boot strap class loader loads the **.class** file from the java library available in the JRE.
C:\Program Files\java\jdk1.8.0\jre\lib\rt.jar
 - a. **.jar** file is the compressed file.
 - b. **rt.jar** file contains the approximately 5000 classes within the different packages.

[**Note:** - If we were using the class of the java library in our class then all that classes also gets loaded in the JVM.]

For Ex: - In the last program **System class**, **String class** and **Integer class** are the library classes that we were using in our Test class.

- Along with our test class there are three library classes also get loaded into the JVM.

2. System class loader

- System class loader is responsible to load the user define classes from the class path.
- The current directory that we are opened in the DOS is the default class path.

3. Extended class loader

This class loader will load the **.class** file from the **jre\lib\ext** folder.
C:\Program Files\Java\jdk1.8.0\jre\lib\ext

In the **ext folder** the **.class** file should not be placed directly rather must be placed within the **.jar** file.

Jar tool in java is used to make a **.jar** file and also used to uncompress the **.jar** file.

.jar tool is also available in the

C:\Program Files\Java\jdk1.8.0\jre\bin

Steps to make the .jar file

Step 1: - prepare the .class file by the javac these are the .class file that we want to move in the .jar file.

C:\java>javac abc.java

.class will be generated.

Step 2: -Execute the jar command

C:\java > java cvf sample.jar *.class

Prompt	Tool	Switches	Name	All the .class
		C-console	of .jar	files in
		V-verbose	files to	current directory
		F-format	be created	

Step 3: - After creating the **sample.jar** file just place it in the **jre\lib\ext** folder.

Step 4: - Delete the .class file from the current directory

Step 5: - Run the program **java Test**

That contains main method

Step 6: - System class loader becomes fail to load **Test.class** but extended class loader will load successfully from

C:\Program Files\java\jdk1.8.0\jre\bin\ext\sample.jar

Sequence of the class loader to load any class

1. Extended class loader
2. System class loader
3. Boot strap class loader

2. Byte code verifier

- Byte code verifier is the tool in the JVM that verify to the code of the .class file loader by the class loader
- In the content in the .class file is found correct by the byte code verifier than the content of the .class file copied into the memory area of the JVM.

3. Memory area

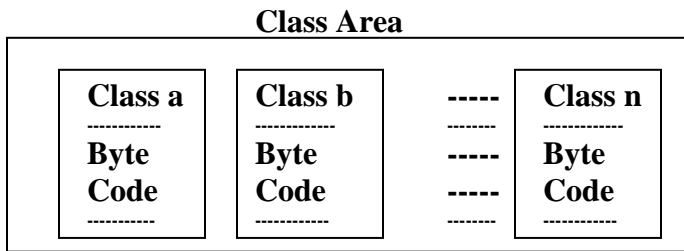
Memory area in JVM there are five types of memory areas in the JVM

1. Class area
2. Heap area
3. Stack area
4. Native area
5. Pc register

1. Class area

Class areas in the JVM are used to hold the .class file which is loaded by the class loader.

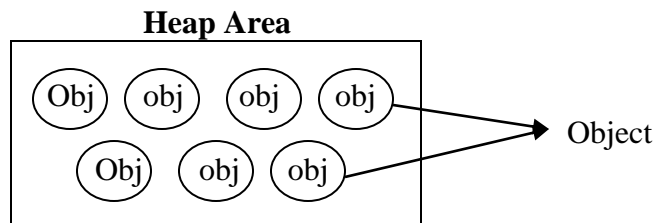
They are only one class area in the JVM which is shared by all the threads.



2. Heap Area

All the objects always are created in the heap area.

There is only one heap area as per JVM which is shared by all the thread running in the JVM.

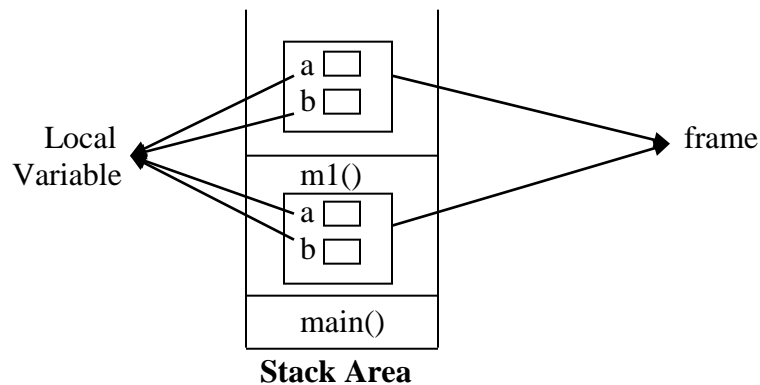


3. Stack Area

Stack area is used in the method execution.

That means when any method is invoked then immediately a new frame is created in the stack and as the method completed then the frame of that method gets destroyed.

[Note: - There is the separate stack in the JVM for the separate threads.]

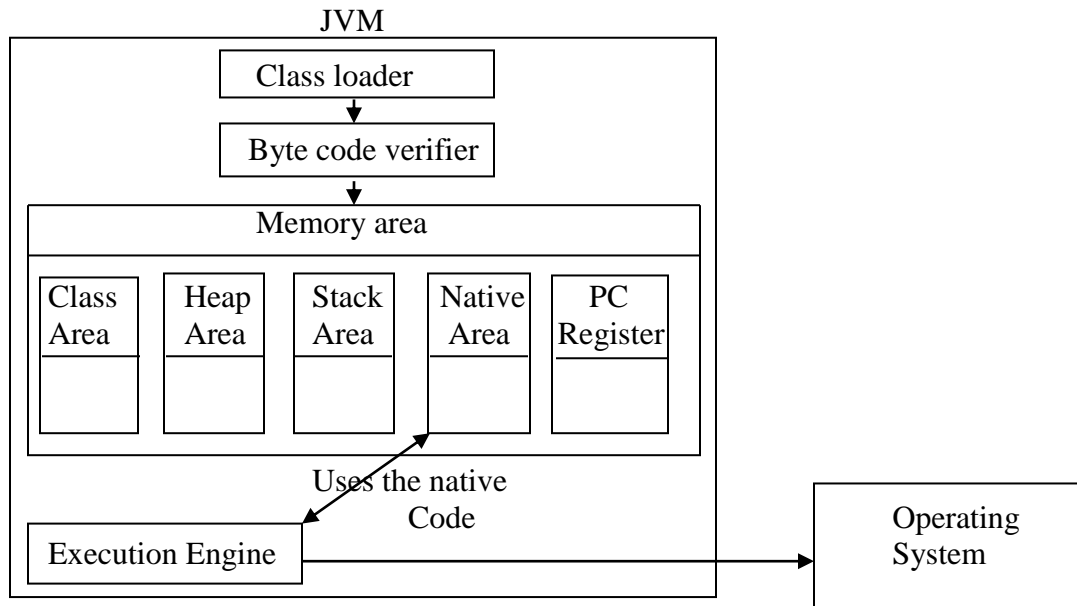


4. Native Area

- Native area in the JVM is use to contain the native code.
- A native code is the code written in java in the special manner by using the JNI (**Java native Interface**).
- Through the native code is the function of the C & C++ can be invoked.
- In the operating system the internal library is developed in the C language that is invoked by JVM through the native code.
- There is a separate native code for the separate thread.

5. PC Register

- A pc register contains the next instruction to be executed.
- There are separate pc register for each thread.



Flow of java within the JVM.

class Factorial

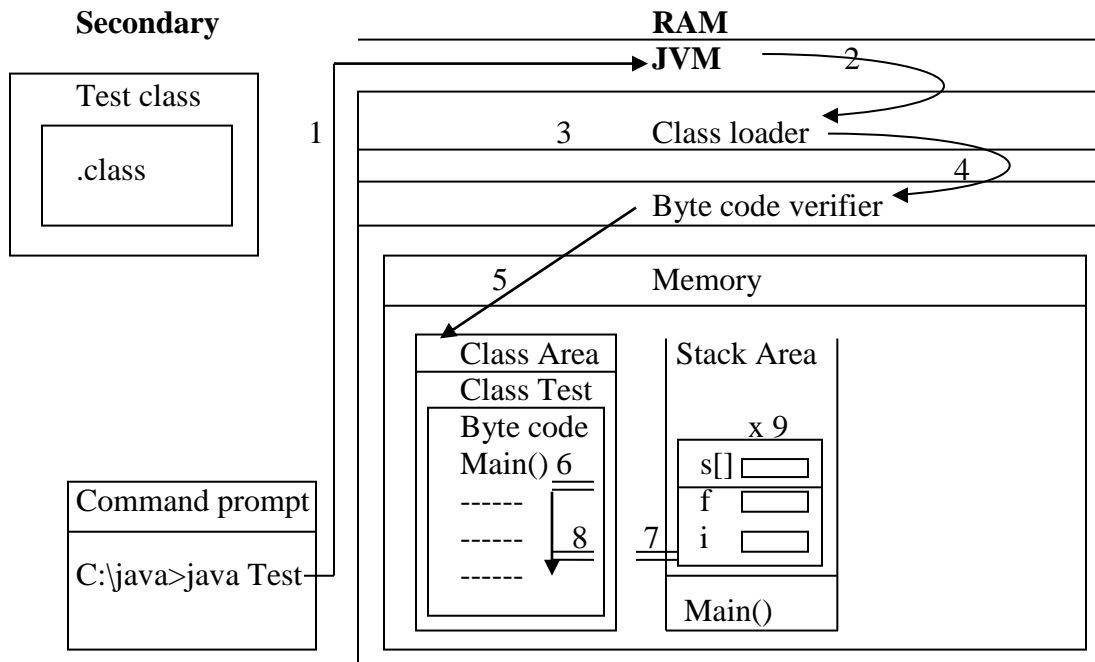
```
{
    public static void main (String s[ ])
    {
        int x=5, f=1, i;
        For (i=1; i<=x; i++)
        {
            f = f * i;
        }
        System.out.println("Factorial = " + f);
    }
}
```

C:\java> java Factorial

└─> 1. Start the JVM

└─> 2 Test class loaded & executed

Complete flow of the Test class in the JVM

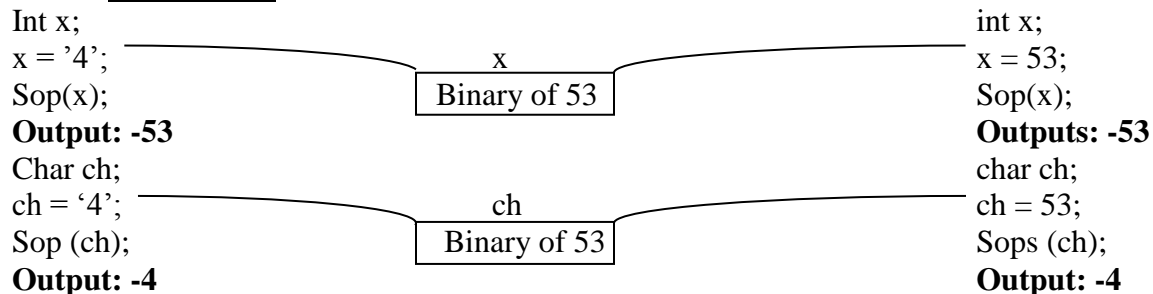


1. JVM loaded in the memory.
2. Class loader invoked by the JVM.
3. **Test.class** file loaded by the class loader.
4. **.class** file provided to the byte code verifier.
5. After the successful verification class loaded into the class area.
6. **main()** method started.
7. Frame of the **main()** method created.
8. Execution of the instruction of the **main()** method.
9. Execution of **main()** method completed and frame gets destroyed.

BASIC OF JAVA

1. Data type
2. Constants
3. Variables
4. Keywords
5. Operators

1. Data Type




```
Char ch1, ch2;
ch1 = '4';
ch2 = '5';
int sum;
sum = ch1 + ch2;
Sop (sum);
Output: -105
```

```
char ch1, ch2;
ch1 = '4';
ch2 = '5';
int sum, x, y;
x = ch1 - 48;
y = ch2 - 48;
sum = x + y;
Sop (sum);
Output: - 9
```

Data type is the type of data it can be numeric value either decimal types or nonnumeric type or it can be the alphabetical, symbols and date type.

All the value is internally stored in the binary format that means in the format 1 and 0.

In the high level programming language there are number of data type to store the values as the real world value.

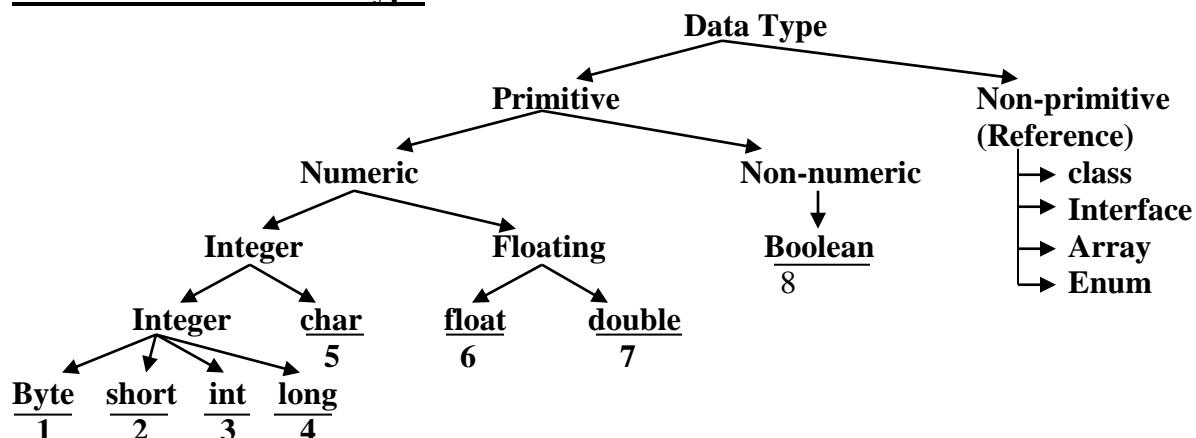
Data Types in Java

5		2	1	= 8 Types of data type
Byte, short, int Long, char	float double	Boolean		
Non floating Point (Integer)	Floating Point			
Numeric		Non-numeric		
1 0 1 0 1 0.....				

Difference between Numeric and Non-numeric

- Numeric type have to be converted by defining dividing for 2 parts non floating and floating.
- Non-numeric type are already stored in the binary format so no needed converted into the binary.

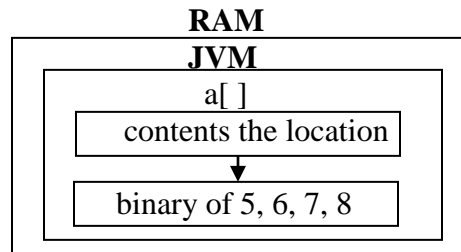
Detailed chart of Data Type



- Reference type's data type the location where the actual data is stored.

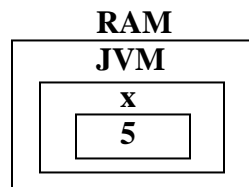
It represents the address where value is stored.

Ex: - `int a[] = {5, 6, 7, 8};`



- The primitive types are the type that represent to the actual data. It represents the actual value.

Ex: -`int x = 5;`



Configuration list of the Primitive Type

Data Type	Size	Range
Byte	1 byte	-128 to +127 (-2^7 to $+2^7 - 1$)
Short	2 bytes	-32768 to +32767 (-2^{15} to $+2^{15} - 1$)
Int	4 bytes	-2^{31} to $+2^{31} - 1$
Long	8 bytes	-2^{63} to $+2^{63} - 1$
Char	2 bytes	0 to 65535 (0 to $+2^{16} - 1$)
Float	4 bytes	-2^{31} to $+2^{31} - 1$
Double	8 bytes	-2^{63} to $+2^{63} - 1$
Boolean	1bit	true/false

2. Constants

- Constant are all values that are mixed in the program.
- Java is the strictly type language so is required to understand the constant for each data type.

1. Int constant: -

int constant will be written with the non-floating point numeric value either with the sign or without sign.

Ex: - 5, 6, -7, 0, -6

2. Byte and Short constant: -

There are no constant for the **byte** and **short** they use the integer constant within the range.

Ex: - byte b;
`b = 5; /* correct */`
`b = 130; /*not within range of byte*/`

```
int x;
x = 12;
byte b;
b = x; /* x is not int constant, it is the int variable */
```

3. Long Constant: -

long constant always be used with the suffix l and L. It can be used with the sign bit or without sign bit can't maintain the floating point values.

Ex: - 5l, 5L, -6l, 0l, etc.

4. Char Constant: -

char constant can be written in 3 ways.

- i. Within the '- '.
- ii. The ASCII or ISO Latin code.
- iii. Unicode: - Unicode constants the 4 hexadecimal digits prefixed with the \u.

Ex: -

- | | | |
|--------------------------------------------------------|---------------------------------------------------------|--------------------------------------------------------------|
| i. char ch;
ch = 'A';
Sop(ch);
O/P – A | ii. char ch;
ch = 65;
Sop (ch);
O/P – A | iii. char ch;
ch = \u0041;
Sop (ch);
O/P – A |
|--------------------------------------------------------|---------------------------------------------------------|--------------------------------------------------------------|

5. Float Constant: -

float constant always be suffered by the small f and F. It can be either negative or positive. In the float we can store maximum up to the 7 place of decimal.

Ex: - 5.5f, +4.5F, -3.2f, 0.87F, 5f, 4F, etc.

6. Double Constant: -

All the decimals values are the by default double constant. Optionally we can suffix the d and D. Double value can be up to the 15 places of the decimal.

Ex: - 5.5, 5.3, -4.3, 0.89d, 0.45D, 6.5D etc.

7. Boolean constant: -

Java provides to 2 keywords as the Boolean constant True/False.

Ex: - Boolean b;
b = true;
b = false;

Type Casting

The type casting is the way by which one type of values converted into another type.

There are Two Type of Casting

1. Implicit
2. Explicit

1. Implicit

- ❖ Programmer needs not to write any extra code for the casting.

- Ex 1: -** int x;
 x = 'A'; /* char to int*/
 Sop (x);
O/P - 65

2. Explicit

- ✚ We have to write some extra code in case of explicit conversion.

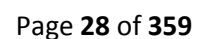
Cases where explicit conversion required

- Ex:** - float to int

- ```
Ex: - int to char
 int x;
 x = -65;
 char ch;
 ch = (char) x;
 Sop (ch);
 O/P -
```

- Ex: -** short to byte  
 short sh;  
 sh = 130;  
 byte b;  
 b = (byte) sh;  
 Sop (b);  
**O/P - -126**

### Implicit conversion chart



### Exception of above diagram: -

int constant can be stored in the byte, short and char type implicit within the range.

|         |          |                  |
|---------|----------|------------------|
| byte b; | char ch; | int x;           |
| b = 50; | ch = 65; | char ch;         |
|         |          | x = 65;          |
|         |          | ch = x; // wrong |

[**Note:** - Float and double always store the value in the exponential format that does way long can be short into the float implicit.]

### Explicit format

Ex: -      1 2 3 4 5 6 7 8 9

Sign → + 1. 2 3 4 5 6 7 8 9 \* 10<sup>8</sup> → exponent

                    Mantissa      base

float (32 bytes)

    → 1 bit (sign)

    → 8 bits (exponent)

    → 23 bits (mantissa)

float ft;

ft = 123456;

| Sign | mantissa       | exponent |
|------|----------------|----------|
| +    | 1. 2 3 4 5 6 7 | 5        |

Value of stored int  
binary format

long lg;

lg = 123456789123l;

float ft;

long lg2;

lg2 = (long) ft;

Sop (lg2);

### Type Casting in case of expression

[**Note:** - If in any expression there are the different types of operands involved then the type's conversion is required.]

**Ex: -**      int res = (int) (5 + 5L + 5.5F + 5.5D);

The result of above expression will be generated in the double type, because double is the largest type in the expression.

|              |                                                                   |                                                          |
|--------------|-------------------------------------------------------------------|----------------------------------------------------------|
| <b>Ex: -</b> | 1.    float ft;<br>ft = 10/4;<br>Sop (ft);<br>O/P – 2             | 2.    float ft;<br>ft = 10/4F;<br>Sop (ft);<br>O/P – 2.5 |
| <b>Ex: -</b> | 3.    float ft;<br>ft = 2.5; // error because 2.5 is double value | 4.    float ft;<br>ft = 10/4.0;                          |

**Ex: -**     5.    byte b1 = 4, b2 = 5, b3;  
              b3 = b1 + b2; // error (here is int constant)  
              Sop (b3);

**Ex: -**     6.    byte b1 = 4, b2 = 5, b3;  
              b3 = (byte) (b1 + b2); // correct  
              b3 = b1; // correct  
              Sop (b3);

## **Variables: -**

Variable are the name of memory location.

Variables are the container for the values.

**There are two types of operations on any variables.**

1. Read
2. Write

## **Rules for naming the variable in java**

1. Any variable name can contains the alphabets, digits, underscore (\_) and dollar (\$) sign.
2. The variable name can't be started with digits.
3. Variable name can't be any keywords.
4. Variable are the not case sensitive.
5. Blank space is not allowed.
6. Valid variable name.  
int rollNo;  
int Roll\_no;  
int rollno123;  
int \$roll;  
int \_roll;
7. Invalid variable name.  
int roll no;  
int char;  
int 123roll;  
int roll@no;

## **Keywords: -**

- ❖ Keywords are the reserved words. The reserve Words that has their special meaning.
- ❖ All the keywords in java are started in the small letters.
- ❖ There are 50 keywords in java and 53 reserved keywords.

## These Keywords are:-

| Data Type | Flow Control | Modifier     | Exception Handling | Class Related | Object Related |
|-----------|--------------|--------------|--------------------|---------------|----------------|
| Byte      | if           | Public       | try                | Class         | New            |
| Short     | Else         | Private      | Catch              | interface     | instanceof     |
| Int       | switch       | Protected    | throw              | Extends       | Super          |
| Long      | Case         | Static       | throws             | implements    | this           |
| Float     | Default      | Final        | Finally            | Package       |                |
| Double    | Do           | Abstract     | assert***          | import        |                |
| Char      | while        | synchronized |                    |               |                |
| Boolean   | for          | strictfp**   |                    |               |                |
|           | Break        | transient    |                    |               |                |
|           | Continue     | Native       |                    |               |                |
|           | Return       | volatile     |                    |               |                |
| 8         | 11           | 11           | 6                  | 6             | 4              |

| Return Type | Unused keywords |          | Reserve word |
|-------------|-----------------|----------|--------------|
| Void        | goto*           | enum**** | true         |
|             | const*          |          | False        |
|             |                 |          | Null         |
| 1           | 2               | 1        | 3            |

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

## Operators: -

Operators are used to perform the operations according to the number of operands they can be 3 types of operators.

1. **Unary Operator** ----- One Operand
2. **Binary Operator** ----- Two Operand
3. **Ternary Operator** ----- Three Operand

## **Operators in Java**

1. Arithmetic Operator
2. Relational Operator
3. Logical Operator
4. Increment and Decrement Operator
5. Ternary Operator

## 1. Arithmetic Operator

Arithmetic Operators are used in mathematical operations.

| Operator | Results        |
|----------|----------------|
| +        | Addition       |
| -        | Subtraction    |
| *        | Multiplication |
| /        | Division       |
| %        | Modules        |

|      | Priority | Associatively |
|------|----------|---------------|
| High | /, *, %  | Left to right |
| Low  | +, -     | Left to right |

## 2. Relational Operator

Relational Operator is also binary operator the result of relational operators are always the Boolean values.

| Operator | Result                |
|----------|-----------------------|
| ==       | Equal to              |
| !=       | Not equal to          |
| >        | Greater then          |
| <        | Less then             |
| >=       | Greater than equal to |
| <=       | Less than equal to    |

[**Note:** - relational operators are non-associative, accept == & !=. ]

|       |                          |                                |
|-------|--------------------------|--------------------------------|
| Ex: - | boolean b                | int x;                         |
|       | b = 5 > 4;               | x = 5 > 4; // error            |
|       | Sop (b);                 |                                |
|       | <b>O/P – true</b>        |                                |
|       | int x = 6, y = 8, z = 9; | int x = 6, y = 8, z = 9;       |
|       | boolean b;               | boolean b;                     |
|       | b = x > y < z; // error  | b = x > y == false; // correct |

## 3. Logical Operator

Operands of the logical operator are always Boolean and there results are also Boolean.

| Operator | Result |        |
|----------|--------|--------|
| &&       | Binary | Binary |
|          | Or     | Binary |
| !        | Not    | Unary  |

| Operator1 | Operator2 | &&    |       | !     |
|-----------|-----------|-------|-------|-------|
| True      | True      | True  | True  | False |
| True      | False     | False | True  | False |
| False     | True      | False | True  | True  |
| False     | False     | False | False | True  |



**Ex: -**        `int x = 5, y = 10;`  
                  `boolean b;`  
                  `b = x > y / 5 && y < x * 4; // O/p – True`  
                  `b = !(x > y); // O/p – True`

#### **4. Increment and Decrement Operator**

These operators are used to increase and decrease the value of variable by 1.

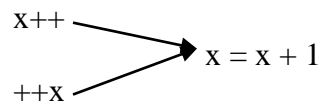
**Ex: -**        `int x = 5;`  
                  `x++; or ++x; // separate statement line`  
                  `Sop (x);`  
                  **O/P – 6**

#### **There are Two Types of Increment and Decrement Operators:**

- i.    **Pre: -**        `++x, --x`
- ii.   **Post: -**       `x++, x--`

There is the no difference between **pre** and **post** when they are used as the single statement.

[**Note: - Pre and Post** both are converted into the `x = x + 1`.]



The difference between **pre** and **post** arise then they are used in any expression.

|                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Ex: - 1.</b>   <code>int x = 5, y;</code><br/>                    <code>y = x++;</code><br/>                    <code>Sop (x); // O/P – 6</code><br/>                    <code>Sop (y); // O/P – 5</code></p> <p style="margin-left: 40px;"> <code>y = x++;      y = x</code><br/> <span style="margin-left: 100px;"><code>x = x + 1</code></span> </p> | <p><b>2.</b>        <code>int x = 5, y;</code><br/>                    <code>y = ++x;</code><br/>                    <code>Sop (x); // O/P – 6</code><br/>                    <code>Sop (y); // O/P – 6</code></p> <p style="margin-left: 40px;"> <code>Y = ++x;      x = x + 1</code><br/> <span style="margin-left: 100px;"><code>y = x</code></span> </p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Ex: -</b>    <code>int x = 5, y = 10, z;</code><br/>                    <code>z = x++ + ++y;</code><br/>                    <code>Sop (x); // O/P – 6</code><br/>                    <code>Sop (y); // O/P – 11</code><br/>                    <code>Sop (z); // O/P – 16</code></p> | <p><code>int x = 5, z;</code><br/>                    <code>z = x++ + ++x + x++;</code><br/>                    <code>Sop (x); // O/P – 6</code><br/>                    <code>Sop (z); // O/P – 19</code></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Pre:        → Increase then use**  
**Post:      → use then increase**

|                                                                                                                                                                                                                                                                                                        |                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| <p><b>Ex: -</b>    <code>int x = 5, y = 10, z;</code><br/>                    <code>z = ++x + --y – x++ + y++;</code><br/>                    <code>Sop (x); // O/P – 6</code><br/>                    <code>Sop (y); // O/P – 10</code><br/>                    <code>Sop (z); // O/P – 18</code></p> | <p><code>int x = 5, y = 10;</code></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|

|                                                                                                                                                                                                                             |                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ex: -</b> <code>int x = 5, y = 10;</code><br><code>boolean b = ++x &gt; y/2 &amp;&amp; y-- == x*2;</code><br><code>Sop (x); // O/P - 7</code><br><code>Sop (y); // O/P - 9</code><br><code>Sop (z); // O/P - True</code> | <code>int x = 5, y = 10;</code><br><code>boolean b = ++x &lt; y/2 &amp;&amp; y-- == x*2;</code><br><code>Sop (x); // O/P - 7</code><br><code>Sop (y); // O/P - 10</code><br><code>Sop (z); // O/P - True</code> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 5. Ternary Operator

It uses the three operands basically these operators use to change the result of the relational operator.

**Syntax: -**

**Relational expression ? True evaluation : False evaluation**

**Ex: -**    `int x = 10, y = 20;`  
           `char ch = x > y ? 'x' : 'y' ;`  
           `Sop (ch);     // O/P - y`

# Object Oriented Programming Structure (oops)

Object Oriented Programming is the approach of the programming in which each and every thing is represent by an object.

## Features of OOPs

1. **Class and Object**
2. **Encapsulation and Abstraction**
3. **Inheritance and Polymorphism**

### 1. Class and Object: -

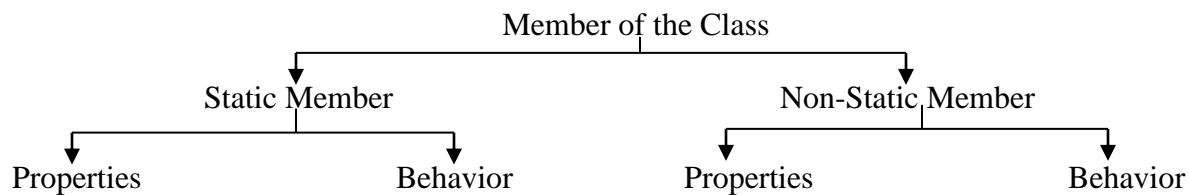
**Object:** - Object is a real world entity that has their specific properties and behaviors.

**For Example:** - Any particular instance of car is known as the object.

**Class:** - Class is the blue print of the object that means class is the definition of the object in which properties and behaviors of the class are defines.

**For Example:** - Car is the class but any particular instance of the car is known as the object.

- Class is the logical representation of anything.
- Object is the physical representation of anything.
- In the class there is the definition of the properties and behavior of the object.
- In any class there can be two types of properties and behavior.
  - i. Static
  - ii. Non-Static

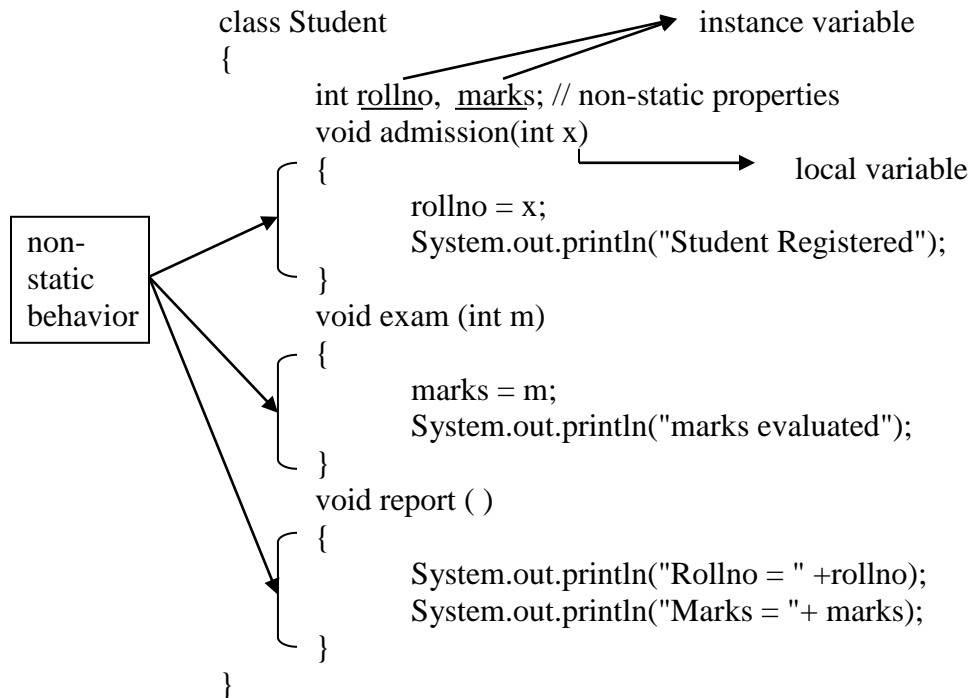


- ✚ Static member always be used without object that means without making any object we can use the static member of the class.
- ✚ Non-Static member can access be used without object that means we have to create an object an object access the non-static member.
- ✚ Properties are use to represent the data of the object.
- ✚ Properties are use to maintain the state of all object.
- ✚ State of the object of the condition in which the object may resides.

**For Ex:** - In case of human object, there can be main properties

  - i. Height
  - ii. Weight
- ✚ Behaviors are the functionality of the object.
- ✚ A behavior performs the operations on properties of the object and generates the output.
- ✚ Programmatically properties are implemented by the variable and behaviors are implemented by the methods (function).

**Ex: -**



**Class Diagram**

|                                                                                                       |                                                                                         |
|-------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| Class Student                                                                                         |                                                                                         |
| <ul style="list-style-type: none"> <li>rollno</li> <li>marks</li> </ul>                               | Non-Static variables <b>or</b> Instance variable <b>or</b> non-static fields/properties |
| <ul style="list-style-type: none"> <li>Admission ( )</li> <li>Exam ( )</li> <li>Report ( )</li> </ul> | Non-Static methods <b>or</b> Non-static behaviors                                       |

## 2. Encapsulation and Abstraction

**Encapsulation: -**

- ❖ Encapsulation wrapping up the properties of the object with the behavior which is known as the encapsulation.
- ❖ In the above student class rollno and marks property are used in the behaviors.

**Abstraction: -**

- ❖ Hiding the background details and opens the essential thing is known as the abstraction.
- ❖ The method calling provides the abstraction because what the output will be generated we known but how that output actually generated we don't know.

**Ex: -** In case of above student class when the behavior are involved by the object we achieve the abstraction.

## 3. Inheritance and Polymorphism

**Inheritance: -**

- Inheritance provides the reusability that means besides to develop the each and everything is new it better to reuse already develop things.

- In the java by making the any class child of any other class we can reuse the properties and behavior of the parent class in the child class.

### **Polymorphism: -**

- Many form of one thing is the polymorphism one name and many forms can also be known as the polymorphism.

**Ex:** - println ( ) method, there are 10 println ( ) method to print the different types of values but all have the same name println ( ).

|                     |                        |
|---------------------|------------------------|
| Println (int x);    | S.o.println (50);      |
| Println (string s); | S.o.println ("Hello"); |
| Println (float f);  | S.o.println (5.5f);    |

In order to use a non-static member of the student class we have to create the object of the student class.

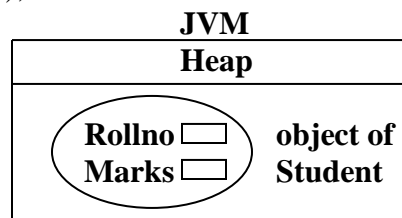
### **new keyword**

**new** keyword in java is used to create the new object in the heap memory.

**Syntax: -**

**new <class name>( );**

**Eg:**     new Student ( );



In order to create and use the object of the student class we have to create the main ( ) method either in the same class or in another class.

### **//Student.java**

```

class Student
{
 int rollno, marks;
 void admission(int r)
 {
 rollno = r;
 System.out.println("Student Registered");
 }
 void exam (int m)
 {
 marks = m;
 System.out.println("marks evaluated");
 }
 void report()
 {

```

```

 System.out.println("Rollno = " + rollno);
 System.out.println("Marks = " + marks);
 }
}
//School.java
class School
{
 public static void main(string s[])
 {
 Student st;
 st = new Student (); //here new is object creation
 st.Admission (101);
 st.Exam (85);
 st.Report ();
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\School>javac Student.java
E:\JAVA\Programs\Core Java Program\Class\School>javac School.java
E:\JAVA\Programs\Core Java Program\Class\School>java School
Student Registered
marks evaluated
Rollno = 101
Marks = 85
E:\JAVA\Programs\Core Java Program\Class\School>_

```

Save the above code of the xyz.java. (User define name can be any name but the recommended name is the class name).

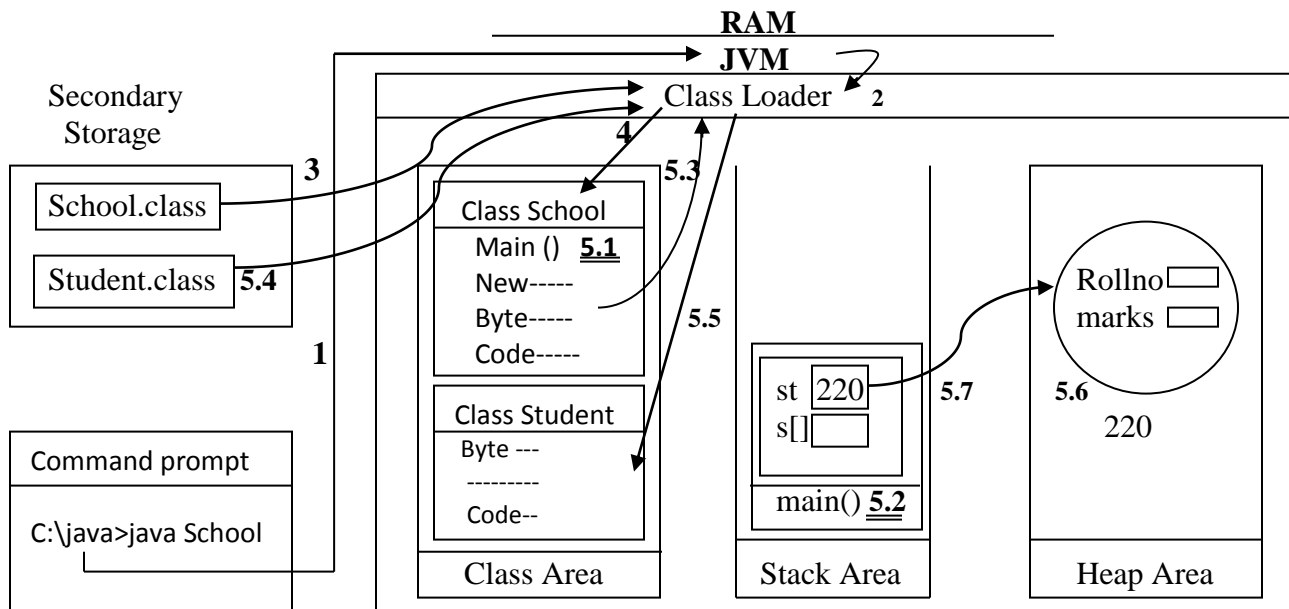
## Reference Variable

- Reference variable is the type of variables that always stores the reference id of the objects.
- The reference id always be generated through the actual address of the object that why there is always different reference id of the object.
- Reference variable always be declared as the class type, interface type, array type, enumeration type.
- Size of the reference variable always is the 4 bytes in the memory.
- One reference variable can contain only one reference id at a time.
- **st.Student ( );**
- This statement creates a new object and assigns the reference id into reference variable.
- **St.Admission (101);**  
 Reference variable providing the reference id of the object to the method
- Through which object any no-static method is invoked within that method the properties of that same object will be used.
- The reference id of the object always is generated in the form of hexadecimal values.
- Format of the reference id is

**Classname@hexadecimal.**

**Ex: - System.out.println (st);**

- It is always recommended by java to make separate **.java** files for the separate class and save the java file with the same name as the class name.
- If we compile one **.java** file another **.java** file automatically gets compile.
- Execution of the last program within the JVM.



1. JVM gets started in the memory.
2. Class loader invoke by the JVM to load the School class.
3. Byte code of the School class retrieve by the class loader.
4. Byte code is copied into the class area.
- 5.1. Main () method is started.
- 5.2. Frame for the main () method created in the stack area.
- 5.3. New keyword encountered to create the object of student class so class loader is invoked to load the student class.
- 5.4. Byte code retrieves.
- 5.5. Byte code copy with in the class area.
- 5.6. Object of the student is created.
- 5.7. Reference id of object is stored in the st variable.

✚ Storing the reference id of object into the more than one reference variable.

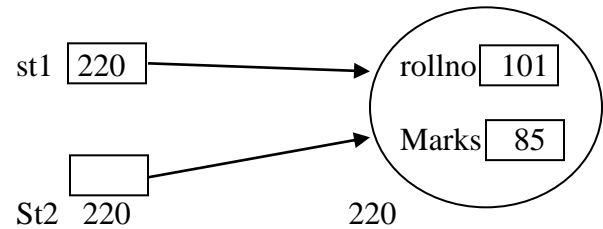
✚ Changes in the main ( ) method program.

```
class School1
{
 public static void main (String []s)
 {
 Student st1, st2;
 st1 = new Student ();
 st1 = st2; //copy the reference id from st1 to st2.
 st1.admission (101);
 st2.exam (85);
 }
}
```

```

 st1.report ();
 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\School>javac Student.java
E:\JAVA\Programs\Core Java Program\Class\School>javac School1.java
E:\JAVA\Programs\Core Java Program\Class\School>java School1
Student Registered
marks evaluated
Rollno = 101
Marks = 85
E:\JAVA\Programs\Core Java Program\Class\School>

```

[Note: - in the java all the non-static variables always have their default value.]

As for rules: -

Primitive type: -

Byte, Short, Int, Long, Float, Double, Char → 0  
 Boolean → False

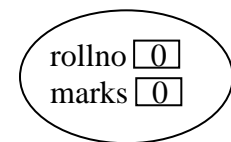
Reference variable: → NULL

Change in the main ( ) method.

```

class School2
{
 public static void main (String []s)
 {
 Student st1, st2;
 st1 = new Student (); // new object created
 st1.Report ();
 }
}

```



```

E:\JAVA\Programs\Core Java Program\Class\School>java School2
Rollno = 0
Marks = 0

```

**Question: -** Create a class employee with the following properties and behavior.

class Employee

**Properties: -** int code, tax, netsalary

**Behavior: -** void join (int c)

[To set the code]

void SetSalary (int sal)

[To set the salary]



```
void Calculate ()
[To set the tax] [10% of salary]
void Report ()
[To show the report]
```

**//Employee.java**

```
class Employee
{
 int code, tax, netsalary;
 void Join(int c)
 {
 code = c;
 System.out.println("Employee Registered");
 }
 void Setsalary(int sal)
 {
 netsalary = sal;
 System.out.println("Employee Salary Added");
 }
 void Calculate()
 {
 tax = netsalary * 10/100;
 System.out.println("Employee tax paid");
 }
 void Reports()
 {
 System.out.println("Employee code = "+code);
 System.out.println("Employee net salary = "+ netsalary);
 System.out.println("Employee income-tax = "+ tax);
 }
}
```

**//Company.java**

```
class Company
{
 public static void main(String []s)
 {
 Employee emp;
 emp = new Employee();
 emp.Join(200);
 emp.Setsalary(9000);
 emp.Calculate();
 emp.Reports();
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Company>javac Employee.java
E:\JAVA\Programs\Core Java Program\Class\Company>javac Company.java
E:\JAVA\Programs\Core Java Program\Class\Company>java Company
Employee Registered
Employee Salary Added
Employee tax paid
Employee code = 101
Employee net salary = 9000
Employee income-tax = 900
E:\JAVA\Programs\Core Java Program\Class\Company>_
```

[Note: - In one reference variable at a time there can be the reference id of any one object stored.]

### Change in the main () method

class School3

```
{
 public static void main (String []s)
 {
 Student st1;
 st1 = new Student (); // new object created
 st1 = new Student (); // new object created
 st1.report ();
 }
}
```

```
E:\JAVA\Programs\Core Java Program\Class\School>java School3
Rollno = 0
Marks = 0
```

- ❖ After the creation of 2<sup>nd</sup> object and referred by the st1 the 1<sup>st</sup> object (220 ID) because unreferenced.
- ❖ Unreferenced object: -Is the object whose reference id does not hold by any reference variable.
- ❖ That means any object in the heap memory whose reference id is not containing by any reference variable.

[Note: - There is the utility in the JVM none as GARBAGE COLLECTION the work of garbage collector is to destroy the unreferenced object from the heap area.]

**There are number of ways by which any object becomes the unreferenced some are as follows: -**

1. When any object is referenced by any single reference variable and the reference id of the new object stored into that reference variable. (as discussed in the last program)
2. Creating the object new keywords without placing the catching reference variable in the left side of assignment operator.  
public static void main (String []s)

```

{
 new Student (); // correct unreferenced variable
}

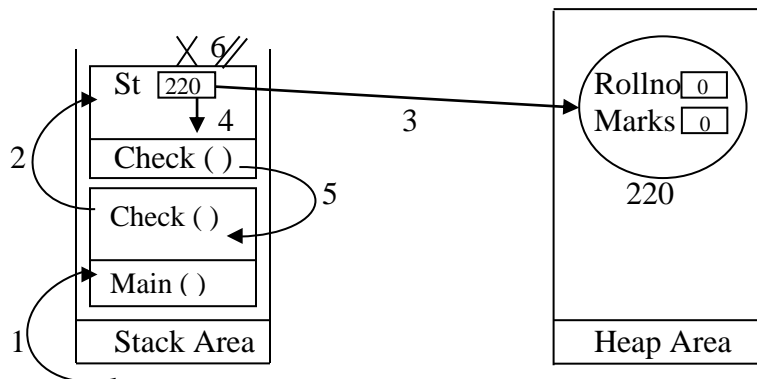
```

3. In case of method calling when the reference variable is the local variable then as the method completed the local reference variable gets destroyed and object becomes unreferenced.

```

class School4
{
 public static void main(String[] args)
 {
 System.out.println("welcome");
 check(); //method calling
 System.out.println("Thank you");
 }
 static void check()
 {
 Student st1;
 st1 = new Student();
 st1.admission(101);
 st1.exam(85);
 st1.report();
 }
}

```



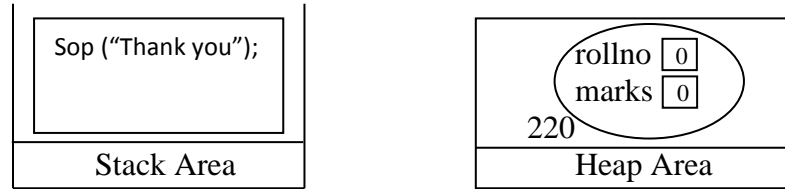
```

E:\JAVA\Programs\Core Java Program\Class\School>java School4
welcome
Student Registered
marks evaluated
Rollno = 101
Marks = 85
Thank you

```

1. Main () method invoked and their frame is created.
2. Check () method invoked and their frame created.
3. Object of Student class created and reference id stored into st.
4. Execution of the check () method.
5. Check () method gets compiled and control is return in the main () .
6. Frame of the check method destroyed and st local variable also gets destroyed.

- After the execution of check ( ) method the Student object becomes unreferenced.



**Question:** - Creating a class for the practice of OOPS.

**//Book.java**

class Book

```
{
 int code, price;
 boolean status;
 void setDetail(int c, int p)
 {
 code = c;
 price = p;
 status = true;
 }
 void issueBook()
 {
 if(status)
 {
 System.out.println("Book issued successfully");
 status = false;
 return;
 }
 System.out.println("Book is not issued");
 }
 void returnBook()
 {
 if(!status)
 {
 System.out.println("Book returned successfully");
 status = true;
 return;
 }
 System.out.println("Book is already available");
 }
 void checkStatus()
 {
 System.out.println("Book code = "+code);
 System.out.println("Book Price = "+ price);
 if (status)
 {
 System.out.println("Status : available");
 }
 }
}
```

```

 else
 {
 System.out.println("Status : Issue");
 }
 }
}

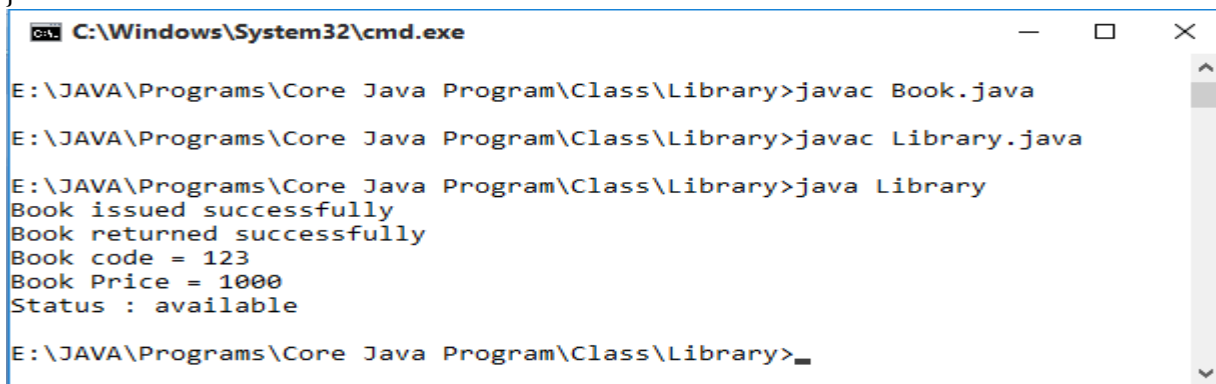
```

**// Library.java**

```

class Library
{
 public static void main(String []args)
 {
 Book b = new Book();
 b.SetDetail(123, 1000);
 b.IssueBook();
 b.ReturnBook();
 b.CheckStatus();
 }
}

```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user is in the directory "E:\JAVA\Programs\Core Java Program\Class\Library". The commands and their outputs are as follows:

```

E:\JAVA\Programs\Core Java Program\Class\Library>javac Book.java
E:\JAVA\Programs\Core Java Program\Class\Library>javac Library.java
E:\JAVA\Programs\Core Java Program\Class\Library>java Library
Book issued successfully
Book returned successfully
Book code = 123
Book Price = 1000
Status : available
E:\JAVA\Programs\Core Java Program\Class\Library>_

```

**// another class Library1.java**

```

class Library1
{
 public static void main(String []args)
 {
 Book b = new Book();
 b.CheckStatus(); // O/P – code = 0, price = 0, status : Issued
 b.ReturnBook(); // O/P – Book returned successfully
 b.IssueBook(); // O/P – Book issued successfully
 b.CheckStatus(); // O/P – code = 0, price = 0, status : issued
 }
}

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Library>javac Library1.java

E:\JAVA\Programs\Core Java Program\Class\Library>java Library1
Book code = 0
Book Price = 0
Status : Issue
Book returned successfully
Book issued successfully
Book code = 0
Book Price = 0
Status : Issue

E:\JAVA\Programs\Core Java Program\Class\Library>_
```

// another class Library2.java

```
class Library2
{
 public static void main(String []args)
 {
 Book b1, b2;
 b1 = new Book();
 b2 = new Book();
 b1.SetDetail(123, 1000);
 b2.IssueBook(); // O/P – Book is not issued
 b1.ReturnBook(); // O/P – Book is already available
 b2 = b1;
 b2.ReturnBook(); // O/P – Book is already available
 b1.CheckStatus(); // O/P – code = 123, price = 1000, status : available
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Library>javac Book.java

E:\JAVA\Programs\Core Java Program\Class\Library>javac Library2.java

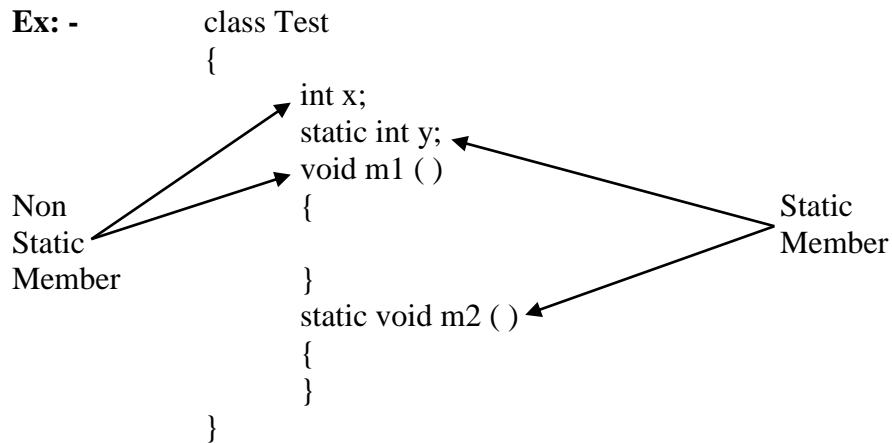
E:\JAVA\Programs\Core Java Program\Class\Library>java Library2
Book is not issued
Book is already available
Book is already available
Book code = 123
Book Price = 1000
Status : available

E:\JAVA\Programs\Core Java Program\Class\Library>_
```

## **Static Modifier (keywords)**

- Static in java is the keyword that works at the modifier.
- Modifiers change the accessibility pattern of the member of the class or the class itself.
- Static keywords can be use with
  - ✓ Variable
  - ✓ Methods
  - ✓ Nested classes (class within class)

**Ex: -**



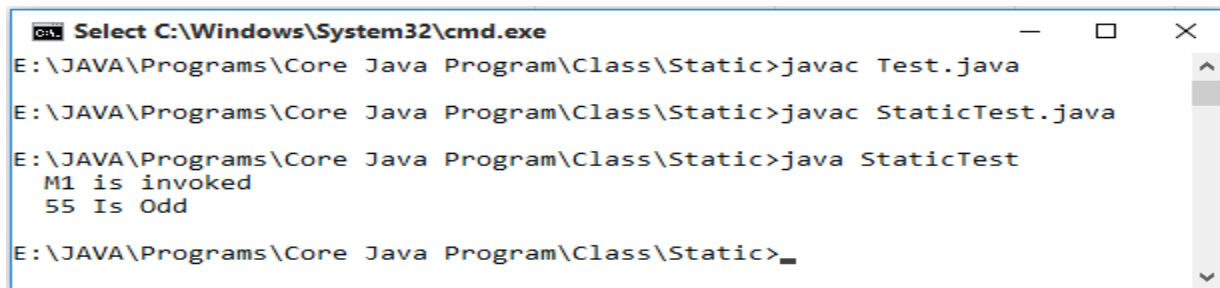
**[Note: -** Static member of the class always be accessed by the class name outside the class.]

**//Test.java**

```
class Test
{
 static int y;
 static void m1(int val)
 {
 y = val;
 System.out.println("M1 is invoked");
 }
 static void m2()
 {
 if (y % 2 == 0)
 {
 System.out.println(y + " Is Even");
 }
 else
 {
 System.out.println(y + " Is Odd");
 }
 }
}
```

**//StaticTest.java**

```
class StaticTest
{
 public static void main(String []s)
 {
 Test.m1(55);
 Test.m2();
 }
}
```



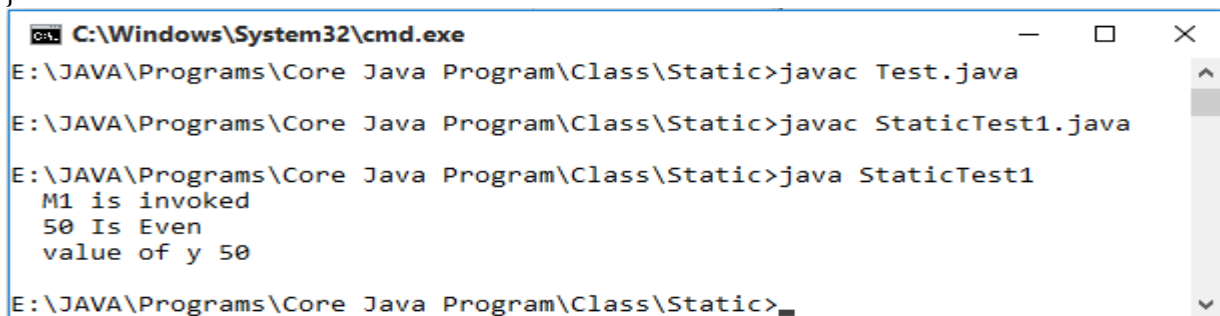
```
Select C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Static>javac Test.java
E:\JAVA\Programs\Core Java Program\Class\Static>javac StaticTest.java
E:\JAVA\Programs\Core Java Program\Class\Static>java StaticTest
M1 is invoked
55 Is Odd
E:\JAVA\Programs\Core Java Program\Class\Static>_
```

[Note: - We can also access also the static variable outside the class with the class name.]

### In the main( ) method: -

class StaticTest1

```
{
 public static void main(String []s)
 {
 Test.m1(50);
 Test.m2();
 System.out.println(" value of y " + Test.y);
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Static>javac Test.java
E:\JAVA\Programs\Core Java Program\Class\Static>javac StaticTest1.java
E:\JAVA\Programs\Core Java Program\Class\Static>java StaticTest1
M1 is invoked
50 Is Even
value of y 50
E:\JAVA\Programs\Core Java Program\Class\Static>_
```

- The static member of the class can also be accessed by the reference variable but in actual with the accessed by the class name.
- Compilers change the reference variable in class name in the byte code.

### Changes in the main ( ) method: -

class StaticTest2

```
{
 public static void main(String []s)
 {
 Test t = new Test();
 t.m1(55);
 t.m2();
 }
}
```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Static>javac Test.java
E:\JAVA\Programs\Core Java Program\Class\Static>javac StaticTest2.java
E:\JAVA\Programs\Core Java Program\Class\Static>java StaticTest2
M1 is invoked
55 Is Odd
E:\JAVA\Programs\Core Java Program\Class\Static>

```

## **Javap Tool**

- This tool is available in the JDK that always works upon the .class file.

### **Syntax**

**C:\java>javap <class name>**

**Ex:-**

**C:\java>javap Student**

```

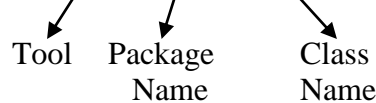
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\School>javap Student
Compiled from "Student.java"
class Student {
 int rollno;
 int marks;
 Student();
 void admission(int);
 void exam(int);
 void report();
}

```

- Javap tool is use to list out the details of the class such as the list of the name of variables and signatures of the methods.
- By the java p tool we can also list out the details of the members of the class in java library.

**Ex: -**

**C:\java>javap java.lang.String**



- If any class is in the package then we have to write the class name with package name.
- In java library all the classes are available in the certain package.
- Till now all our classes are created without package.

**C:\java>javap -c Switch**

- Javap tool with the hyphen c (-c) switch not only display the member list rather it also display the complete assembly instruction available in the .class file

**C:\java>javap -c StaticTest**

**Ex:**

```

class StaticTest
{
 public static void main(String []s)
 {
 Test.m1(55);
 }
}

```

```

 Test.m2();
 }
}

```

```

C:\Windows\System32\cmd.exe

E:\JAVA\Programs\Core Java Program\Class\Static>javap -c StaticTest
Compiled from "StaticTest.java"
class StaticTest {
 StaticTest();
 Code:
 0: aload_0
 1: invokespecial #1 // Method java/lang/Object
 2: <init>:()V
 3: return

 public static void main(java.lang.String[]);
 Code:
 0: bipush 55
 1: invokestatic #2 // Method Test.m1:(I)V
 2: invokestatic #3 // Method Test.m2:()V
 3: return
}

```

**[Note: -** Class member = class properties + class behavior  
 Properties are variable  
 Behaviors are methods.]

## **Static Variable**

- Static variables are created at the time of class loading.
- There is only one copy of static variable created.
- Static variable always be created in the class area.
- Static variable can be used in the non-static methods of the same class.
- There is only one copy of the static variable created in the class area which is shared by the objects of the class.

```

class Test1
{
 int x;
 static int y;
 void m1()
 {
 x = 20;
 y = 30;
 }
}

```

- Purpose of the static variable in any class is to shared the data between the objects of that class.
- Static also have their default value.

```

//Test2.java
class Test2

```

```

{
 int x;
 static int y;
 void m1 (int val)
 {
 x = val;
 y = y + val;
 }
 void show()
 {
 System.out.println ("x : = " + x);
 }
}

//main class
class MainDemo
{
 public static void main (String []s)
 {
 Test2 t1 = new Test2();
 Test2 t2 = new Test2();
 Test2 t3 = new Test2();
 t1.m1(10);
 t2.m1(20);
 t3.m1(40);
 t1.show();
 t2.show();
 t3.show();
 System.out.println("y : = " + Test2.y);
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Class\Static>javac Test2.java
E:\JAVA\Programs\Core Java Program\Class\Static>javac MainDemo.java
E:\JAVA\Programs\Core Java Program\Class\Static>java MainDemo
x : = 10
x : = 20
x : = 40
y : = 70
E:\JAVA\Programs\Core Java Program\Class\Static>

```

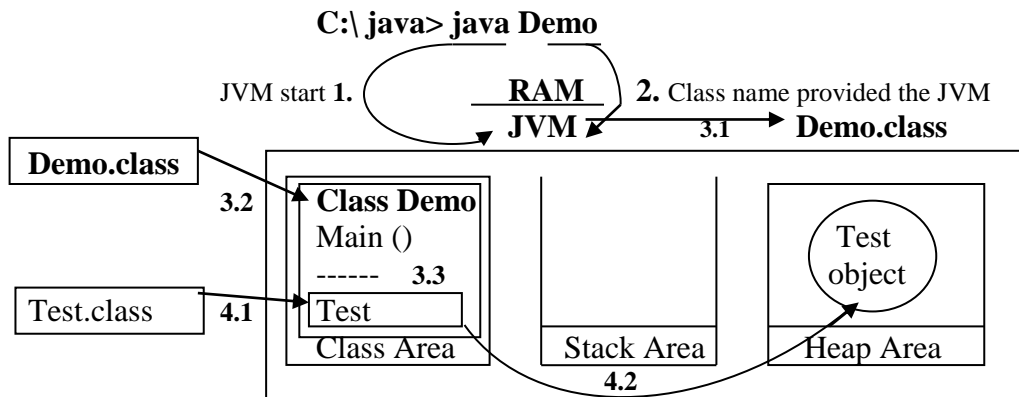
## Class Loading

Class loading is the process through which any class gets loaded from the hard disk to the class area in the JVM.

**Question: -** When any class loaded in the JVM?

**Answer: -** When the first objects of the class created or first time any static member with class name is accessed whichever is earlier immediately class gets loaded in the JVM.

- Static variable always gets the memory at the time of class variable.
- Before the creation of the first object of any class gets loaded.
- That means before the creation of first object the static variable are created.



1. JVM gets started and loaded into the RAM.
2. Class name provided to the JVM.
- 3.1 Main () method called by the JVM.
- 3.2 Before the starting of main () method the class Demo class loaded in the class area.
- 3.3 Form the main () method new keyword encountered to create the first object of the Test class.
- 4.1 Before creating the Test object the Test class gets loaded in the class area.
- 4.2 Then the Test object is created.

**Question: -** Why the static variable can be used in the non-static methods?

**Answer: -** There can be 2 main issues regarding to class any variable.

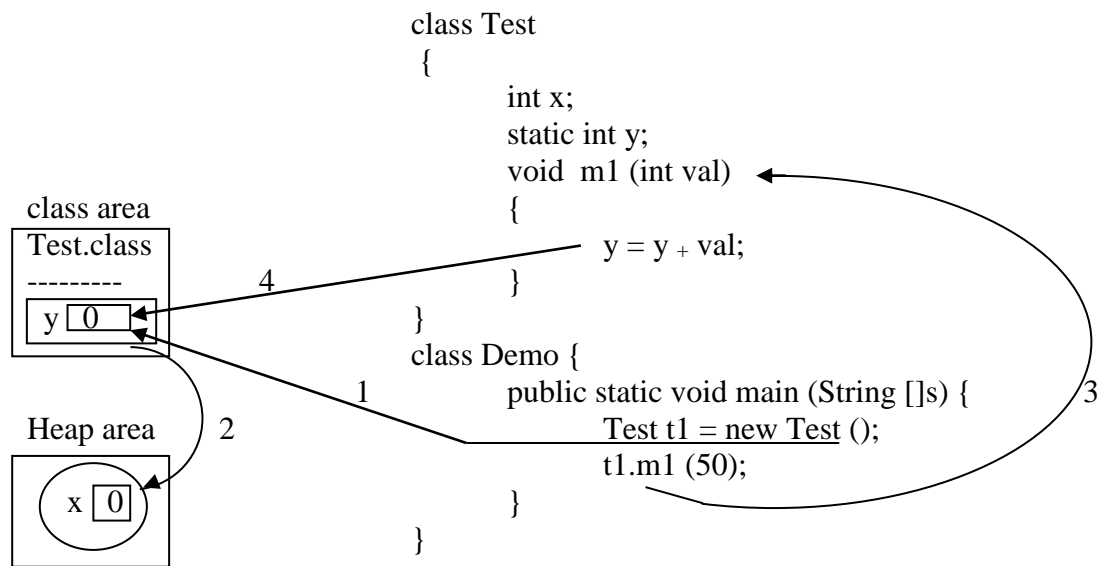
**1<sup>st</sup> issue: -** Is that variable already gets the memory, that means variable declaration executed or not.

**2<sup>nd</sup> issue: -** The actual memory address where the variable is created.

Non-static method always be invoked with the object and before the object created the class already gets loaded and static variable created.

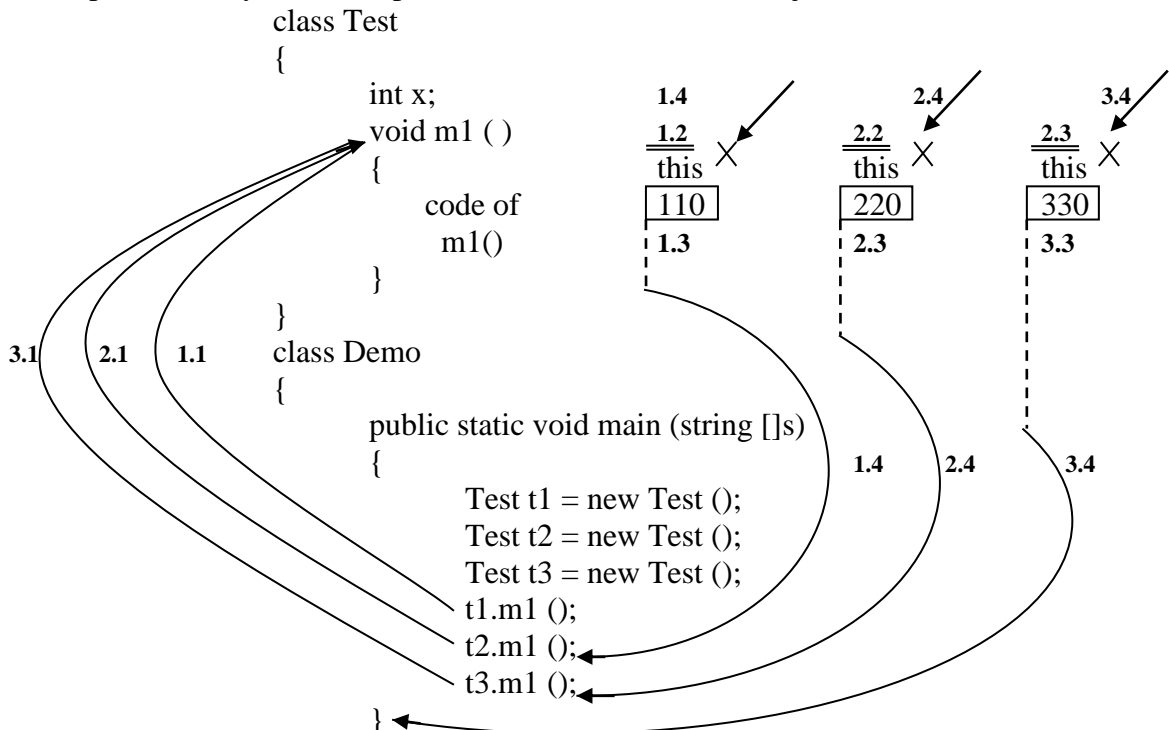
Both the above issue will not occurred when we will use the static variable in the non-static method of the same class why?

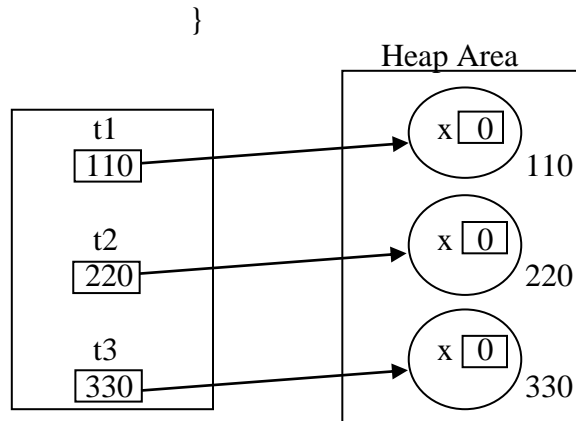
Non-static method always be invoked with the object and before the object creation the class already gets loaded and static variable created.



### this keyword: -

- this keyword always be used in the non-static methods and constructors.
- This keyword is use to represent the current calling object in the non-static method.
- We can print this keyword as it prints the reference id of the object.





1.1 m1 () method is invoked by the 1<sup>st</sup> object.

1.2 Reference id of 1<sup>st</sup> object is loaded into **this**.

1.3 Execution of the method.

1.4 Execution of the method gets finished and control is returning after that this keyword destroys.

2.1 m1 () method is invoked again with the second object.

2.2 Reference id of 2<sup>nd</sup> object is loaded into **this**.

2.3 Execution of the method.

2.4 Execution of the method gets finished and control is returning after that this keyword destroys.

3.1.m1 () method is invoked again with the second object.

3.2.Reference id of 3<sup>rd</sup> object is loaded into **this**.

3.3.Execution of the method.

3.4.Execution of the method gets finished and control is returning after that this keyword destroys.

**Question: -** How the reference id of the invoking object reached into the non-static method within **this keyword**?

- Through the additional code of the compiler in the byte code the reference id of the object reached in the non-static method the **this keyword**.
- After the compiling the following additional code inserted into the byte code.

**Source code**

```

class Test
{
 int x;
 void m1 (int val)
 {
 x = val;
 }
}
class Demo
{
 public static void main (String []s)
 {

```

**Byte code**

```

class Test
{
 int x;
 void m1 (int val, Test this)
 {
 this.x = val;
 }
}
class Demo
{
 public static void main (String []s)
 {

```

```

 Test t1 = new Test ();
 t1.m1 (50);
 }
}

 Test t1 = new Test ();
 t1.m1 (50, t1);
 }
}

```

- In the bypassing the reference variable as an extra argument compiler sent the reference id of calling object in the non-static.

## Data Shadowing

- When the name of the instance variable (non-static variable) is name as the name of the local variable this is known as the data shadowing.
- In that case the instance variable shadowing by the local variable.

```

class Test
{
 int x, y; //instance variable
 void set ()
 {
 int x; //local variable
 x = 10; //local x will be used
 this.y = 20; // instance y is used
 this.x = 30; //instance x will be used (forcefully use through this keyword)
 }
 void show ()
 {
 System.out.println ("x = " + this.x);
 System.out.println ("y = " + this.y);
 }
}

```

// main class

```

class Demo
{
 public static void main (String []s)
 {
 Test t1 = new Test ();
 t1.set ();
 t1.show ();
 }
}

E:\JAVA\Programs\Core Java Program\Data Shadowing>javac Test.java

E:\JAVA\Programs\Core Java Program\Data Shadowing>java Demo
x = 30
y = 20

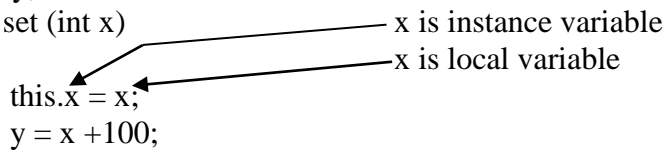
```

**[Note: -** If we want to access the instance variable by skipping the local variable then we have to explicitly write the **this keyword**.]

[**Note:** - The parameters are also being local variables of the method so data shadowing can be accrued with the parameters.]

//Test.java

```
class Test
{
 int x, y;
 void set (int x)
 {
 this.x = x;
 y = x + 100;
 }
 void show ()
 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}
```



//Main class

```
class Demo1
{
 public static void main (String []s)
 {
 Test1 t1 = new Test1();
 t1.set (20);
 t1.show ();
 }
}
```

```
E:\JAVA\Programs\Core Java Program\Data Shadowing\Test1>javac Test1.java
```

```
E:\JAVA\Programs\Core Java Program\Data Shadowing\Test1>java Demo1
```

```
x = 20
y = 120
```

## Data Shadowing with the Static Variable

- With the static variable data shadowing always be resolved with the help of class name.

```
class Test
{
 static int x, y;
 void set (int x)
 {
 this.x = x;
 this.y = x + 100;
 }
 void show ()
}
```



```

 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}

```

**//Main class**

class Demo2

```

{
 public static void main (String []s)
 {
 Test t1 = new Test ();
 t1.set (20);
 t1.show ();
 }
}

```

```
E:\JAVA\Programs\Core Java Program\Data Shadowing\Test2>javac Test2.java
```

```
E:\JAVA\Programs\Core Java Program\Data Shadowing\Test2>java Demo2
```

```

x = 20
y = 120

```

**[Note: -** The name of static variable and non-static variable can never be the same in the same class.]

class Test

```

{
 static int x, y;
 int x; //error
}

```

## **Static Method**

- Static methods are also known as the class methods.
- Static method can never use the non-static member of the class directly because they never invoked by the object.
- If we get the reference id of any object in the static method then with that object we can use the non-static members.

class Sample

```

{
 int x, y;
 static void m1 ()
 {
 x = 10; //error, x is the non-static variable
 }
}

```

**//main class**

class Demo3

```

{

```

```

 public static void main (String []s)
 {
 Sample s1 = new Sample (); //calling the m1 () method with class name
 }
}

```

**[Note: -** In the above program m1 ( ) method will be invoked by the Sample class name so no reference id of Sample object will be moved in the m1 ( ) method implicitly.]

- That means there is no **this keyword** available in the m1 ( ) method show how we can use the x variable in the m1 ( ) method.

**Conclusion: -** Implicitly no reference id of any object is available in the static method but explicitly we can take the reference id of object in any static method.

- There are following mainly 2ways used
  1. By passing the object reference id in the argument of static method.
  2. By using the new keyword in the static method.

#### **1<sup>st</sup> way**

```

class Demo4
{
 public static void main (String []s)
 {
 Sample s1 = new Sample ();
 Sample.m1 (s1);
 s1.show();
 }
}

```

#### **// Changing in the Sample class**

```

class Sample
{
 int x, y;
 static void m1 (Sample s)
 {
 x = 10; //error, x is the non-static variable
 s.x = 10; //error
 s.y = 20; //error
 }
 void show ()
 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}

```

```

C:\Windows\System32\cmd.exe

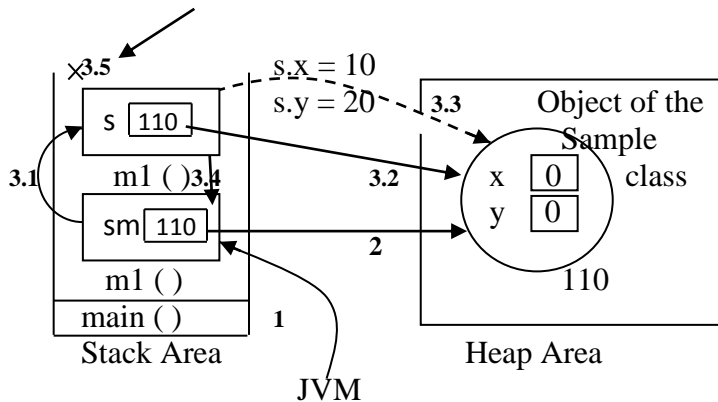
E:\JAVA\Programs\Core Java Program\StaticMethod>javac Sample.java

E:\JAVA\Programs\Core Java Program\StaticMethod>javac Demo.java

E:\JAVA\Programs\Core Java Program\StaticMethod>java Demo
x = 10
y = 20

E:\JAVA\Programs\Core Java Program\StaticMethod>

```



1. JVM invokes the `main()` method.
2. From the `main()` method object of the `Sample` class created in the heap memory and referenced by the local variable `sm` from the `main()` method.
- 3.1 `m1()` method invoked and reference is passed in the argument.
- 3.2 Reference id copied into `s` variable.
- 3.3 Value in the object property is stored.
- 3.4 `m1()` method finished and control is return.
- 3.5 Frame of `m1()` method destroyed and local variable also destroyed.

## // Second way

```

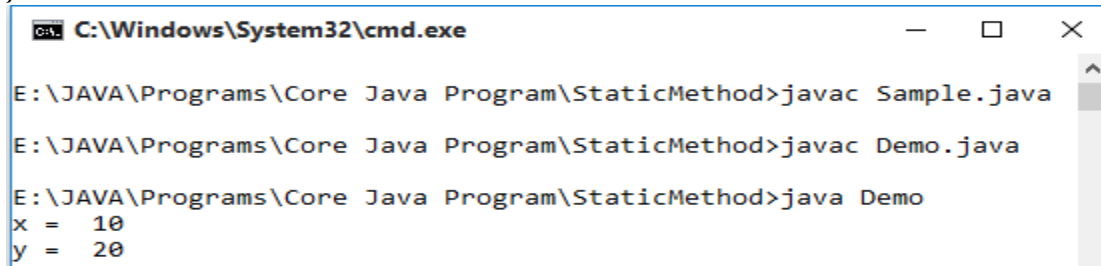
class Demo5
{
 public static void main (String []s)
 {
 Sample.m1 ();
 }
}
class Sample
{
 int x, y;
 static void m1 ()
 {
 Sample s1 = new Sample ();
 s1.x = 10;
 s1.y = 20;
 }
}

```

```

 s1.show();
 }
 void show ()
 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\StaticMethod>javac Sample.java
E:\JAVA\Programs\Core Java Program\StaticMethod>javac Demo.java
E:\JAVA\Programs\Core Java Program\StaticMethod>java Demo
x = 10
y = 20

```

### **Combining the main class and service class together: -**

- Main class is the class that contains the main ( ) method.
- Service class is the class that contains that non-static behavior for the different service.

```

class Test
{
 int x, y;
 void set (int val)
 {
 x = val;
 y = val + 100;
 }
 void show ()
 {
 System.out.println (" x = " + x);
 System.out.println (" y = " + y);
 }
 public static void main (String []s)
 {
 x = 10; // error
 set (10); // error
 // JVM always invoked to the main () with class name. Test.main ()
 Test t = new Test ();
 t.set (10);
 t.show ();
 t.x = 100;
 }
}

```

- We can never use this keyword in the static method.

**Example: - Sample for practice**

**//Product.java**

```

class Product
{
 int qty, price;
 static int collection;
 static void setCollections (int val)
 {
 collection = val;
 }
 void setDetails (int q,int p)
 {
 qty = q;
 price = p;
 }
 void doSales (int unit)
 {
 if (qty >= unit)
 {
 qty -= unit;
 collection = collection + price * unit;
 System.out.println (unit + "Sold sufficient");
 return;
 }
 System.out.println ("Insufficent Quantity");
 }
 void doPurchase (int unit)
 {
 qty += unit;
 int costprice = price - (price * 10/100);
 collection = collection - costprice * unit;
 System.out.println ("Unit purchased");
 }
 void report ()
 {
 System.out.println ("\nQuantity " + qty);
 System.out.println ("price " + price);
 }
 public static void main (String []s)
 {
 collection = 10000;
 Product p1, p2;
 p1 = new Product ();
 p2 = new Product ();
 p1.setDetails (50, 1000);
 p2.setDetails (60, 300);
 p1.doSales (30);
 p2.doSales (20);
 p1.doPurchase (50);
 }
}

```

```

 p1.report ();
 p2.report ();
 System.out.println ("\nTotal collection " + collection);
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program>javac Product.java
E:\JAVA\Programs\Core Java Program>java Product
30Sold sufficient
20Sold sufficient
Unit purchased
Quantity 70
price 1000
Quantity 40
price 300
Total collection 1000
E:\JAVA\Programs\Core Java Program>

```

## **Types of variable in java**

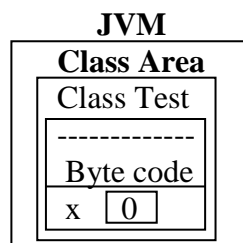
1. Static variable
2. Instance variable
3. Local variable

These three types specify the location, creation time and number of copies of any variable.

### **1. Static variable**

- ❖ Static variable always be created at the class level.
- ❖ Static variable always gets the memory in the class area within the JVM.
- ❖ There is only one copy of static variable is created in the memory.

[Note: - remaining point we will already discuss.]



### **2. Instance variable**

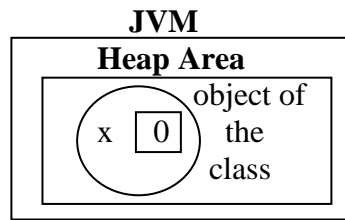
➤ class

```

{
 int y;
 other method
}

```

- y is the instance variable.
- Instance variable also declare at the class level.
- Instance variable always be created within the heap area when the object of the class is created.



- There are multiple copies of the instance variable created in the heap area.

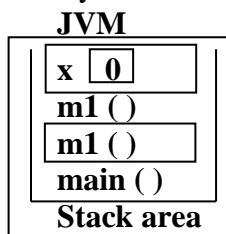
### 3. Local Variable

- Variables declared within any method, constructor or blocks are known as the local variable.

- class Test

```
{
 void m1 ()
 {
 int x;
 }
}
```

- Local variable always gets the memory in the stack area within the frame of the method.



- There is only one copy of the local variable exist at any particular time in the stack area, when the multithreading and recursive method calling is not used.

### Differences between the Local, Stack and Instance Variable

| Base                 | Local                                                | Instance                                                                       | Static                                                           |
|----------------------|------------------------------------------------------|--------------------------------------------------------------------------------|------------------------------------------------------------------|
| Memory area          | Stack area                                           | Heap area                                                                      | Class area                                                       |
| Creation time        | When method is executing.                            | When object is creating.                                                       | When class loading.                                              |
| How to access        | Directly within same method.                         | Always with object reference id via <b>this keyword</b> or reference variable. | Always accessed by the class name or directly in the same class. |
| Use of the specifier | Never                                                | Can be used                                                                    | Can be used                                                      |
| No of copies         | Only one at time, if no recursion & multithreading.  | Multiple copies                                                                | Always only one copy is created.                                 |
| Default value        | No default value, have to be initialized before use. | Always have default value.                                                     | Always have default value.                                       |

[**Note:** - In any scope the local variable can be created and destroyed again and again.]

```
class Test
{
 void m1 ()
 {
 int x;
 for (int i = 1; i <= 10; i++)
 {
 int y = i + 10;
 }
 //After finish the loop i & y will be destroyed
 int x;
 int y;
 }
}
```

### **Object as a method argument**

- In java we can pass any object in the argument of any method but actually the reference id of the object is passed not the actual object.

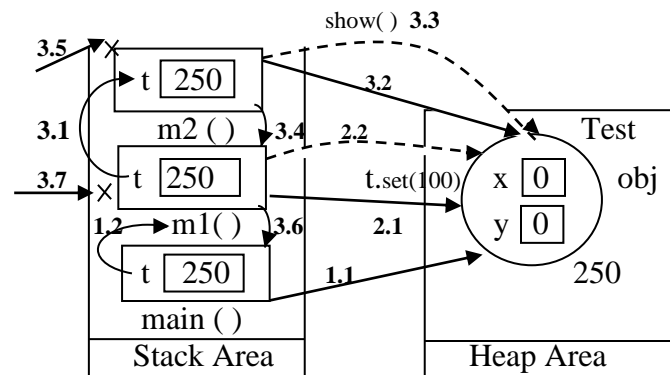
```
class Test
{
 int x, y;
 void set (int val)
 {
 x = val;
 y = x + val;
 }
 void show ()
 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}
```



// main class

class Demo

```
{
 static void m1 (Test t)
 {
 t.set (100);
 System.out.println ("m1 is invoked");
 m2 (t2); //Calling the m1 ()
 }
 static void m2 (Test t)
 {
 t.show ();
 }
 public static void main (String []s)
 {
 Test t; //local reference variable
 t = new Test ();
 m1 (t);
 }
}
```



**2.2** Set method invoked with the reference id of the object. (Frame will also be created and in the stack and destroy after the execution of set ( ) method)

**Example: -**

class Test

{

int x, y;

void set (int val)

{

x = val;

y = x + val;

}

void show ( )

{

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

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

```

 }
}

// main class
class Sample
{
 int a, b;
 static void setValue (int val)
 {
 a = val;
 b = val + 100;
 }
 void m1 (Test t)
 {
 t.set (a + b);
 }
 void show ()
 {
 System.out.println ("a = " + a);
 System.out.println ("b = " + b);
 //t.show ();
 //t.set (50);
 }
 public static void main (String []s)
 {
 Test t = new Test ();
 Sample sm = new Sample ();
 sm.setValue (400);
 sm.m1 (t);
 sm.show (t);
 t.show ();
 }
}

E:\JAVA\Programs\Core Java Program\Object's as an method argument\Test1>javac Sample.java
E:\JAVA\Programs\Core Java Program\Object's as an method argument\Test1>java Sample
a = 400
b = 500
x = 900
y = 1800

```

## **Method Returning the Object**

- Any method can return the object but actually the reference id of the object is return not actual object.

```

class Test
{
 int x, y;
 void set (int val)
 {

```

```

 x = val;
 y = x + val;
 }
 void show ()
 {
 System.out.println ("x = " + x);
 System.out.println ("y = " + y);
 }
}
class Sample
{

```

```

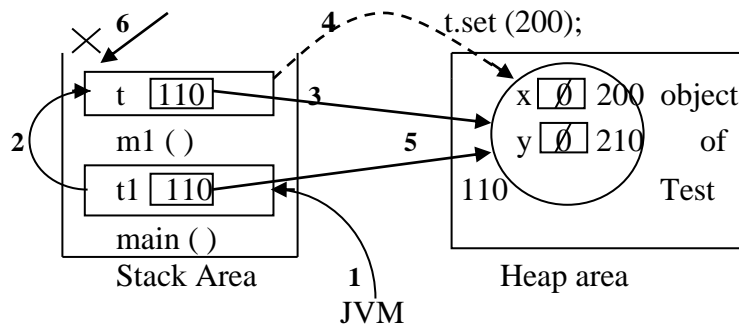
 Static Test m1()
 {
 Test t = new Test ();
 2 ↓
 t.set (200);
 return (t);
 }
 public static void main (String []s)
 {
 Test t1;
 t1 = m1 ();
 t1.show ();
 }
}

```

```

E:\JAVA\Programs\Core Java Program\Method_running_the_object>java Sample
X is: 200
Y is: 210

```



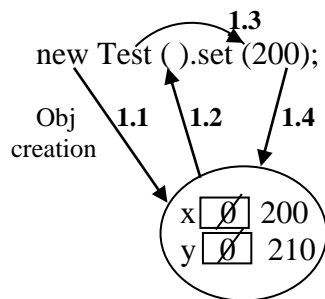
1. JVM invoked to main () method.
  2. m1 () method is invoked by the main.
  3. Object of Test class created from m1 () in the heap memory.
  4. Reference id of the object is return.
  5. Reference id is copy in the t1 ref variable in main () method. So object of test referenced from main () method.
  6. Frame of m1 () method destroyed and t variable also gets destroyed.
- If any method has the class name in their return type that method must return reference id of the object of that class or returns the null.

## Anonymous Object

- Anonymous object are known as the use and throw object that means from the object is created at that case the object can't be use further that means without making reference variable only single time we can use anonymous object.

```
class Test
{
 int x, y;
 void set(int val)
 {
 x = val;
 y = val + 10;
 }
 void show()
 {
 System.out.println("X is: "+x);
 System.out.println("Y is: "+y);
 }
}

class Demo
{
 public static void main (string []s)
 {
 new Test ().set (200);
 new Test().show(); // It is new object
 }
}
```



3.5. New object created.

3.6. Reference id of object return.

3.7. Through reference if the set ( ) method is invoked.

3.8. From set ( ) method objects properties gets modified.

[Note: - In main ( ) method in the next line the test object can't be used.]

[Note: - After invocation of set method the test object become unreferenced.]

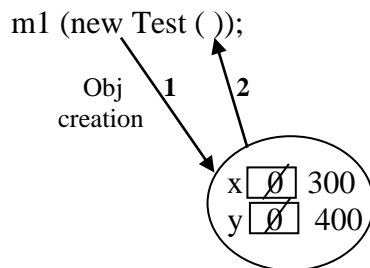
## Passing the object reference id the argument anonymously.

```
class Test
{
 int x, y;
```

```

void set(int val)
{
 x = val;
 y = val + 100;
}
void show()
{
 System.out.println("X is: "+x);
 System.out.println("Y is: "+y);
}
}
class Demo1
{
 static void m1 (Test t)
 {
 t.set (300);
 t.show ();
 }
 public static void main(String []s)
 {
 m1(new Test ());
 }
}
E:\JAVA\Programs\Core Java Program\Anonymous_Object>javac Demo1.java
E:\JAVA\Programs\Core Java Program\Anonymous_Object>java Demo1
X is: 300
Y is: 400

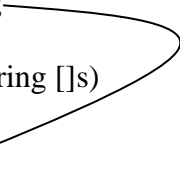
```



1. Test object created.
2. Reference id returns and placed in the braces of `m1 ( )` method.
3. Reference id of test object is placed in the argument.

## Returning the reference id of object anonymously from any method

```
class Demo
{
 static Test m1 ()
 {
 return (new Test ());
 }
 public static void main (string []s)
 {
 Test t;
 t1 = m1 (); //Reference id return to main ()
 }
}
```



## Chain of the method calling: -

- In case of non-static method there can be the chain of method calling that means in one statement more than one method can be invoked.

```
class Demo
{
 static void m1(Test t)
 {
 t.set(200);
 t.show();
 Or // new Test ().set (200).show ();
 Anonymously
 }
}
```

- In case chain of method calling the method must return the reference id of the object so that with the returned reference id the next method could be invoked.
- In the above example of method calling chain the set method must return the reference id of Test object either same test object via **this keyword** or the reference id of any new test object.

## **Change in the Test class**

```
class Test
{
 int x, y;
 Test set (int val)
 {
 this.x = val;
 this.y = val + 100;
 return (this);
 }
 void show ()
 {
```

```

 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
}
class Demo
{
 static Test m1 ()
 {
 return (new Test());
 }
 public static void main (String []s)
 {
 Test t = new Test ();
 t.set (200).show();
 //or
 // new Test ().set (200).show ();
 }
}
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Test>javac Test.java
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Test>java Demo
x = 200
y = 210

```

**[Note: -** The catching variable on the left hand side depends upon the return type of last method invoked in the class.]

```
// Test t = new Test ().set (200).show (); // compiler Error
```

- Return type of show ( ) method is void so catching variable can't be place on the left hand side if show ( ) method returns there reference id of current object than catching variable can be placed.

```

class Test1
{
 int x, y;
 Test set(int val)
 {
 this.x = val;
 this.y = val + 10;
 return(this);
 }
 void show()
 {
 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
}
class Demo1

```

```

{
 public static void main (String []s)
 {
 Test t = new Test ();
 new Test();
 t.set (200).show();
 }
}
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Test>javac Demo.java
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Test>java Demo
x = 200
y = 210

```

- Now we can place catching variable it is not necessary/mandatory to catch the value return by any method it is.

```

class Demo
{
 int x, y;
 static Demo m1 (int val)
 {
 return (new Demo());
 }
 Demo set (int val)
 {
 x = val;
 Demo d = new Demo ();
 d.y = val+100;
 return (d);
 }
 void show ()
 {
 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
}
class Sample
{
 public static void main (String []args)
 {
 Demo d1, d2;
 d1 = Demo.m1 (100);
 d2 = d1.set (1000);
 d1.show ();
 d2.show ();
 }
}

```



```
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Demo>javac Demo.java
E:\JAVA\Programs\Core Java Program\Anonymous_Object\Demo>java Sample
x = 1000
y = 0
x = 0
y = 1100
```

## Constructor

- ✚ Constructors are the special type of non-static method in any class that are automatically invoked when the object of the class is created.
- ✚ The purpose of the constructor is to initialize the object.
- ✚ Object initialization means to assign the values in the object properties according to the requirement either from the end user or generated in the program.

### Characteristic of the constructor: -

1. The name of the constructor always be same as the class name.

```
class Test
{
 Test ()
 {

 }
}
```

2. Constructor can never have any return type.
3. Within the constructor, **this keyword** can be used.
4. Constructors are the non-static member of the class.
5. With the constructors access specifiers (private / public / protected) can be used.
6. In the constructor static members of the same class can be accessed directly.

### **Example: -**

```
class Test
{
 int x, y;
 Test()
 {
 x = 10;
 y = 20;
 System.out.println("Object created & initialize");
 }
 void show()
 {
 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
 public static void main(String []s)
 {
```

```

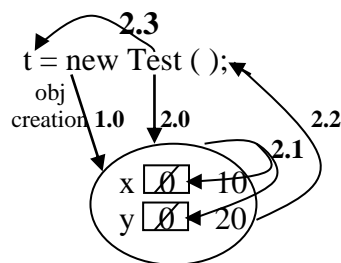
 Test t;
 System.out.println(" welcome");
 t = new Test();
 t.show();
 }
}

E:\JAVA\Programs\Core Java Program\Constructor>javac ConstructorTest.java

E:\JAVA\Programs\Core Java Program\Constructor>java Test
welcome
Object created & initialize
x = 10
y = 20

```

[Note: -In the following way the object is created and constructor is invoked.]



- 1.0. New object of Test created.
- 2.0. Constructor is invoked.
- 2.1. From the constructor object properly are initialized.
- 2.2. Reference id of object return.
- 2.3. Reference id is copied into the `t`.

## **Type of Constructor**

1. Default constructor
2. Parameterize constructor

### **1. Default constructor**

- The no argument constructor is known as the default constructor that means the default constructor.
- Example in test class there was the default constructor.

```

class Test
{
 int x, y;
 Test()
 {
 x = 10;
 y = 20;
 System.out.println("Object created & initialize");
 }
 void show()
 {

```

```

 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
 public static void main(String []s)
 {
 Test t;
 System.out.println(" welcome");
 t = new Test();
 t.show();
 }
}
E:\JAVA\Programs\Core Java Program\Constructor>javac ConstructorTest.java
E:\JAVA\Programs\Core Java Program\Constructor>java Test
welcome
Object created & initialize
x = 10
y = 20

```

## 2. Parameterize constructor

- Constructor with the argument list is known as the parameterize constructor.

```

class Test
{
 int x, y;
 Test (int val)
 {
 x = val;
 y = val + 100;
 }
 void show ()
 {
 System.out.println("x = " + x);
 System.out.println("y = " + y);
 }
 public static void main(String []args)
 {
 Test t;
 t = new Test (50); // calling of parameterize constructor
 t.show();
 }
}
Parameterize Constructor
E:\JAVA\Programs\Core Java Program\Constructor\Parameterized>javac ParameterizedTest.java
E:\JAVA\Programs\Core Java Program\Constructor\Parameterized>java Test
x = 50
y = 150

```

**[Note: -** If we doesn't create any constructor in the class then compiler a default constructor in the class within the byte code.]

[**Note:** - If we create any constructor in the class then compiler doesn't provide the default constructor in the class.]

### **Constructor Overloading**

- In any class there can be more than one constructor argument list.
- The argument list can be default either by changing the number of arguments or by changing the type of arguments.

class Area

```
{
 int x, y, z, res, type;
 Area (int x)
 {
 this.x = x;
 type = 1;
 }
 Area (int x, int y)
 {
 this.x = x;
 this.y = y;
 type = 2;
 }
 Area (int x, int y, int z)
 {
 this.x = x;
 this.y = y;
 this.z = z;
 type = 3;
 }
 void calculation ()
 {
 if (type == 1)
 {
 res = x*x;
 }
 if (type == 2)
 {
 res = x*y;
 }
 if (type == 3)
 {
 int s = (x+y+z)/2;
 res = (int)(math.sqrt(s*(s-x)*(s-y)*(s-z)));
 }
 }
 void show ()
 {
```

```

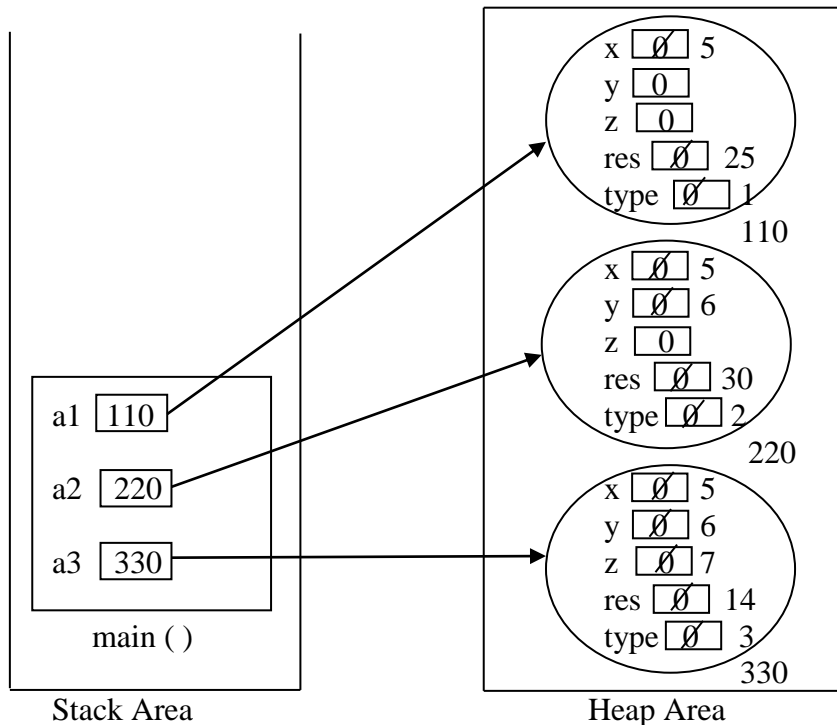
 if (type =1)
 System.out.println ("Area of square: " + res);
 if (type = 2)
 System.out.println ("Area of Rect: " + res);
 if (type = 3)
 System.out.println ("Area of Triangle: " + res);
 }
 public static void main (String []args)
 {
 Area a1, a2, a3;
 a1 = new Area (5);
 a2 = new Area (5, 6);
 a3 = new Area (5, 6, 7);
 a1.calculation ();
 a2.calculation ();
 a3.calculation ();
 a1.show ();
 a2.show ();
 a3.show ();
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Constructor\Constructor_Overloading>javac Area.java
E:\JAVA\Programs\Core Java Program\Constructor\Constructor_Overloading>java Area
Area of sqare is: 25
Area of Rectangle is: 30
Area of Triangle is: 14

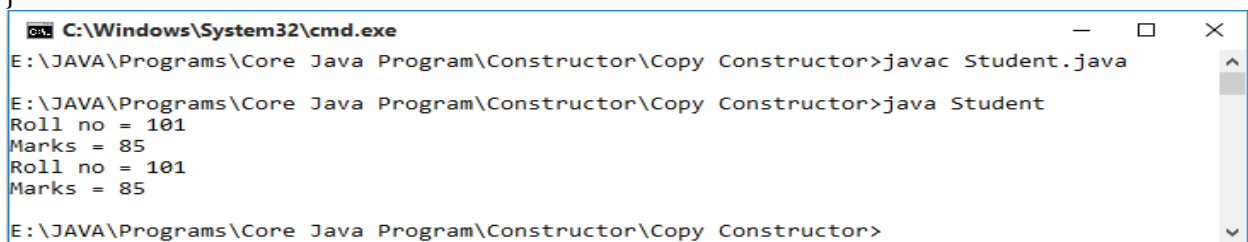
```



## Copy Constructor

- A constructor that has the reference of the object of same class is known as the copy constructor.

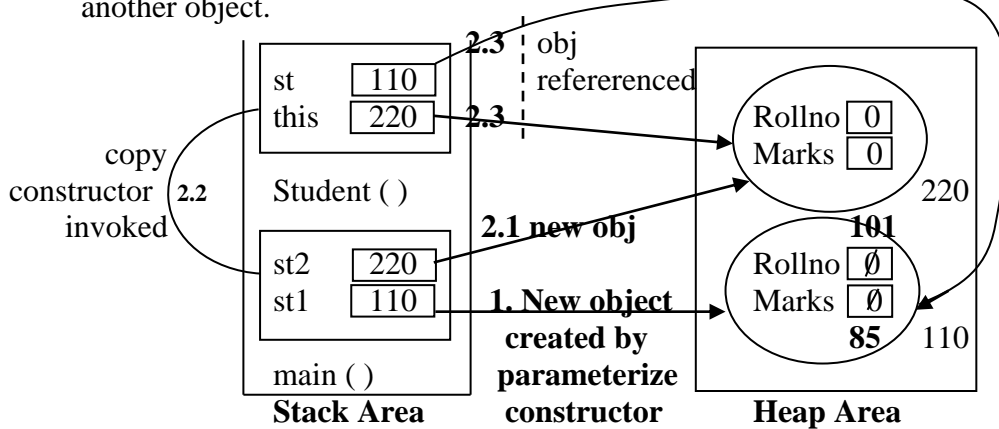
```
class Student
{
 int rollno, marks;
 Student (int rollno, int marks)
 {
 this.rollno = rollno;
 this.marks = marks;
 }
 Student (Student st)
 {
 this.rollno = st.rollno;
 this.marks = st.marks;
 }
 void show ()
 {
 System.out.println ("Roll no = " + rollno);
 System.out.println ("Marks = " + marks);
 }
 public static void main (String []args)
 {
 Student st1, st2;
 st1 = new Student (101, 85);
 st2 = new Student (st1);
 st1.show ();
 st2.show ();
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Constructor\Copy Constructor>javac Student.java
E:\JAVA\Programs\Core Java Program\Constructor\Copy Constructor>java Student
Roll no = 101
Marks = 85
Roll no = 101
Marks = 85
E:\JAVA\Programs\Core Java Program\Constructor\Copy Constructor>
```

[Note: - copy constructors are the parameterize constructor.]

- Purpose of the copy constructor is to copy the property value of the one object into another object.



[**Note:** - For the constructor invocation in the stack also a new frame is created for the local variable and as the constructor created frame gets destroyed.]

**Question: - What happen if we write the return type in the constructor?**

**Answer: -**In that case the constructors become the normal method and we have to invoke it explicitly that means in any class there can be the method with the same name.

class Test

```
{
 int x;
 void Test (int val)
 {
 x = val;
 }
 void show ()
 {
 System.out.println ("x = " + x);
 }
 public static void main (String []args)
 {
 Test t = new Test ();
 t.Test (50); //explicit calling of method
 t.show ();
 }
}
```

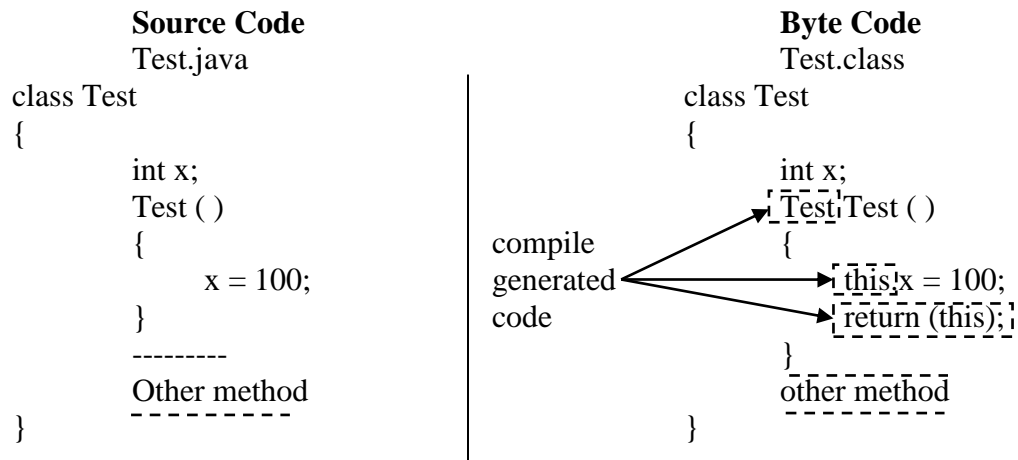
1    /\* This is not the constructor but a normal non-static method.\*/

```
E:\JAVA\Programs\Core Java Program\Constructor>javac Test.java
E:\JAVA\Programs\Core Java Program\Constructor>java Test
x = 50
E:\JAVA\Programs\Core Java Program\Constructor>
```

**Question: - Why constructor never any return type?**

**Answer: -** Constructor always have the implicit return type that means they never have the programmer defined return type but in the byte code compiler provide the return type of the constructor.

Compiler provides the return type same as the class name through which constructor returns the reference id of current object. That means constructor always return the reference id of the current object.



### Initialization Block (init Block)

- **init blocks** are the blocks in the class that are always executed when the object that class is created.

```

class Test
{
 int x, y;
 {
 System.out.println ("init block");
 x = 10;
 }

 Other constructor and method

}

```

- There can be any number of **init block** and all are executed in the sequence in which they are created in the class.

```

class Test2
{
 int x;
 {
 System.out.println ("init block");
 x = 10;
 }
 void show ()
 {
 System.out.println ("x = " + x);
 }
 {
 System.out.println ("Second init block");
 }
}

```



```

public static void main (String []args)
{
 Test2 t;
 System.out.println ("Welcome");
 t = new Test2 ();
 t.show ();
}
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Init_Block>javac Test2.java

E:\JAVA\Programs\Core Java Program\Init_Block>java Test2
Welcome
init block
Second init block
x = 10

```

[**Note:** - The concept of constructors also remains same along with the concept of **init block**.]

- If there is the **init block** defined by the programmer and no contractor defined also in that case the default constructor provided by the compiler.
- At the time of object creation 1<sup>st</sup> init block executed then constructor invoked.
- In the **init block** we can write any valid programming statement executed returning value.

### **Purpose of init block**

- In case of constructor there can be only one constructor invoked as the time of object creation selected on the basic of number of arguments.
- If there are multiple constructors then any common code we have to write in all the constructors.
- **init block** are not selective means when the object is created all the init blocks are executed.
- The reassume to behind to create the multiple init block is use to make the class readable means different task can be coded in the different init block.
- Unlike the constructors if we don't write any init block then compiler will not provide in any init block.

### **Internal point**

- After the compiler compile in-lined the code of all init block in the started in each constructor.
- That means the purpose of init block is to manage the source code.

### **static Block**

- **static blocks** always be created in the class by using the static keyword.
- All the **static blocks** always be executed at the time of loading of the class.

```

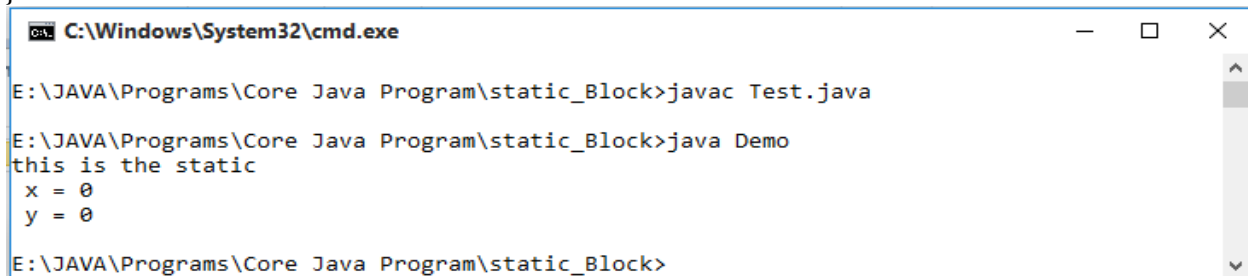
class Test
{

```

```

int x;
static int y;
static
{
 System.out.println("this is the static");
}
void show ()
{
 System.out.println (" x = " + x);
 System.out.println (" y = " + y);
}
}
class Demo
{
 public static void main (String []s)
 {
 Test t;
 t = new Test();
 // first object creation so test class gets loaded
 t.show();
 }
}

```



```

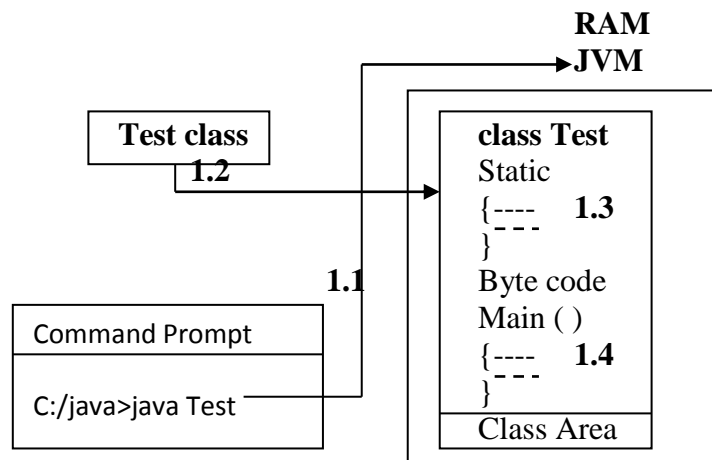
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\static_Block>javac Test.java
E:\JAVA\Programs\Core Java Program\static_Block>java Demo
this is the static
x = 0
y = 0
E:\JAVA\Programs\Core Java Program\static_Block>

```

- Accept returning the value we can write any valid programming statement in the static block.
- There can be any number of static blocks in any class at the time of class loading all the static blocks are executed in the sequence in which they are declared.

**Question: -** If the static in the main class then when it will be executed?

**Answer: -** As we write the java command JVM immediately invokes the main( ) method with the class name at method the main( ) class will be loaded and there static executed.



**1.1** JVM started and loaded in the RAM and call the main ( ) method.

**1.2** Main ( ) is the 1<sup>st</sup> static member to be invoked with the test class so the class is loaded.

**1.3** Just after the class loading static block executed.

**1.4** Execution of main ( ) method started.

[**Note:** - Static block is the block always be executed before the main ( ).]

**//Student.java**

```
class Student
{
 int rollno, marks;
 Student(int rollno, int marks)
 {
 this.rollno = rollno;
 this.marks = marks;
 System.out.println("Student object created");
 }
 static
 {
 System.out.println("Student class is loaded");
 }
 void show()
 {
 System.out.println("\n Roll no = " + rollno);
 System.out.println("Marks = " + marks);
 }
}
```

**//School.java**

```
class School
{
 static
 {
 System.out.println("School class loaded");
 }
}
```

```

public static void main(String s[])
{
 Student st1, st2;
 System.out.println("welcome in main");
 st1 = new Student(101,65);
 //class will loaded then object created
 st2 = new Student (102, 55);
 //onle object created
 st1.show();
 st2.show();
}
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\static_Block\School>javac Student.java

E:\JAVA\Programs\Core Java Program\static_Block\School>javac School.java

E:\JAVA\Programs\Core Java Program\static_Block\School>java School
School class loaded
welcome in main
Student class is loaded
Student object created
Student object created

 Roll no = 101
Marks = 65

 Roll no = 102
Marks = 55

E:\JAVA\Programs\Core Java Program\static_Block\School>

```

[Note: - Up to the jdk1.6 when we write the java command to run the main ( ) class than JVM 1<sup>st</sup> loaded the class without checking that there is main ( ) method exist or not. In that case the static block executed and after that main ( ) method not found error generated at the run time.]

### **Purpose of static block is to initialize the static variable.**

- At the time of class loading 1<sup>st</sup> of all static variables are created then just after static block executed without any time gap.

```

class Test
{
 static int x;
 static
 //static void m1()
 {
 int val = (int)(Math.random()*100);
 x = val * 10;
 }
 void show()
 {
 if(x % 2 == 0)

```

```

 System.out.println("Even");
 else
 System.out.println("Odd");
 }
 public static void main(String s[])
 {
 Test t;
 System.out.println("Welcome");
 t = new Test();
 t.show();
 }
}
E:\JAVA\Programs\Core Java Program\static_Block>javac Test1.java
E:\JAVA\Programs\Core Java Program\static_Block>java Test
Welcome
Odd

```

[**Note:** - At the time of class loading x variable will be created but no static( ) method will be invoked implicitly.]

- Before the m1; m2( ) method is invoked then value of the x will be found as 0(zero).

## **Access Specifiers**

- In the OOPS based technologies there are the access specifier that specifies is the scope of the accessibility of any class and there members.

## **In java there are 4 types of specifires.**

1. **Private:** -Only within the same class.
2. **Default:** - Within any class of same package.
3. **Protected:** - Within any class of same package and within child class of other package.
4. **Public:** - Within any class of same package and other.

[**Note:** - **Public**, **Protected** and **Private** are the keywords and where none of these are present the default specifier will work.]

- In java not any class, method & variable can be accessed without accessed specifire.
- Till now except the main( ) all the class, method & variable have default access specifires.
- Any class in java can have only the **public** and **default** specifire.
- Any class member (variable or method can have any access specifire)

e. g

```

public class Test //This is the public class
{

}

```

- In any **.java** file there can be only one public class.
- The **.java** file must be saved with the name of the public class.
- Without creating the package we can use only **private** and **default** access specifier.
- Creating the private member in the class.

```
class Test
{
 private int x;
 Test()
 {
 System.out.println("Default constructor");
 x = 100;
 }
 private Test(int x)
 {
 this.x = x;
 }
 private void display()
 {
 System.out.println(" value of x = " + x);
 }
 void show()
 {
 System.out.println(" x = " + x);
 }
}
class Demo
{
 public static void main(String s[])
 {
 Test t1, t2;
 t1 = new Test();
 //t1.x = 100; //error
 //t1.display; //error
 t1.show();
 //t2 = new Test(55); //error
 }
}
```

**Question: -** Can we create only the private constructor in any class.

**Answer: -** Yes, we can create but in that case the object of that can't be created outside the class. In that case the through the static method of that same class we can create and return the object of that class.

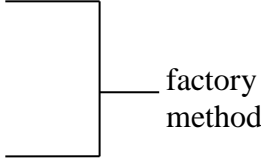
e.g

```
class Test
{
 private int x;
 private Test(int val)
```

```

 {
 this.x = x;
 }
 void show()
 {
 System.out.println(" x = " + x);
 }
 static Test getobject(int val)
 {
 Test t = new Test(val);
 return(t);
 }
}
class Demo
{
 public static void main(String []s)
 {
 Test t1, t2;
 t1 = Test.getobject(80);
 t2 = Test.getobject(65);
 t1.show();
 t2.show();
 }
}

```



- getobject( ) method is consider as the factory method that means the factory of the object of Test class.

## **Singleton Pattern**

- This is the enhanced version of the factory pattern according to this pattern only one object of any class can be created at most.

```

class Test
{
 private static Test t; //static Test t;
 private int x;
 {
 this.x = x;
 }
 void show()
 {
 System.out.print("x = ");
 }
 static Test getobject(int val)
 {
 if(t == null)
 t = new Test(val);
 return(t);
 }
}

```

```

}
class Demo
{
 public static void main(String []s)
 {
 Test t1, t2;
 t1 = Test.getobject(50);
 //this.t = null; // this.t=null; this constant is break the singleton pattern
 t2 = Test.getobject(56);
 t1.show();
 t2.show();
 }
}

```

## **Encapsulation**

- Encapsulation is the process through which the properties of the object wrapped by the behavior of the object that means properties of object should always be affected by the behavior of that object not directly.

```

Student st = new Student();
st.marks = 85; //breaking the encapsulation
st.exam(85); //following the encapsulation

```

- By making non-static properties private we can enforce the encapsulation.

## **Abstraction (Hiding)**

- Hiding the complicity and open up the essential thing are known as the abstraction.
- In the abstraction what the output will come we know but how that output will be generated we don't know.
- The 1<sup>st</sup> level abstraction we achieve through the method calling where we know about the output of that method but how that output will be generated we don't know.
- In java higher form of abstraction we achieve through the abstraction classes and interfaces.

## **Association**

- Association means the linking of one thing with another thing.
- In java linking the reference of the object with different parts of the program this is known as the association.
- By storing the reference id of the object in the reference variable we achieve the association.

## **The association in java can be of three types.**

- i). Association of the object with the method.
- ii). Association of the object with the other object.
- iii). Association of the object with the class.

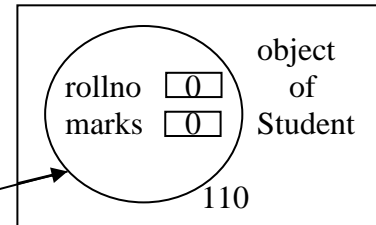


### i) Association of the object with the method

- It can be achieved by making the reference variable as the local in any method.

- E. g

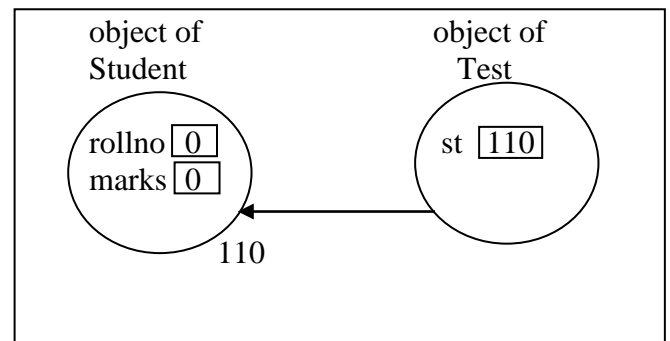
```
class Test
{
 public static void main(String []s)
 {
 Student st;
 st = new Student();
 }
}
```



### ii) Association of the object with the other object.

- By making the reference variable non-static one object can be associated with the other object.

```
class Test
{
 Student st;
 public static void main(String []s)
 {
 Test t = new Test();
 t.set();
 }
 void set()
 {
 st = new Student();
 }
}
class Test
{
 Student st = new Student();
 public static void main(String []s)
 {
 Test t = new Test();
 }
}
```



- In java any object never is contained by other object rather the reference id of object contained by other object.
- In java any object can never be the local or non-static or static rather the reference variables are the static or non-static or local.

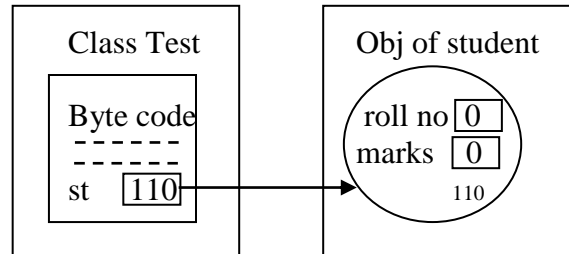
### iii) Association of the object with the class.

- By making the reference variable static we can associate the object with any class.

```

class Test
{
 static Student st;
 public static void main(String s[])
 {
 st = new Student();
 }
}

```



- Association is also as the containership that means id of the objet contained by method.
- Id of object contained by another object.
- Id of object contained by any class.

- When one object contained the reference id of another object then it is called has a relationship between two object.

**For example** – mobile has a battery, student has a book and car has an engine.

**According to the OOAD the association can be further divided in two categories.**

- i) **Aggregation:** - Optional association.
- ii) **Composition:** - Mandatory association

Car has the stereo (Aggregation)  
 Car has the engine (Composition)

- Composition says without the part the whole cannot be existing
- Aggregation says without part the whole can be existing

### **E. g of Aggregation**

```

class Test
{
 Student st;
 void set(int x)
 {
 if(x >= 100 &&x <= 999)
 {
 st = new Student(x);
 }
 }
 public static void main(String []s)
 {
 Test t;
 t = new Test();
 int val = Integer.parseInt([0]s);
 t.set(val);
 }
}

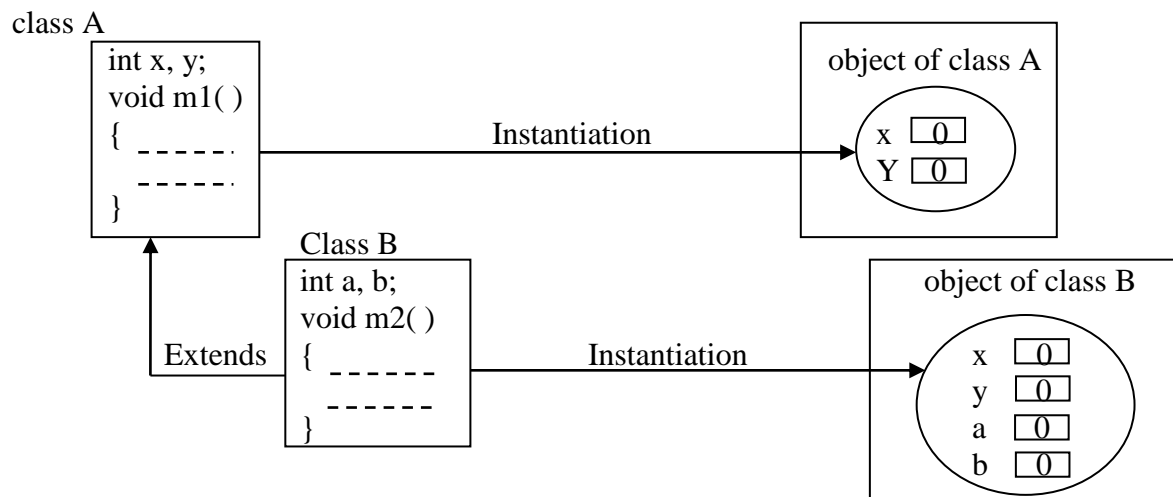
```

## E. g of Composition

```
class Test
{
 Student st = new Student(101);
 void set(int x)
 {
 if(x >= 100 && x <= 999)
 {
 st = new Student(x);
 }
 }
 public static void main(String []s)
 {
 Test t;
 t = new Test();
 }
}
```

## Inheritance

- Inheritance is used to achieve the reusability in java that means decide to developed the each and everything is new it is better to reuse the already developed things.
- This feature of inheritance reduces the effort to generate the common code again and again.
- The real world example is the java programming language itself that means the java is inherited from the C and C++ all the syntax of java is derived from C and C++.
- The environment of inheritance in any application build up between the classes that means one class can reuse the feature of another class.
- **extends** keyword is used to achieve the reusability.



- In the object of B class copy of x and y variable also will be created that shows the reusability of x and y variable.
- In the same way A object of B class we can also invoke to the m1( ) method.

```

class A
{
 int x, y;
 void m1(int val)
 {
 x = val;
 y = val + 100;
 System.out.println(" m1 Invoked");
 }
}
class B extends A
{
 int a, b;
 void m2(int val)
 {
 a = val;
 b = val + 100;
 }
 void show()
 {
 System.out.println(" x = " + x);
 System.out.println(" y = " + y);
 System.out.println(" a = " + a);
 System.out.println(" b = " + b);
 }
}
class Test
{
 public static void main(String []S)
 {
 B b = new B();
 b.m1(50);
 b.m2(40);
 b.show();
 }
}
E:\JAVA\Programs\Core Java Program\Inheritance>javac Inheritance.java
E:\JAVA\Programs\Core Java Program\Inheritance>java Test
m1 Invoked
x = 50
y = 150
a = 40
b = 140

```

- Member of the parent class can be used in the child class or with object of child class.
- Private member of the parent class can never be accessed in the child class or with the object of child class.

Class A [parent/super class]

Class B [child/sub class]



```
class A
{
 private int x, y;
 void m1(int val)
 {
 x = val;
 y = val + 100;
 System.out.println(" m1 invoked ");
 }
 int getX()
 {
 return(x);
 }
 int getY()
 {
 return(y);
 }
}
class B extends A
{
 int a,b;
 void m2(int val)
 {
 a = val;
 b = val + 100;
 }
 void show()
 {
 //System.out.println(" x = " + x); //error
 //System.out.println(" x = " + x); //error
 System.out.println(" x = " + getX());
 System.out.println(" y = " + getY());
 System.out.println(" a = " + a);
 System.out.println(" b = " + b);
 }
}
class Test
{
 public static void main(String []s)
 {
 B b = new B();
 b.m1(50);
 b.m2(60);
 b.show();
 }
}
```

```

 }
}
E:\JAVA\Programs\Core Java Program\Inheritance\Test1>javac InheritanceTest.java
E:\JAVA\Programs\Core Java Program\Inheritance\Test1>java Test
m1 invoked
x = 50
y = 150
a = 60
b = 160

```

## **Inheritance and class loading**

- At the time of loading of the child class the parent class have already be loaded if not then just report the child class loading the parent class will also be gets loaded.

```

class A
{
 static
 {
 System.out.println(" A class loaded");
 }
}
class B extends A
{
 static
 {
 System.out.println(" B class loaded");
 }
}
class C extends B
{
 static
 {
 System.out.println(" C class loaded");
 }
}
class LoadingTest
{
 public static void main(String []s)
 {
 C ref;
 System.out.println(" Welcome in main");
 new C();
 }
}

```

```

}
E:\JAVA\Programs\Core Java Program\Inheritance\Test2>javac InheritanceTest.java
E:\JAVA\Programs\Core Java Program\Inheritance\Test2>java LoadingTest
Welcome in main
A class loaded
B class loaded
C class loaded

```

```

class A
{
 static
 {
 System.out.println(" A class loaded");
 }
}
class B extends A
{
 static
 {
 System.out.println(" B class loaded");
 }
}
class C extends B
{
 static
 {
 System.out.println(" C class loaded");
 }
}
class LoadingTest
{
 public static void main(String []s)
 {
 C ref;
 new A();
 System.out.println(" welcome in main");
 new C();
 }
}

```

```

E:\JAVA\Programs\Core Java Program\Inheritance\Test3>javac InheritanceTest.java
E:\JAVA\Programs\Core Java Program\Inheritance\Test3>java LoadingTest
A class loaded
welcome in main
B class loaded
C class loaded

```

## Inheritance of the static member

```
class A
{
 static int x;
 static void m1(int val)
 {
 x = val;
 }
}
class B extends A
{
 static void m2(int val)
 {
 x = x + val;
 }
}
class StaticTest
{
 public static void main(String []s)
 {
 B.m1(100);
 B.m2(500);
 System.out.println (" value of x = " + B.x);
 }
}
E:\JAVA\Programs\Core Java Program\Inheritance\Inheritance_static_member>javac InheritanceTest.java
E:\JAVA\Programs\Core Java Program\Inheritance\Inheritance_static_member>java StaticTest
value of x = 600
```

## Inheritance of the constructor

- Constructor of the parent class never be inherited in the child class.
- From the child class parent constructor can't be invoked after the object creation and concept of inheritance applied on to the object after their creation.

## Construction Changing

- This is the concept always be seen when the object is created in the inheritance that means object of child class is created along with the constructor of child class the constructor of parent class also be invoked.

```
class A
{
 int x;
 A()
 {
 System.out.println(" constructor of A class");
 }
}
```



}



- SS

```

}
E:\JAVA\Programs\Core Java Program\Inheritance\Test2>javac Test.java
E:\JAVA\Programs\Core Java Program\Inheritance\Test2>java B
constructor of A class
constructor of B class

```

[Note: - We can also write `super( );` in any constructor as the 1<sup>st</sup> statement.]

[Note: - **super** can never be placed accept the 1<sup>st</sup> line in the constructor.]

- If parent class doesn't have the default constructor then in the constructor of child class we have to write the `super( )` statement.

```

class A
{
 int x;
 A(int val)
 {
 x = val;
 System.out.println(" constructor of A class");
 }
}
class B extends A
{
 B()
 {
 super(5);
 System.out.println(" constructor of B class");
 }
 public static void main(String []s)
 {
 new B();
 }
}

```

↗ Calling of parameterize constructor of A class.

```

E:\JAVA\Programs\Core Java Program\Inheritance\Test2>javac Test.java
E:\JAVA\Programs\Core Java Program\Inheritance\Test2>java B
constructor of A class
constructor of B class

```

[Note: - If we doesn't write the **super(5);** in the constructor of B then compiler itself will place the **super( )** to invoke the default constructor of A class that will generated the compiler error.]

- In the A class there is no default constructor because there is the programmer defined parameterizes contractor.

## **Constructor chaining in the same class**

Any one constructor of any class can invoke another constructor of same class at the time of one object creation.

**//Test.java**

```
class Test
{
 Test()
 {
 System.out.println(" Default constructor");
 }
 Test(int val)
 {
 this(); //calling of default constructor
 System.out.println(" Parametrized constructor");
 }
 public static void main(String []s)
 {
 new Test(5);
 }
}
```

E:\JAVA\Programs\Core Java Program\Constructor chaining\Test5>javac Test.java

E:\JAVA\Programs\Core Java Program\Constructor chaining\Test5>java Test

Default constructor

Parametrized constructor

**[Note: -** Constructor chaining can't be the cyclic.]

- Cyclic chain means the calling constructor becomes the calling constructor of their calling constructor.
- In case of cyclic constructor chaining compiler automatically write the super.

```

class A
{
 int x;
 A() ← 4
 {
 System.out.println (" Default constructor in A");
 }
 A(int val) ← 3
 {
 this();
 System.out.println (" Parameterized constructor in A");
 }
}
class B extends A
{
 int y;
 B() ← 3
 {
 this(5); //no super statement provided by the compiler
 system.out.println (" Parameterized constructor in B");
 }
 B(int val) ← 1
 {
 super(val);
 system.out.println ("Parameterized constructor in B");
 }
 public static void main(String []s)
 {
 new B();
 }
}
E:\JAVA\Programs\Core Java Program\Constructor chaining\Test6>javac Test.java
E:\JAVA\Programs\Core Java Program\Constructor chaining\Test6>java B
Default constructor in A
Parametrized constructor in A
Parametrized constructor in B
Default constructor in B

```

- If in the any class there is only the private constructor then that class can never be inherited.
- From the child constructor the parent class constructor always be invoked that's way we can't make the child class.

```

class A
{
 private A()
 {

```

```

 System.out.println("Default constructor of A");
 }
}
class B extends A
{
 B()
 {
 super(); //implicit super() to invoke the constructor of A
 System.out.println("constructor of B");
 }
}
class PTest
{
 public static void main(String s[])
 {
 B b = new B();
 }
}

```

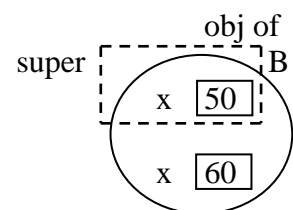
## **Data hiding**

- When the variable name in the child class is the same as the variable names of the parent class, this is known as data hiding.

```

class A
{
 int x;
 void setA(int val)
 {
 x = val;
 }
}
class B extends A
{
 int x;
 void setB(int val)
 {
 x = val;
 }
 void show()
 {
 System.out.println(" x of A : " + super.x);
 System.out.println(" x of B : " + x);
 }
}
class DataHidingTest
{
 public static void main(String []s)
 {

```



```

 B ref = new B();
 ref.setA(50);
 ref.setB(60);
 ref.show();
 }
}
E:\JAVA\Programs\Core Java Program\DataHiding>java DataHidingTest
x of A : 50
x of B : 60

```

- For the child class we can access the parent class variable by using the super keyword.
- Super is nothing but it is only the casted this.
- We can also write (+ ((A)this.x))

## **Binging of the variable in java**

This is just the checking of the declaration of any variable when any variable is used in the program that always be validated with the declaration.

There are two types of the binding: -

- i). Compile Time Binding (CTB).
- ii). RunTime Binding (RTB).

### **1. Compile Time Binding (CTB): -**

Compile time binding always be performed by seeing types of reference variable from outside the class within the class compile time binding is performed with the members of the same class.

E.g.

```

class A
{
 int x;
 void setA(int val)
 {
 x = val;
 }
}
class B extends A
{
 int x;
 void setB(int val)
 {
 x = val;
 }
 void show()
 {
 System.out.println(" x of A : " + super.x);
 System.out.println(" x of B : " + x);
 }
}

```

```

class BindingTest
{
 public static void main(String []s)
 {
 B ref = new B();
 ref.setA(50);
 ref.setB(60);
 ref.show();
 }
}
E:\JAVA\Programs\Core Java Program\DataHiding\CTB>javac BindingTest.java
E:\JAVA\Programs\Core Java Program\DataHiding\CTB>java BindingTest
x of A : 50
x of B : 60

```

- Compile time binding can be seeing in the class file by using the java command.
- In the above program the binding of the variable are shown.
- All variable always be bind at the compile time.
- That means there is no run time binding of the variables.

## 2. RunTime Binding (RTB): -

- Run time binding always be performed by java & run time environment by seeing the type of object.
- The non-static methods except the private and final methods are always bind both time that means compile time and run time.
- Non-static methods at the compile time bind with type of reference variable and at the run time they bind by using the object.
- If the type of reference variable and the type of object is changed then we can visualize.
- Compile time binding and runtime binding separately
- The reference variable of parent class can hold the reference id of the child class of object.

```

class A
{
 void m1()
 {
 System.out.println(" m1 of A");
 }
}
class B extends A
{
 void m2()
 {
 System.out.println(" m2 in B");
 }
}

```

```

}
class BindingTest
{
 public static void main(String s[])
 {
 A ref = new B();
 ref.m1();
 }
}
E:\JAVA\Programs\Core Java Program\DataHiding\RTB>javac BindingTest.java
E:\JAVA\Programs\Core Java Program\DataHiding\RTB>java BindingTest
m1 of A

```

[**Note:** - All the methods whose binding perform at the runtime is child dynamic method dispatching.]

## **Method Overriding**

- When the child class has the method with the same signature as the method is already available in the parent class when this is known as the method overriding.
- Method overriding available in java is only use to achieve the dynamic method dispatching.
- During the method overriding in the child class we can't assign the weakest access privilege.

**In java there are four types of access specifier.**

1. Private
2. Default
3. Protected
4. Public

1. Private  $\leftrightarrow$  most restricted / weakest access privilege.
2. Public  $\leftrightarrow$  most restricted / strongest access privilege.

- If the restriction label becomes high is comparison to the method of the parent class this is known as the weakness access privilege.

```

class A
{
 void m1()
 {
 System.out.println(" m1 in class A");
 }
}
class B extends A
{
 /* private void m1() //compile time error */
 void m1()

```



```

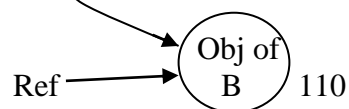
 {
 System.out.println(" m1 in B");
 }
 }
}
class Sample
{
 public static void main(String []s)
 {
 A ref = new B();
 ref.m1();
 }
}
E:\JAVA\Programs\Core Java Program\Method_Overriding\Sample>javac Sample.java
E:\JAVA\Programs\Core Java Program\Method_Overriding\Sample>java Sample
m1 in B

```

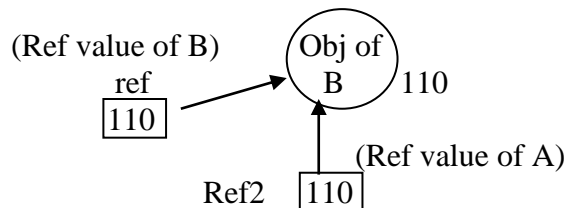
## Upcasting & Downcasting

- String the reference id of the child class object into the reference variable of the parent class is known as the upcasting.
- Upcasting can be direct or indirect.

A ref = new B( ); //direct upcasting



B ref = new B( );  
A ref2 = ref;



- Through the upcasting we can only invoked the method which are declares in the parent class that means the unique method of the child class can't be invoked.

```

class A
{
 void m1()
 {
 }
 void m2()
 {
 }
}
class B extends A
{

```

```

 void m1()
 {
 }
 void m2()
 {
 }
 }
class MainSample
{
 public static void main(String []s)
 {
 A ref = new B();
 ref.m1();
 ref.m2();
 //ref.m3(); // compile error
 }
}

```

➤ Block diagram of class B

|                        |       |   |                                                        |
|------------------------|-------|---|--------------------------------------------------------|
| inheritance            | m2( ) | ✓ | Method that can be invoked by the variable of A class. |
| Overriding method in B | m1( ) | ✓ |                                                        |
| Self method of B       | m3( ) | X |                                                        |

## **Downcasting**

- Downcasting is not the opposite of upcasting that means downcasting never says that is the reference variable of child class stored the reference id of parent object.
- It is never possible to store the reference id of parent object into the reference variable of child class.
 

```

B ref = new A(); //never possible
A ref = new B(); //Upcasting
B ref2 = (B) ref; //Downcasting

```
- Downcasting is a process always be performed after the upcasting.
- In the process of downcasting the reference id of the child objects into the reference variable into the object.

### **Why upcasting**

- Upcasting is used to achieve the dynamic method dispatching & through the dynamic method dispatching the runtime polymorphism will be achieved.

### **Why downcasting**

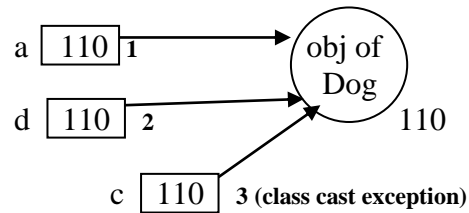
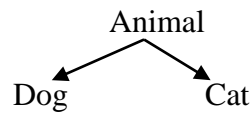
- Downcasting is required to invoke the unique method of the child class.

## **Rules for the down Casting**

1. Down casting always be explicit otherwise compilation error will be generated.

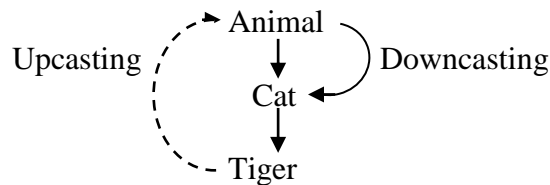
- The down casting always must be performed in the same hierarchy in which the up casting performed.

For Example: - There is the hierarchy of animal type class.

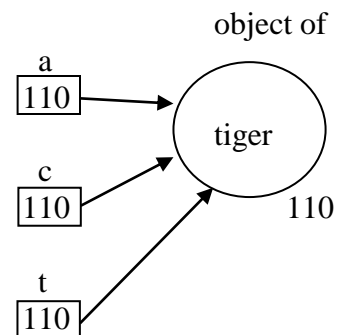


- Animal a = new Dog ( );
- Dog d = (Dog) a; // correct down casting
- Cat c = (Cat) a; //wrong down casting / class cast exception

- If we performed the wrong down casting then class cast exception will be generated at the runtime.
- Problem of the down casting always be raised at the runtime not compile time because object are the runtime of the java.
- In the multilevel inheritance down casting can be perform up to the same level or up to any middle level.



- Animal a
- a = new Tiger( ); //up-casting
  - Cat c = (Cat) a; //correct down-casting
  - Tiger t = (Tiger)a;
- Animal a
- a = new Tiger( );
  - Cat c = (Cat) a;
  - Tiger t = (Tiger)a; //class cast exception



**[Note: -** If the two classes are not in the child parent relationship which other then the casting can't be possible that means compilation error will be generated.]

class Student

```

{

}

```

Employee e = new student( ); //compilation error

class Employee

```

{

}

```

Employee e = new Employee( );  
student st = (student) e; //compilation error

[**Note:** - We can never copy the reference variable of the child implicitly, if always be perform explicitly.]

```
Animal a = new cat();
cat c;
c = a; //compilation error
c = (cat) a; //correct
```

➤ **Practical use of the up-casting and down-casting**

```
class Animal
{
 void eat()
 {
 System.out.println("eat in animal");
 }
 static void forest(Animal a)
 {
 a.eat();
 }
}
```

- At the compile time a.eat( ); will be bind with the cat( ) method of the animal class itself but in actual at the run time eat( ) method will be invoked of the child whose object contained by the reference variable of the animal class(a) in the argument of forest( ) method.
- a.eat( ); statement shows the abstraction for the developer of the animal class.

```
class Animal
{
 void eat()
 {
 System.out.println(" eat in animal");
 }
 static void forest(Animal a)
 {
 a.eat();
 }
}
class Dog extends Animal
{
 void eat()
 {
 System.out.println(" eat in Dog");
 }
}
class Cat extends Animal
{

```

```

 void eat()
 {
 System.out.println(" eat in Cat");
 }
 }
class Test
{
 public static void main(String []s)
 {
 Dog d = new Dog();
 Cat c = new Cat();
 Animal.forest(d);
 Animal.forest(c);
 }
}
E:\JAVA\Programs\Core Java Program\Casting\Sample>javac Test.java
E:\JAVA\Programs\Core Java Program\Casting\Sample>java Test
eat in Dog
eat in Cat

```

**[Note: -** In the java all the methods are not dispatched dynamically rather same methods are dispatched (bind) statically and same methods are dispatched (bind) dynamically.]

- The compile will always decide that at the runtime which method will be dispatched dynamically by the java JRE will follow the static (compile time) binding.
- Compiler writes the following assembly instructions in the .class file to internal the JRE.

| Assembly Instruction | Type of Binding | Type of Method Calling                                                                                            |
|----------------------|-----------------|-------------------------------------------------------------------------------------------------------------------|
| Invoke Static        | Static          | Calling of static method.                                                                                         |
| Invoke special       | Static          | Calling of static method calling of private static method, constructor calling, and method call by super keyword. |
| Invoke virtual       | Dynamic         | Non-private and non-static method calling.                                                                        |
| Invoke interface     | Dynamic         | Method of the interfaces.                                                                                         |

**[Note: -** Java runtime environment perform the dynamic dispatching if in the child class the same signature method (overriding method) is found that means at the time of compile time binding and runtime binding the method signature can be different, so with the overloading method we can never achieve the dynamic binding.]

```

class A
{
 void m1(float f)
 {
 System.out.println(" m1 in A ");
 }
}

```

```

}
class Test
{
 static void check (A ref)
 {
 ref.m1(50); //50 is the int type argument
 }
}
class B extends A
{
 void m1(int x) //Overloaded method
 {
 System.out.println(" m1 in B");
 }
 public static void main(String []s)
 {
 B b = new B();
 Test.check (b);
 }
}
E:\JAVA\Programs\Core Java Program\Casting\Sample1>javac Test.java
E:\JAVA\Programs\Core Java Program\Casting\Sample1>java B
m1 in A

```

### **Rule of the dynamic dispatching**

At the runtime during the dynamic method dispatching the signature of the method can't be changed with the signature of the method which was bind at the compile time.

### **Polymorphism**

- One name many form known as the polymorphism it is the most important feature of the OOPS.
- In the OOPS based technology the **method overloading**, **operator overloading** and **method overriding** are used to achieve the polymorphism.

### **Depending upon the type of binding there are two types of polymorphism.**

1. Compile time polymorphism
2. Run time polymorphism

#### **1. Compile time polymorphism**

- If the one form amongst the multiple forms selected at the compile time of child not be changed at the runtime then it is known as the compile time polymorphism.

**E.g.:** - Method overriding and operator overloading.

## 2. Runtime polymorphism

- If the selection of any one form is performed amongst the multiple forms as the runtime, then it is known as the runtime polymorphism.

**E.g.:** - Method overriding.

- When parent class has **private method**, then method overriding is not possible.

[**Note:** - Private method can't be overridden that means method overriding can't be performed on the private method of the parent class.]

- That's why the binding of the private method calling always be the static (invokes special).

## instanceof Operator

- **instanceof** operator is the keyword in the java that works as an operator.
- The work of the instanceof operator is to check the object type contained by any reference variable.

**E.g.:**

```
Animal a = new Dog();
if (a instanceof Dog)
{
 System.out.println("Dog type instance");
}
if (a instanceof Cat)
{
 System.out.println("Cat type instance");
}
```

- The result of instanceof operator always be the Boolean value.
- Practical use of the instanceof operator is to prevent the **ClassCastException** during the down casting.

```
class Animal
{
 void eat()
 {
 System.out.println(" eat in animal");
 }
 void drink()
 {
 System.out.println(" drink in animal");
 }
 static void forest(Animal a)
 {
 a.eat();
 if (a instanceof Dog)
 {
 Dog d = (Dog) a;
```

```

 d.power();
 }
 if (a instanceof Cat)
 {
 Cat c = (Cat) a;
 c.hunt();
 }
}
class Dog extends Animal
{
 void eat()
 {
 System.out.println(" eat in Dog");
 }
 void power()
 {
 System.out.println(" power in Dog");
 }
}
class Cat extends Animal
{
 void eat()
 {
 System.out.println(" eat in Cat");
 }
 void hunt()
 {
 System.out.println(" hunt in Cat");
 }
}
class Test
{
 public static void main (String []s)
 {
 Dog d = new Dog();
 Cat c = new Cat();
 Animal.forest(d);
 Animal.forest(c);
 }
}

```

E:\JAVA\Programs\Core Java Program\Casting\INSTANCEOF>javac Test.java

E:\JAVA\Programs\Core Java Program\Casting\INSTANCEOF>java Test

```

eat in Dog
power in Dog
eat in Cat
hunt in Cat

```



- Instance of the operator in the multilevel interface.

```

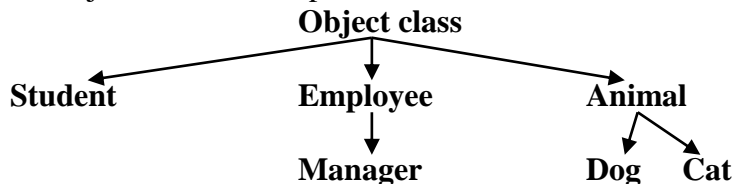
A
↑
B
↑
C
A ref = new C();
System.out.println(ref instanceof B); //true
System.out.println(ref instanceof C); //true
System.out.println(ref instanceof A); //true

A ref = new B();
System.out.println(ref instanceof B); //true
System.out.println(ref instanceof C); //false
System.out.println(ref instanceof A); //true

```

## Object class

- Java follows the single root hierarchy that means if there is no parent of any class then the object class will be parent of the class.



[Note: - Directly or indirectly each and every class is the child of the object class.]

```

Object ref = new Student (); // correct
Object ref2 = new Animal (); // correct
Object ref3 = new Cat (); // correct

```

[Note: - With the reference variable of object class only the method of object class itself can be invoked but the object class methods mostly are the non-static and non-private, so compile time assembly instruction will be invoked virtual, so through the dynamic dispatching they can be invoke of the child class it available in the overriding form.]

## Method of the object class

- **public int hashCode ( )**
- **public String toString ( )**
- **public boolean equals (Object ob)**
- **etc.**

### 1. hashCode ( ) method

- These methods return an integer value known as the **hashCode** that is generated from the actual address of the object.
- For the two different objects the value of **hashCode** always be the different.

```

class Student
{
 int rollno, marks;

```

```

Student(int rollno, int marks);
{
 this.rollno = rollno;
 this.marks = marks;
}
void show()
{
 System.out.println(" Rollno = " + rollno);
 System.out.println(" Marks = " + marks);
}
}
class HashCodeTest
{
 public static void main(String []s)
 {
 Student st1 = new Student (101, 85);
 Student st2 = new Student (102, 75);
 System.out.println(st1.hashCode ());
 System.out.println(st2.hashCode ());
 }
}
E:\JAVA\Programs\Core Java Program\Method\HashCode>javac HashCodeTest.java
E:\JAVA\Programs\Core Java Program\Method\HashCode>java HashCodeTest
31168322
17225372

```

## 2. toString() method

- toString method returns the id of the object the reference id always be in following format.

**className@hexadecimalvalue**

(generated from the hashCode value released by the hashCode( ))

```

class Student
{
 int rollno, marks;
 Student(int rollno, int marks)
 {
 this.rollno = rollno;
 this.marks = marks;
 }
 void show()
 {
 System.out.println(" Rollno = " + rollno);
 System.out.println(" Marks = " + marks);
 }
}

```

```

class ToStringTest
{
 public static void main(String []s)
 {
 Student st1 = new Student(101, 85);
 Student st2 = new Student(102, 75);
 System.out.println(st1.hashCode());
 System.out.println(st1.toString());
 System.out.println("\n" +st2.hashCode());
 System.out.println(st2.toString());
 }
}
E:\JAVA\Programs\Core Java Program\Method\ToString>javac ToStringTest.java
E:\JAVA\Programs\Core Java Program\Method\ToString>java ToStringTest
31168322
Student@1db9742
17225372
Student@106d69c

```

**[Note: -** The toString( ) method internally invoke to the hashCode( ) method to get hashCode value.]

```

class Student
{
 int rollno, marks;
 Student(int rollno, int marks)
 {
 this.rollno = rollno;
 this.marks = marks;
 }
 public int hashCode()
 {
 return(marks);
 }
 void show()
 {
 System.out.println(" Rollno = " + rollno);
 System.out.println(" Marks = " + marks);
 }
}
class ToStringTest
{
 public static void main(String []s)
 {
 Student st1 = new Student(101, 85);
 Student st2 = new Student(102, 75);
 }
}

```

```

 System.out.println(st1.hashCode());
 System.out.println(st1.toString());
 System.out.println("\n" + st2.hashCode());
 System.out.println(st2.toString());
 }
}
E:\JAVA\Programs\Core Java Program\Method\ToString1>javac ToStringTest.java
E:\JAVA\Programs\Core Java Program\Method\ToString1>java ToStringTest
85
Student@55
75
Student@4b

```

### Example: -

```

class A
{
 void m1()
 {
 System.out.println(" m1 in A");
 }
 void m2()
 {
 this.m1();
 }
}
class B extends A
{
 void m1()
 {
 System.out.println(" m2 in B");
 }
}
class Test
{
 public static void main(String []s)
 {
 B ref = new B ();
 //A ref = new A ();
 ref.m2 ();
 }
}
E:\JAVA\Programs\Core Java Program\Method\Practice>javac Test.java
E:\JAVA\Programs\Core Java Program\Method\Practice>java Test
m2 in B

```

**Question:** - How the toString ( ) method works in the object class?

➤ Different of toString ( ) as follow in the object class:

```
public String toString ()
{
 int x = hashCode ();
 string = Integer.toHexString (x); //to convert into hexadecimal
 return (getClass (). getName () + "@" + str);
}
```

(Class name / returns the class name of the object through  
which this method invoked.)

➤ There are the 10 println methods in print stream class.

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

(Object of printStream class)

```
class PrintStream
{
 public void println(int x)
 {

 }
 public void println(char ch)
 {

 }
 public void println(boolean b)
 {

 }
 println () are available for all primitives

 public void println(String str)
 {
 println the str onto the console
 }
 public void println(Object ob)
 {
 String refId = ob.toString();
 ---- Code to print the reference Id-----
 }
}
```

**Edit on the Student class**

```
class Student
{
 int rollno, marks;
 Student(int rollno, int marks)
```

```

 {
 this.rollno = rollno;
 this.marks = marks;
 }
 public int hashCode()
 {
 return(marks);
 }
 public String toString()
 {
 return(rollno + " : " + marks);
 }
 void show()
 {
 System.out.println(" Rollno = " + rollno);
 System.out.println(" Marks = " + marks);
 }
 }
}
class ToStringTest
{
 public static void main(String []s)
 {
 Student st1 = new Student (101, 85);
 Student st2 = new Student (102, 75);
 System.out.println(st1.hashCode());
 System.out.println(st1.toString());
 System.out.println(st1);
 System.out.println("\n" + st2.hashCode());
 System.out.println(st2.toString ());
 System.out.println(st2);
 }
}
E:\JAVA\Programs\Core Java Program\Method\ToSting2>javac ToStringTest.java

E:\JAVA\Programs\Core Java Program\Method\ToSting2>java ToStringTest
85
101 : 85
101 : 85

75
102 : 75
102 : 75

```

### **equals() Method**

- equals () method in the object class use to compare the objects on the basis of the reference id.

#### **Signature**

- **public boolean equals(Object ob)**

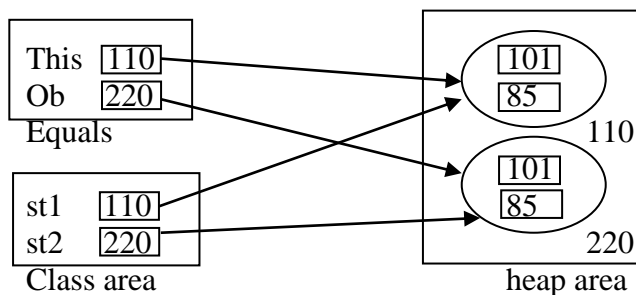
```

class Student
{
 int rollno, marks;
 Student(int rollno, int marks)
 {
 this.rollno = rollno;
 this.marks = marks;
 }
 void show()
 {
 System.out.println(" Rollno = " + rollno);
 System.out.println(" Marks = " + marks);
 }
}
class EqualsTest
{
 public static void main (String []s)
 {
 Student st1 = new Student (101, 85);
 Student st2 = new Student (101, 85);
 System.out.println (st1.equals(st2));
 }
 public boolean equals (Object ob)
 {
 return (this == ob);
 }
}

```

E:\JAVA\Programs\Core Java Program\Method\Equals>javac EqualsTest.java

E:\JAVA\Programs\Core Java Program\Method\Equals>java EqualsTest  
false



### **Overriding the equals ( ) method in the student class**

```

class Student
{
 int rollno, marks;
 Student (int rollno, int marks)
 {
 this.rollno = rollno;
 }
}

```

```

 this.rollno = marks;
 }
 public int hashCode ()
 {
 return(marks);
 }
 public boolean equals(Object ob)
 {
 if(ob instanceof Student)
 {
 Student st = (Student)ob;
 return(st.rollno == this.rollno);
 //return(ob.rollno == this.rollno); //error
 }
 return(false);
 }
 void show()
 {
 System.out.println(" Rollno : " +rollno);
 System.out.println(" Marks: " +marks);
 }
}
class EqualsTest
{
 public static void main(String []s)
 {
 Student st1 = new Student(101, 85);
 Student st2 = new Student(101, 85);
 System.out.println(st1.equals(st2));
 }
}
E:\JAVA\Programs\Core Java Program\Method\Equals1>javac EqualsTest.java
E:\JAVA\Programs\Core Java Program\Method\Equals1>java EqualsTest
true

```

### **abstract keyword: -**

- Abstract is also the keyword in java.
- Abstract is the non-access modifier.
- Abstract keyword can be used with the classes and methods.
- Through the abstract keyword we achieve the abstraction and runtime polymorphism explicitly.

### **abstract class: -**

- Abstract class can never be instantiated that means we can never create the object of the abstract class.
- We can create the reference variable of the abstract class.



- The purpose of the abstract classes in java to maintain the inheritance (type of relationship) between the classes of the application.
- Abstract classes always be used in the inheritance never be used in the abstraction.

```
abstract class Animal
{
 void eat()
 {
 System.out.println("eat in animal");
 }
 static void check(Animal a)
 {
 a.eat();
 }
}
class Cat extends Animal
{
 // does not override method of Animal
 public static void main(String s[])
 {
 Cat c = new Cat();
 Animal.check(c); //Upcasting will be performed
 }
}
E:\JAVA\Programs\Core Java Program\Method\Abastract>javac Cat.java
E:\JAVA\Programs\Core Java Program\Method\Abastract>java Cat
eat in animal
```

- From the calling environment the check ( ) method always be invoked by passing the reference of any child object of the animal class. That means 100% surety of the up-casting.

### **abstract ( ) Method**

- abstract ( ) methods are also known as the incomplete method.
- abstract ( ) method only have their signature that means never have body.

**E.g.:**

**abstract void m1( )**

### **Rules of abstraction**

- If any class has any abstract method then that class have to be declared as the abstract class but each and every abstract class need not compulsory have the any abstract method.
- Abstract class without any abstract ( ) method – possible.
- Abstract ( ) method without abstract class - not possible.

**E.g.:**

```
abstract class A
{
 void m1()
 {

 }
}
```

} possible

**E.g.:**

```
abstract class A
{
 abstract void m1();
}
```

} possible

**E.g.:**

```
class A
{
 abstract void m1();
}
```

} not possible

**[Note: -** In the child class of any abstract ( ) method we have to override all the abstract ( ) method all the parent class otherwise the child class also have to be declared as the abstract class.]

- **ref.m1()** in the check method always invoked to the m1( ) method of the child class whose object is received in the argument in ref.

```
abstract class A
{
 abstract void m1();
 static void check(A ref)
 {
 ref.m1();
 }
}
```

- In java through the abstract class or abstract method there will be the 100% security of the run time polymorphism.

class B extends A

```
{
 void m1()
 {
 System.out.println(" m1 in the B");
 }
}
```

```

 }
}
class Test
{
 public static void main(String []s)
 {
 B b = new B();
 A.check(b);
 }
}
E:\JAVA\Programs\Core Java Program\Method\AbastractMethod>java Test
m1 in the B

```

- abstract( ) method can never be providing because the binding of the private method always be the static.
- abstract( ) method can never be static.

### **final keyword**

Final keyword also works as the non-access modifies, final keyword can be used with the **class, methods and variable**.

A final class if make any class final then it can never be inherited.

In the java library system and string classes are the final classes.

```

final class A
{

}
class B extends A
{
 void m1()
}

```

[**Note:** - final and abstract keyword can't be used together.]

They are like opposite of each other. So if any class is already abstract then we can't make then final vice-versa.

**abstract class means** – no association only inheritance.

**final class means** -- no inheritance only association.

### **final( ) method**

If any methods is marked as final then no one can override it is the child class that means final( ) method can't be overridden.

```

class A
{

```

```

 final void m1()
 {
 }
 }
class B extends A
{
 void m1() //error
}

```

**[Note: -** If any class is already final then there is no significance or use of making any final inside that class.]

**Question: -** Can final( ) method be static?

**Answer: -** Yes, but their hiding will not be possible.

**Question: -** Can final( ) method be private?

**Answer: -** Yes.

```

class A
{
 private final void m1()
 {
 }
}
class B extends A
{
 void m1() // correct, this is not method overriding as the private m1() from A class is
 { // not even inheritance.
 }
}

```

There is practically no use significance to make final( ) method as private.

**Question: -** Can final class contain abstract method?

**Answer: -** nope.

**Question: -** Can abstract class contain final( ) method?

**Answer: -** Yes.

## **Final variable**

In java final variable work as constants

The value in the final variable can never be changed, that means after one time initialization we can never change the value of final variable.

Final variable in java are written with all letter cap(conversion).

Final variable never have their default values.

## **There are 3 types of final variable.**

- 1) Static final variables:
- 2) Non-Static final variables / Instance final variables:
- 3) Local final variables:

### **1) Static final variables:**

These variables can't have their default value.

Final static variable can be initialized either at the time of their declaration or in the static block but window initializing any final variable can't be used.

```
class Test
{
 final static int n=0;
 static
 {
 x = 5; //error
 }
}
```

### **2) Non-Static final variables / Instance final variables:**

These variables can be initialized at the time of their

- i) Declaration
- ii) Initialization block
- iii) Constructors

### **3) Local final variables:**

Can be initialized at

- i) Declaration
- ii) At any other place within local scope.

```
Class Test
{
 void m1()
 {
 final int x = 10;
 x++; //error
 final int y;
 y = 5;
 y = 10; //error
 }
 void m2(final int z)
 {
 z++; //error
 }
}
```

## Interface

Interfaces are class like construction in java with lots of restrictions.

In interfaces all the method will be declared as abstract( ) method.

That means the interface all the methods can be only public and abstract (implicit).

In the interface all the variables are public final and static and can never be changed (implicitly).

We can make a reference variable of interface but can't create its objects.

After compilation the .class file also will be generated for the interface.

### In an interface there can't be

- i) **Constructor**
- ii) **Static block**
- iii) **Non-abstract( ) method**

Conventionally name of an interface should be an adjective.

Interface keyword in java is used to declare a new interface.

### Example

interface Testable

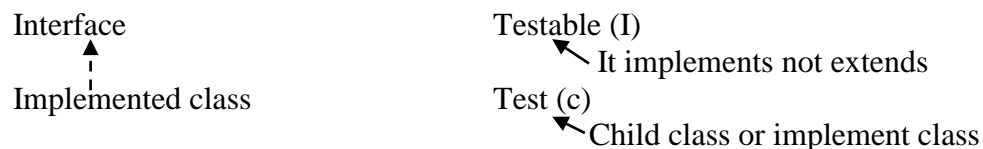
```
{
 [public final static] int x = 5;
 public final static int y = 10;
 public abstract void m1()
 [public abstract] void m2()
 protected abstract void m3() //error
}
```

→ implicitly placed keyword

### implements keyword

It used to make the child class of an interface is known as implemented class of the interface and must override all the abstract( ) method of the interface.

A child class will always implements an interface and never extends the interface.



A class is also a type of relationship between implemented classes.

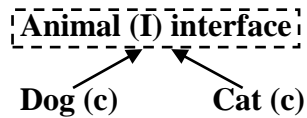
So, implemented class is a type of interface.

In reference variable of interface we can store the object of any implemented class.

**Animal a = new Dog ( );**

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

Dog is the type of animal so with the reference variable of interface we can only call (bind) the method declared is that interface but at the runtime all the calling of the methods of interface are dispatched dynamically because methods of the interface will always be non-static and non-private.



[**Note:** - Dog is a type of animal.]

```
interface Animal
{
 public abstract void eat () ;
 public abstract void drink () ;
}
class Demo
{
 static void check (Animal a)
 {
 a.eat () ;
 a.drink () ;
 }
}
class Dog implements Animal
{
 public void eat ()
 {
 System.out.println("eat in the Dog");
 }
 public void drink()
 {
 System.out.println("Drink in Dog");
 }
 public static void main (String args[])
 {
 Demo.check (new Dog ());
 }
}
class Cat implements Animal
{
 public void eat ()
 {
 System.out.println("eat in Cat");
 }
 public void drink()
 {
 System.out.println("drink in Cat");
 }
 public static void main (String args[])
 {
 Demo.check (new Cat ());
 }
}
```

```

 }
}
E:\JAVA\Programs\Core Java Program\Method\Interface>java Cat
eat in Cat
drink in Cat

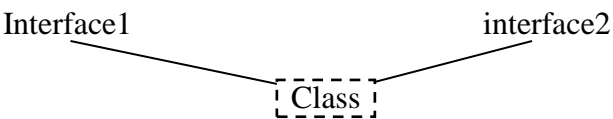
E:\JAVA\Programs\Core Java Program\Method\Interface>java Dog
eat in the Dog
Drink in Dog

```

Interfaces are also use in java to active the abstraction and runtime polymorphism with 100% surety.

Through the interface the developers prepares the set of abstract ( ) method as the slandered for futuristic development.

Any class can implement more than two interfaces.



```

interface I1
{
 void m1();
 void m2();
}
interface I2
{
 void m1();
 void m2();
}
class C1 implements I1, I2
{
 public void m1()
 {
 System.out.println(" m1 is C1");
 }
 public void m2()
 {
 System.out.println(" m2 is C1");
 }
 public void m3()
 {
 System.out.println(" m3 is C1");
 }
 public static void main(String args[])
 {
 C1 ref = new C1();
 I1 ione = ref;
 }
}

```



```

 I2 itwo = ref;
 ione.m1();
 ione.m2();
 //ione.m3(); //Error
 itwo.m1();
 itwo.m2();
 //itwo.m3(); //Error
 }
}

E:\JAVA\Programs\Core Java Program\Method\Interface1>javac Test.java

E:\JAVA\Programs\Core Java Program\Method\Interface1>java C1
m1 is C1
m2 is C1
m1 is C1
m2 is C1

```

Any interface can extend to any one or more other interface.

```

interface I1
{
 interface I1
 {
 }
 interface I2 extends I1
 {
 }
 class C1 implements I2
 {
 //Have to override both the methods of interface I1 & I2.
 }
}

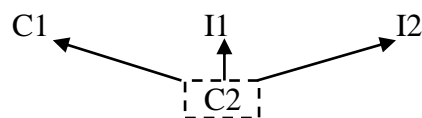
```

Any interface can never implement to the other interface.

Any class can extend to any other class along with implements to the other interface.

An interface can't extend a class.

In this case first extends and second implements.



```

interface I1
{
 void m1();
 void m2();
}
class C1
{

```

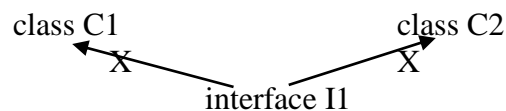
```

 public void m1()
 {
 System.out.println(" m1 in C1");
 }
 }
class C2 extends C1 implements I1
{
 public void m2()
 {
 System.out.println(" M2 in C2");
 }
 public static void main(String args[])
 {
 I1 ref = new C2();
 ref.m1();
 ref.m2();
 }
}
E:\JAVA\Programs\Core Java Program\Method\Interface2\p1>javac Test.java
E:\JAVA\Programs\Core Java Program\Method\Interface2\p1>java C2
m1 in C1
M2 in C2

```

**Rules:** - Each and every interface are child of object class (Exceptional case).

Any interface can never extend or implement another class (except object class) but exceptionally all the interface are the child of the object class.



```

interface I1
{
}
class Test implements I1
{
 public static void main(String s[])
 {
 I1 ref = new Test();
 System.out.println(ref.toString());
 }
}

```

From Object class

```
E:\JAVA\Programs\Core Java Program\Method\Interface2>javac Test.java
Test.java:6: error: Test is not abstract and does not override abstract method m2() in I1
class Test implements I1
^
1 error
```

## **PACKAGE**

**Packages** are the arrangement of the class in the form of folders, packages are not only the folder but something moves a folder.

### **Benefits of the package**

- |                   |                                       |
|-------------------|---------------------------------------|
| 1. Security       | 1. Arrangement of class and interface |
| 2. Collectiveness | 2. Achieving second level security.   |

### **Security**

Any developer can make their class members as default so that these members can be accessed outside the classes (within the package).

By putting classes into package the member of the class can be restricted through access by the class's mode by other developer, but within the package of Dev1 the classes can use the member (default member) of each other.

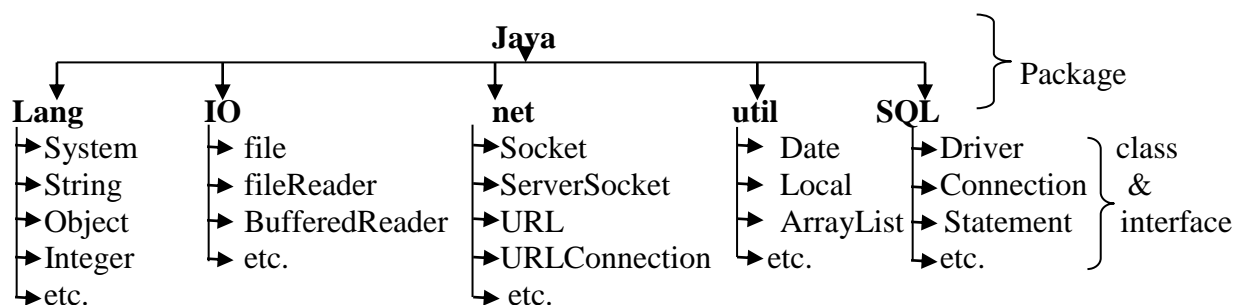
### **Collectiveness**

By using packages the classes related to any specific type of work can be grouped together, so that only by importing one package multiple classes can be used.

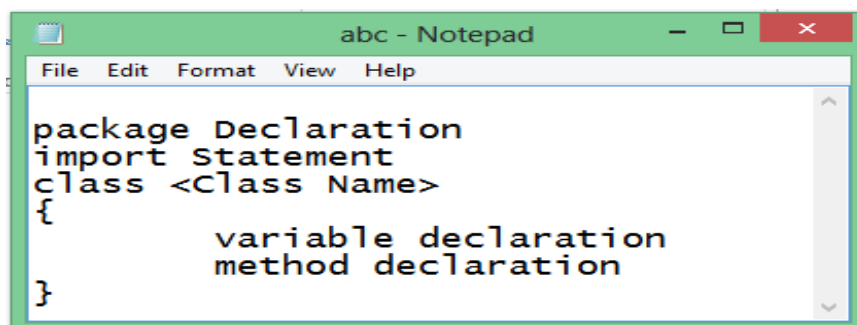
### **In java library there are following two main packages**

1. Java package
2. Javax package

[Note: - Further these packages contain sub-packages.]



### **Complete Syntax of .java file creation**



## **By using the package**

1. One type of the classes and interface can be grouped together for their easy accessibility in the program.
2. In java after the class level security by using the private specified any developer can make the class members and the class itself as default, so that these class member can't be accessed outside the package.

**Example: -** Importing the package or the class of the package?

**//Test.java**

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

```
class Test
```

```
{
```

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

```
 {
```

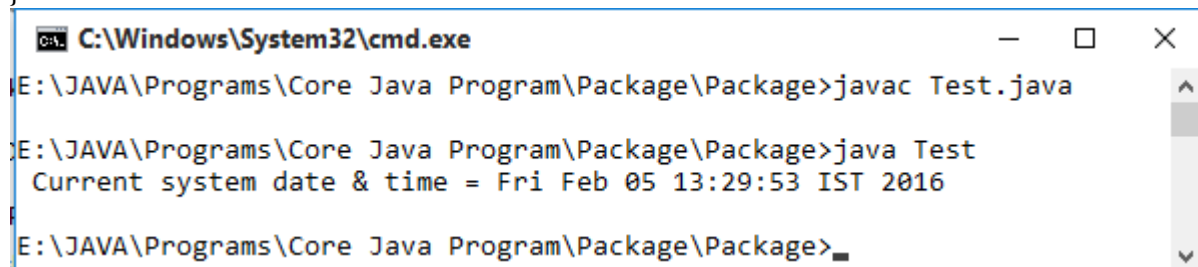
```
 Date d = new Date();
```

```
 System.out.println("current system date & time = " +d);
```

```
 //d.toString() will be invoked in method.
```

```
 }
```

```
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user has navigated to the directory "E:\JAVA\Programs\Core Java Program\Package\Package". They have entered the command "javac Test.java" to compile the program. Then, they entered "java Test" to run the program. The output of the program is displayed: "Current system date & time = Fri Feb 05 13:29:53 IST 2016". The prompt is now waiting for the next command.

[Note: - In the Date class the toString( ) method is available in the overridden format.]

## **Date class**

An instance of the **Date** class of **java.util** package represents the date and time.

The default constructor of date class always octane current system date and time form the native Operating System.

**toString( )** method in the **Date** class available in the overriding format.

Any package that we are importing in our program always be search out in the following sequence.

1. In the **rt.jar** (by the **BootStreap.class** loader)
2. In the **jre/lib/ext** folder (by the **Extended** class loader)
3. In the **class path** (by the **Systemlass** loader)

[Note: - **java** & **javax** are the reserve package names that mean we can't create our own package with these names.]

**Question: -** If any class in available without package and also available within the package that we are importing in our source code then from which location the class will be used?

**Answer:**

It depends upon how we are importing the package.

## import keyword

**import** keyword in java is used to import any package.

In **java.lang** package is by default import in each program.

In all java programs have java.lang package automatically imported.

In order to use a class of any package first of all we have to invoke that package into our program then we can use the classes.

**In the importing statement is as follows:**

- **import java.util.\*;**

This will import all the classes of the util package.

In this case 1<sup>st</sup> of all the class will be searched in all the three location (**rt.jar, then ext folder, then classpath**) all the location without the package folder.

If without package the class is found then it will be used if all three location without package class is not found then in the package the class will be searched in all location.

- **import java.util.Date;**

If we write the import statement link that then the Date class always be searched with the **java.util** package in all 3 location.

[**Note:** - The classes placed in the **ext folder** must be declared as the public and also must be existing in the **.jar** file]

We can use a class without importing

- It is in **java.lang** package
- It is in the current directory

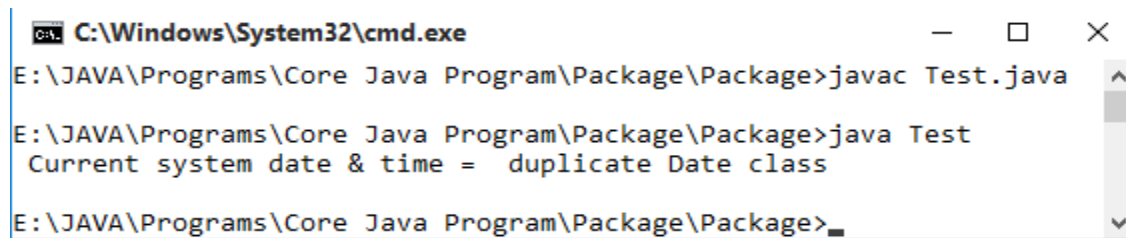
## Creating user defined Date class

**//Date.java**

```
class Date
{
 public String toString()
 {
 return(" duplicate Date class ");
 }
}
```

**//Test.java**

```
import java.util.*;
class Test
{
 public static void main (String s[])
 {
 Date d = new Date();
 System.out.println("current system date & time = " +d);
 //d.toString() will be invoked in method.
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\Package>javac Test.java
E:\JAVA\Programs\Core Java Program\Package\Package>java Test
Current system date & time = duplicate Date class
E:\JAVA\Programs\Core Java Program\Package\Package>
```

### Creating custom java/util folder

Now placing custom Date class is util folder package **java.util**.

**//Date.java**

```
package java.util;
public class Date
{
 public String toString()
 {
 return(" duplicate Date class ");
 }
}
```

**//PackTest.java**

```
import java.util.*;
class PackTest
{
 public static void main(String s[])
 {
 Date d = new Date();
 System.out.println("Current System Date and Time:"+d);
 }
}
```

```
E:\JAVA\Programs\Core Java Program\Package\package1>javac PackTest.java
E:\JAVA\Programs\Core Java Program\Package\package1>java PackTest
Current System Date and Time:Fri Feb 05 15:32:32 IST 2016
```

### package keyword

**package** keyword is used in **.java** file to declare the package.

If any java file there is the package declaration then after the compilation the **.class** file must be placed in the package folder.

**//Emp.java**

```
package p1.p2;
public class Emp
{
 private int code, salary;
 public Emp(int code, int salary)
 {
 this.code = code;
 this.salary = salary;
 }
}
```

```

 }
 public String toString()
 {
 return(code+ " : " +salary);
 }
}

```

[Note: - In the class can be either **public** or **default**.]

Default class can be used within the other class if same package.

Public class can be used in the same package or in any class.

The other package by using the import statement

Any class which is public and have the package declaration can be used in the class of the any other package only by the import statement.

Save the **Emp.java** in the following folder.

**D:\abc\p1\p2\Emp.java**

```

graph TD
 A[D:\abc\p1\p2\Emp.java] --> B[Classpath]
 A --> C[package folder]
 A --> D[package]

```

**Compile the Emp.java**

```
E:\JAVA\Programs\Core Java Program\Package\Package2\p1\p2>javac Emp.java
```

```
E:\JAVA\Programs\Core Java Program\Package\Package2\p1\p2>
```

---

Create the **Company.java** and use **Emp** class by importing their package.

**//Company.java**

```
import p1.p2.Emp;
```

```
class Company
```

```
{
```

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

```
 {
```

```
 Emp e = new Emp(101, 3000);
```

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

```
 }
```

```
}
```

Save the **Company.java** in the regular folder there all the programs are saved.

Set the **classpath**

```
C:\>set classpath = D:\abc ←
```

Compile the **Company.java**

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\Package2>set classpath=E:\JAVA\Programs\Core Java Program\Package\Package2\p1\p2;

E:\JAVA\Programs\Core Java Program\Package\Package2>javac Company.java

E:\JAVA\Programs\Core Java Program\Package\Package2>java Company
101 : 3000

E:\JAVA\Programs\Core Java Program\Package\Package2>
```

## classpath

**classpath** are the location in the file system from where the compiler locates the class and packages and at the runtime the system class loader of the JRE loads the classes.

By default only the current directory is included in the classpath that's why the compiler and execution we have to set the directory as the prompt where the **.java** file and **.class** file of our program situated.

### To check the existing classpath

C:\>set classpath

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\Package2>set classpath
classpath=E:\JAVA\Programs\Core Java Program\Package\Package2\p1\p2

E:\JAVA\Programs\Core Java Program\Package\Package2>_
```

### To set the class path

C:\>set classpath = E:\abc; (to set the current directory in the classpath)

(Command name) (variable environment name) (class path separator)

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\Package2>set classpath=E:\JAVA\Programs\Core Java Program\Package\Package2\p1\p2

E:\JAVA\Programs\Core Java Program\Package\Package2>_
```

[Note: - In order to locate the **.java** that is to be compiled the compiler never consider to the class path.]

E:\.....\Package>javac Company.java

The **Company.java** must be existing in package Test folder.

If **Company.java** is exist on any other location then we have to specify the full path.

E:\.....\Package>javac d:\abc\Company.java



To retain the old user defined path.

**C:\>set classpath = % classpath%; d:\abc;**

(To retain the old user defined classpath)

### **-d option of javac tool**

**-d** stands for destination.

Through the **-d** at the time of compilation we can set the destination folder of the **.class** file.

**Javac -d d:\temp Test.java**

(Destination folder of the Test.class)

**Example: -**

**Javac -d d:\abc Company.java**

**Note: -** If we compile the any **.java** file with **-d** and that **.java** file have package declaration then **-d** switch also automatically creates the package folder.

### **Compile Employee.java using -d**

**Javac -d . Employee.java**

Space  
(Dot current directory)

```
E:\JAVA\Programs\Core Java Program\Package\Package2>javac -d . Emp.java
```

```
E:\JAVA\Programs\Core Java Program\Package\Package2>_
```

### **-cp option of javac and java tool**

**-cp** stands for classpath without setting the classpath by using this option we can specify the class path locations to the **javac** and **java** tools.

**Syntax:-**

**Javac -cp d:\abc;. Company.java**

**Example:**

**Java -cp d:\abc;. ompany.java**

(;- Is check classpath)

(. – automatically assume classpath as current directory)

### **Bad class file error**

Occurs when file that we are compile using another is not exist in there package declaration or to their package.

### **Steps related to -cp options**

1. Save **Company.java** at desktop in program folder.
2. Save **Emp.java** into d:\abc\p1\p2

Here p1 and p2 are package name that contain by the Emp.java file in there package declaration.

3. Now open **cmd** from location desktop\program.
4. Now use following command to compile Company.java.

**Javac -cp d:\abc;. Company.java**

5. To run Company.java used the command.

**Java -cp d:\abc;. Company**

### Steps for %classpath% command

1. Save Company.java at location  
**D:\abc**
2. Save 'p1' package at default folder link '**package**' folder name.  
**'package'= folder name**
3. **Set classpath = %classpath%;. D:\abc** to retain path
4. Open "cmd" at package folder where 'p1' is saved.
5. Compile **Company.java** using full address.  
**Java d:\abc\Company.java**

Since **Company.java** still not its current location so for searching company.java use full address.

6. Set path of current directory using command.  
**Set classpath = .**
7. Now retain path again using command set classpath.  
**Classpath = %classpath%; d:\abc**
8. Then compile again using full address.  
**Javac d:\abc\Company.java**

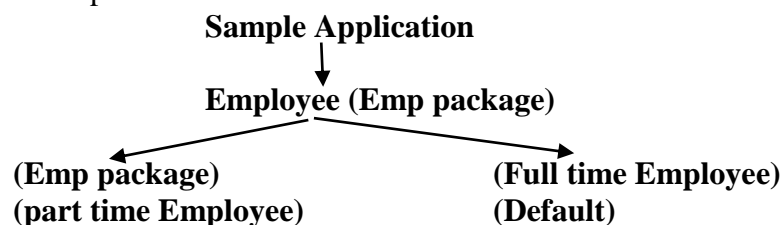
[**Note:** - If any .java file there is the package declaration then that .java file can contain it must one public class.]

[**Note:** - If any .java file there is any public class declaration then that .java file must be saved with the name of that public class.]

In order to create more than one public class in one package then we have to create save file with the same package declaration.

**Question:** - Why .java file save as the name of public class?

**Answer:** - Auto compilation.



```
//Employee.java
package Emp;
public class Employee
{
```

```

private int code;
int tax;
public Employee(int code)
{
 this.code = code;
}
public void setTax(int code)
{
 this.tax = tax;
}
public void report()
{
 System.out.println("Code: "+code);
}
protected int getTax()
{
 return(tax);
}
protected int getCode()
{
 return(code);
}
}

```

#### **//PartTimeEmployee.java**

```

package Emp;
public class PartTimeEmployee extends Employee
{
 private int rate, hom;
 public PartTimeEmployee(int code, int rate, int hom)
 {
 super(code);
 this.rate = rate;
 this.hom = hom;
 }
 public void calculate()
 {
 tax = (rate * hom) * 12/100;
 }
 public void report()
 {
 System.out.println("\nCode: "+getCode());
 System.out.println("Tax: " +tax);
 System.out.println("no of hours: " +hom);
 }
}

```

**//FullTimeEmployee.java**

```
package Emp;
public class FullTimeEmployee extends Employee
{
 private int basic;
 public FullTimeEmployee(int code, int basic)
 {
 super(code);
 this.basic = basic;
 }
 public void calculate()
 {
 int vt = basic * 12/100;
 setTax(vt);
 }
 public void report()
 {
 System.out.println("\nCode: "+getCode());
 System.out.println("Basic: " +basic);
 System.out.println("Tax: " +getTax());
 }
}
```

**//Company.java**

```
import Emp.Employee;
import Emp.PartTimeEmployee;
import Emp.FullTimeEmployee;
class Company
{
 public static void main(String s[])
 {
 Employee e = new Employee(101);
 e.setTax(5000);
 e.report();
 PartTimeEmployee pe = new PartTimeEmployee(102, 100, 150);
 pe.claculate();
 pe.report();
 FullTimeEmployee fe = new FullTimeEmployee(103, 2500);
 fe.calculate();
 fe.report();
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\Package3>javac -d . Employee.java
E:\JAVA\Programs\Core Java Program\Package\Package3>javac -d . FullTimeEmployee.java
E:\JAVA\Programs\Core Java Program\Package\Package3>javac -d . PartTimeEmployee.java
E:\JAVA\Programs\Core Java Program\Package\Package3>javac Company.java
E:\JAVA\Programs\Core Java Program\Package\Package3>java Company
Code: 101

Code: 102
Tax: 0
no of hours: 150

Code: 103
Basic: 2500
Tax: 0

E:\JAVA\Programs\Core Java Program\Package\Package3>
```

**Example:** - Can we create the main class with in the package?

**Answer:** - Yes, but always be run with the package name.

```
package mypack;
```

```
public class Test
```

```
{
```

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

```
 {
```

```
 System.out.println(" This is main");
```

```
 }
```

```
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Package\package4>javac -d . Test.java
E:\JAVA\Programs\Core Java Program\Package\package4>java mypack.Test
This is main

E:\JAVA\Programs\Core Java Program\Package\package4>
```

## EXCEPTION HANDLING

**Exception** is the abnormal conditions that can be arise at the runtime.

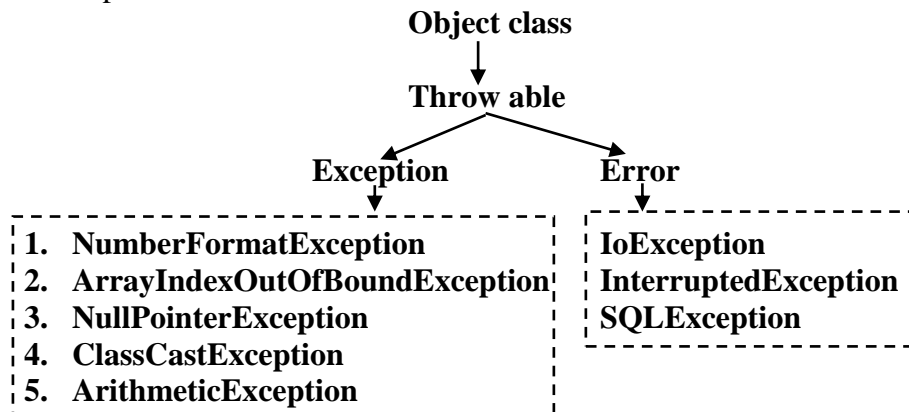
There abnormal situation may be raised by the wrong input given by the user exception can also be arise by the abnormal behavior by the runtime resource such as life system, network socket, database connection etc..

Handling exception means to write the code as the provision of the exceptional situations.

If the developer doesn't to make the provision in there code for handling the exception is arise at runtime then the definitely the program will terminate.

In **java.lang** package there is the hierarchy of exception classes at the runtime JRE will generate the exception by creating the object one of these exception classes.

By creating the object of these exception classes programmatically we can explicitly generate exception.



### Unchecked Exception

1. Error in java will arise at runtime and compile time but exception always be raised at the runtime.
2. Exception can be handled programmatic only but runtime error can never handled programmatically if they arise always terminate program.

**Example: -**

1. **Static OverflowError**
2. **JVM is running out of memory error.**

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

There is the un-terminated resource calling of m1( ) that will create infinite frame in stack.

```
//Test1.java
class Test1
{
 int x;
 Test1()
 {
 new Test1();
 }
 public static void main(String args[])
 {
 Test1 t = new Test1();
 }
}
```

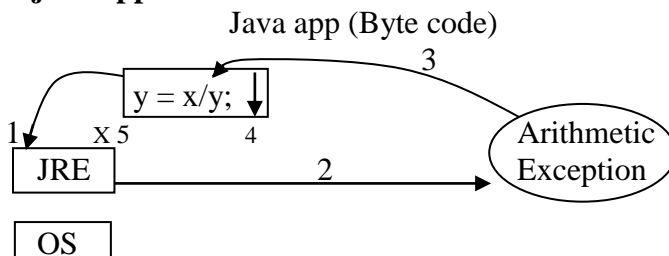
```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest1>javac Test1.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest1>java Test1
Exception in thread "main" java.lang.StackOverflowError
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
 at Test1.<init>(Test1.java:6)
```

**JVM** is running out of memory.

In case of unchecked exception it develops wants to handle the exception then have to write the code but in case the checked exception develops have to write the code in both the case either to handle then as if develop don't want to handle them.

[**Note:** - All the child of the **RuntimeException** classes is always considers as the unchecked Exception.]

**Diagram to show the flow of the generation unchecked Exception by the JRE and handling by the java application**



1. Division operation encountered and statement is moved in the JRE for their exception.
2. Value of denominate is '0' so exception object is created.
3. Reference of exception object provided into the application.
4. If the handler (catch block) is existing then execution of the program is continuous.

5. If handle doesn't exist then the reference of the execution return to works the JRE and program gets terminated

### **Keyword related to the exception handling**

- try
- catch
- throw
- throws
- finally

#### **try keyword**

**try** keyword used to make the block known as the try block that contains the java code that can generate the exception.

#### **Catch keyword**

**catch** keyword also used to make the block known as the code block. Catch block contains the code for the exception handling.

#### **throw keyword**

**throw** keyword used to generate the exception either checked or unchecked implicitly.

#### **throws keyword**

**throws** keyword mainly used in case of checked exception to propagate the exception.

#### **finally keyword**

**finally** keyword also used to make the block known as the finally block always be executed either exception is generated or not and it generated then handle or not.

**Question: -** A java program to take two numeric values from command line argument handles and performs the division operation.

**//ExpTest.java**

class ExpTest2

```
{
 public static void main(String args[])
 {
 int x, y, z;
 x = Integer.parseInt(args[0]);
 y = Integer.parseInt(args[1]);
 System.out.println(" You enter number : " + x + " " + y);
 z = x/y; //Can generate the ArithmeticException
 System.out.println("Result= "+z);
 }
}
```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest2>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest2>java ExpTest 40 20
You enter number : 40 20
Result= 2
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest2>_

```

**ArithmeticException** will be generated.

### Creating the try & catch block to handle the Exception.

//ExpTest3.java

```
class ExpTest3
```

```
{
```

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

```
 {
```

```
 int x, y, z;
```

```
 try
```

```
 {
```

```
 x = Integer.parseInt(args[0]);
```

```
 y = Integer.parseInt(args[1]);
```

```
 System.out.println(" You enter number : " + x + " " + y);
```

```
 z = x/y; //Can generate the ArithmeticException
```

```
 System.out.println("Result= "+z);
```

```
 }
```

```
 catch(ArithmeticException e)
```

```
 {
```

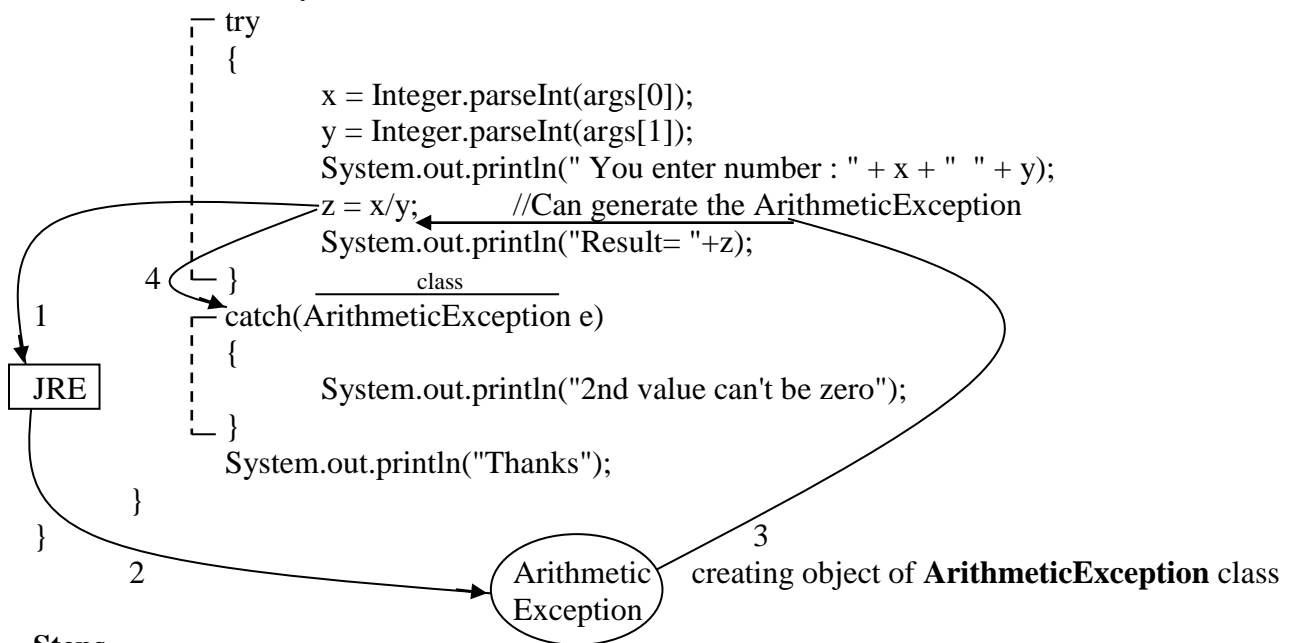
```
 System.out.println("2nd value can't be zero");
```

```
 }
```

```
 System.out.println("Thanks");
```

```
 }
```

```
}
```



#### Steps

1. Division statement encountered and values of the 'x and y' obtained by the JRE.
2. Value of 'y' is found as zero by the JRE then execution object is created.
3. Reference of the execution object provided into main( ) method.
4. Reference of the Exception object cached in the catch block.

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest3>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest3>java ExpTest 40 20
You enter number : 40 20
Result= 2
Thanks
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest3>

```

[**Note:** - In case of unchecked exception if try block doesn't exist or proper catch block doesn't exist then unchecked exception automatically moved towards JRE.]

If any exception got by the JRE from the application then immediately the program will be terminated.

### Explicit generation of the exception

//ExpTest4.java

class ExpTest

{

    public static void main(String args[])

    {

        int x, y, z;

        try

        {

            x = Integer.parseInt(args[0]);

            y = Integer.parseInt(args[1]);

            System.out.println(" You enter number : " + x + " " + y);

            if(y <= 0)

            {

                throw new ArithmeticException( );

                throw new ArithmeticException("y can't be 0");

            }

            z = x/y;

            //Can generate the ArithmeticException

            System.out.println("Result= "+z);

        }

        catch(ArithmeticException e)

        {

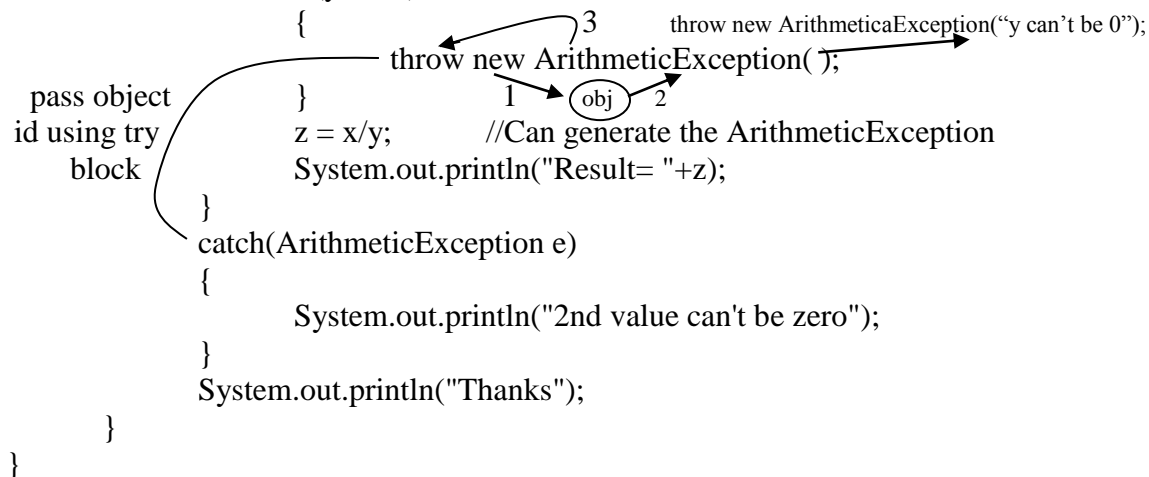
            System.out.println("2nd value can't be zero");

        }

        System.out.println("Thanks");

    }

}



```

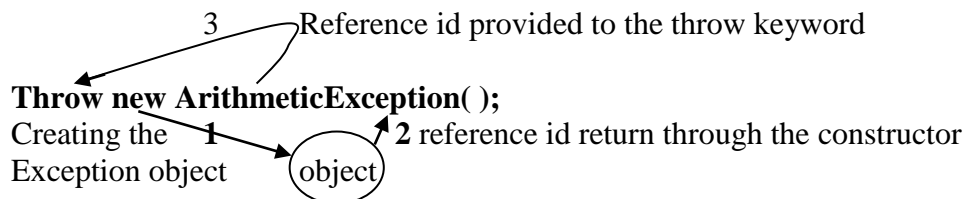
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest4>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest4>java ExpTest 40 20
You enter number : 40 20
Result= 2
Thanks
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest4>

```

If the try block not found then reference id of **ArithmeticException** will pass the JRE.  
Through explicit generation of any exception we can make out own condition.  
In case of implicit generation JRE already have the condition in which the exception will be raised.

### Steps to generate Exception explicitly

1. Create the exception object.
  2. Provide the reference id of the exception object through the throw keyword.
- Both of these steps will perform in the single statement.



In case of unchecked exception if the throw keyword exists in the try block then the reference id of the execution diverted towards catch block otherwise moved towards JRE

```

printStackTrace();

```

If the throws class is use to print all the information related to the execution such as line number name of exception

```

Catch(ArithmeticException e)
{
 e.printStackTrace();
}

```

In each and every exception class toString( ) method available in the overriding format printStackTrace( ) method internally invokes the toString.

## Method of the exception object

//MyThrowable.java

```
class MyThrowable
```

```
{
```

```
 void printStackTrace()
```

```
 {
```

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

```
 }
```

```
 }
 class MyArithmeticException extends MyThrowable
```

```
 {
```

```
 public String toString()
```

```
 {
```

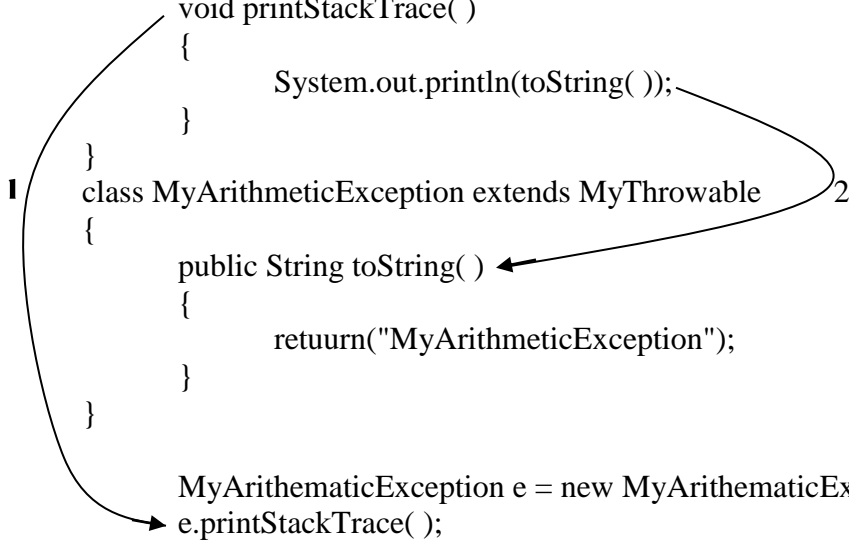
```
 return("MyArithmeticException");
```

```
 }
```

```
 }
```

```
 MyArithmeticException e = new MyArithmeticException();
```

```
 e.printStackTrace();
```



[Note: - In each and every exception class there are two types of constructor.]

1. Default constructor
2. Parameterized constructor

Declared **ArithmeticException** class as follows in the **java.lang** package

//ArithmeticException.java

```
public class ArithmeticException extends RuntimeException
```

```
{
```

```
 String msg, className;
```

```
 ArithmeticException()
```

```
 {
```

```
 msg = " ";
```

```
 }
```

```
 ArithmeticException (String msg)
```

```
 {
```

```
 this.msg = msg;
```

```
 }
```

```
 public String toString()
```

```
 {
```

```
 className = getClass().getName();
```

```
 return(className + " : " + msg);
```

```
 }
```

```
}
```

Calling the parameterized constructor

## Polymorphic catch

In catch block we can declare the reference variable of any parent exception class.

## Rules to declared try-catch block

Try block must be associated with at least one catch block or one finally block with the one try block there can be multiple catch block.

That means from any one try block different type of exception can be generated.

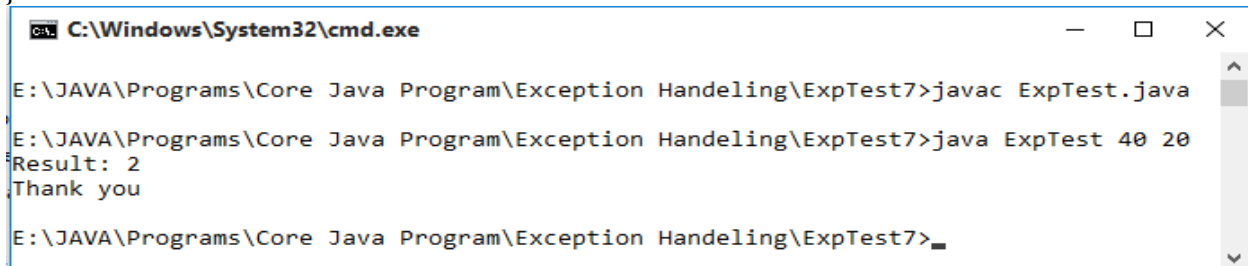
If there is the finally block then it will always be placed after the try and catch block.

There can be almost one finally blocks there can't be further any other statement.

In between try, catch blocks there can't be further any other statement.

**//ExpTest6.java**

```
class ExpTest
{
 public static void main(String s[])
 {
 int x, y, z;
 try
 {
 x = Integer.parseInt(s[0]);
 y = Integer.parseInt(s[1]);
 z = x/y;
 System.out.println("Result: "+z);
 }
 catch(NumberFormatException e)
 {
 e.printStackTrace();
 }
 catch(ArithmeticException e)
 {
 e.printStackTrace();
 }
 catch(ArrayIndexOutOfBoundsException e)
 {
 e.printStackTrace();
 }
 System.out.println("Thank you");
 }
}
```

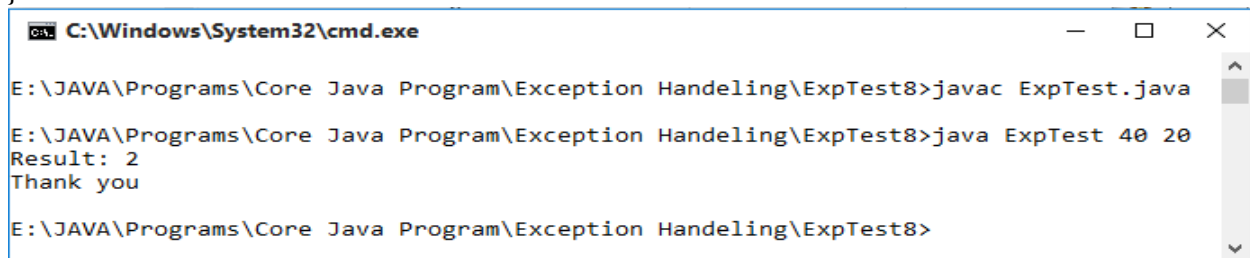


The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The prompt is at "E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest7>". The user has entered "javac ExpTest.java" and "java ExpTest 40 20". The output shows "Result: 2" and "Thank you".

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest7>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest7>java ExpTest 40 20
Result: 2
Thank you
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest7>
```

**//ExpTest7.java**

```
class ExpTest
{
 public static void main(String s[])
 {
 int x, y, z;
 try
 {
 x = Integer.parseInt(s[0]);
 y = Integer.parseInt(s[1]);
 z = x/y;
 System.out.println("Result: "+z);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 System.out.println("Thank you");
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>java ExpTest 40 20
Result: 2
Thank you
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>
```

It will handle all type of exception.

**Question: -** In polymorphic catch block just identity the actual exception object.

**Answer: -** e.printStackTrace( );

**//ExpTest8.java**

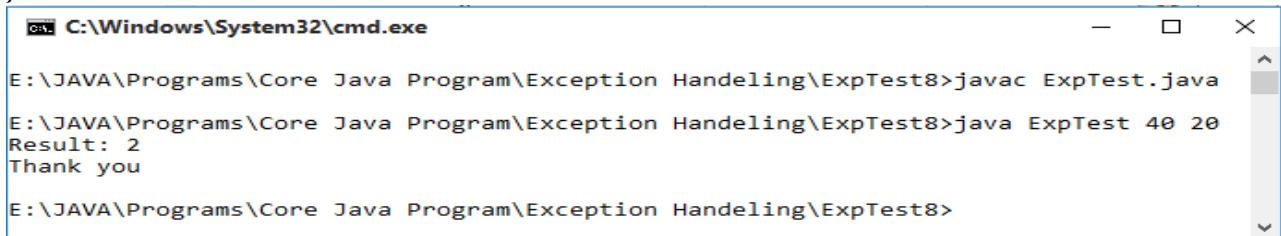
```
class ExpTest
{
 public static void main(String s[])
 {
 int x, y, z;
 try
 {
 x = Integer.parseInt(s[0]);
 y = Integer.parseInt(s[1]);
 z = x/y;
 System.out.println("Result: "+z);
 }
 catch(Exception e)
 {

```

```

 e.printStackTrace();
 }
 System.out.println("Thank you");
}
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>java ExpTest 40 20
Result: 2
Thank you
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest8>

```

## Nested try catch block

Within the one try block further the sequence of new try, catch can be started.

```

try
{
 try
 {
 Statement;
 }
 catch(..... e)
 {
 Statement;
 }
}
catch(..... e)
{
 Statement;
}

```

If any exception is raised in the inner try block then first of all it will be diverted to the inner catch block and if the catch block is not according to generated then execution is forwarded towards outer catch.

**//NestedTryTest.java**

```

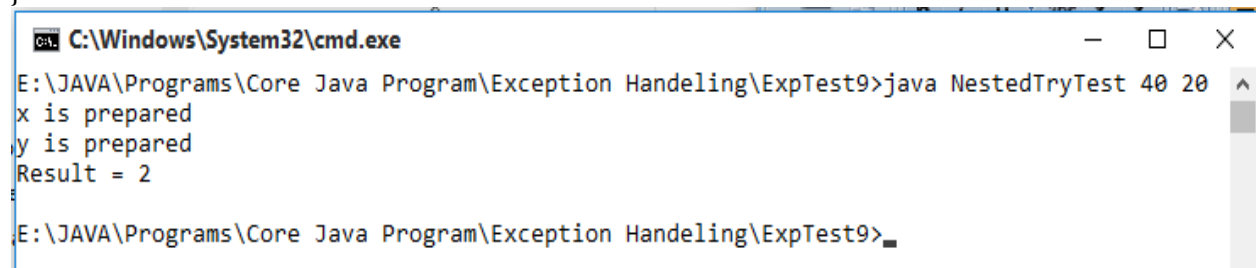
class NestedTryTest
{
 public static void main(String s[])
 {
 int x, y, z;
 x = y = z = 0;
 try
 {
 try
 {
 x = Integer.parseInt(s[0]);
 System.out.println("x is prepared");
 }
 }
 }
}

```

```

 }
 catch(NumberFormatException e)
 {
 x = 10;
 }
 try
 {
 y = Integer.parseInt(s[1]);
 System.out.println("y is prepared");
 }
 catch(NumberFormatException e)
 {
 y = 2;
 }
 z = x/y;
}
catch(ArrayIndexOutOfBoundsException e)
{
 System.out.println("insufficient values supplied");
}
catch(ArithmeticException e)
{
 z = 1;
}
System.out.println("Result = "+z);
}
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handling\ExpTest9>java NestedTryTest 40 20
x is prepared
y is prepared
Result = 2
E:\JAVA\Programs\Core Java Program\Exception Handling\ExpTest9>

```

## **Finally block**

Finally block are always gets executed either exception is raised or not if raised then handle or not.

There can be only one finally block with any try block.

Finally block can never be existing without try block.

Exception of the finally block doesn't means handling of the exception.

It can be declared in last the program.

### **//FinallyTest.java**

```

class FinallyTest
{
 public static void main(String s[])

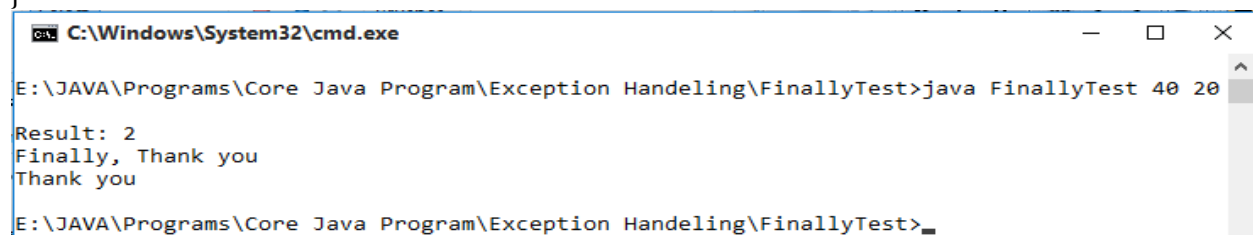
```



```

{
 int x, y, z;
 try
 {
 x = Integer.parseInt(s[0]);
 y = Integer.parseInt(s[1]);
 z = x/y;
 System.out.println("Result: "+z);
 }
 catch(ArithmeticException e)
 {
 e.printStackTrace();
 }
 finally
 {
 System.out.println("Finally, Thank you");
 }
 System.out.println("Thank you");
}
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handling\FinallyTest>java FinallyTest 40 20
Result: 2
Finally, Thank you
Thank you
E:\JAVA\Programs\Core Java Program\Exception Handling\FinallyTest>

```

If we put return statement in finally

It is never recommended to put '**return**' statement in the finally block because it always dissolve to the unhandled exception.

```

finally
{
 System.out.println("Finally, Thank you");
 return;
}

```

In this case **System.out.println("Thank you");** is unreachable code and hence we have to either remove this **System.out.println("Thank you");** or comment it.

## **Benefit of finally block**

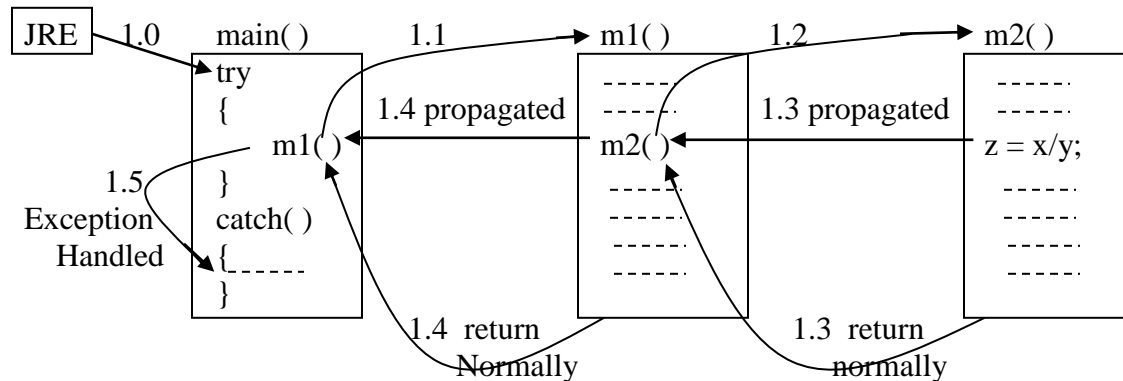
To invoke most important code like closing a file, closing database connections

It always be use to write the important code such as closing the file, closing the database connections.

## **Exception propagation**

Any exception can be handled in the same method in which it is raised or it can be handled in the calling function.

The unchecked exceptions are automatically propagated in the calling function.  
The unchecked, unhandled checked exceptions have to be propagated in the calling function by using the throws keyword.



### Example of Exception propagation

// ExpTest9.java

import java.io.\*;

class ExpTest

{

    public static void main(String s[ ])

    {

        try

        {

            int x = Integer.parseInt(s[0]);

            int y = Integer.parseInt(s[1]);

            int z = calc (x, y);

            System.out.println("Result = "+z);

        }

        catch(ArithmeticException e)

        {

            e.printStackTrace( );

        }

        System.out.println("Thank you");

    }

    static int calc(int a, int b)

    {

        System.out.println("Welcome in calc");

        try

        {

            int c = a/b;

            return(c);

        }

        finally

        {

            System.out.println("This is finally block");

        }

    }

```
}
}

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest9>java ExpTest 40 20
Welcome in calc
This is finally block
Result = 2
Thank you
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest9>_
```

[Note: - In case of return statement the finally block also gets executed.]

[Note: - In case of **System.exit** the finally block will not be executed.]

### // ExpTest10.java

```
import java.io.*;
class ExpTest
{
 public static void main(String s[])
 {
 try
 {
 int x = Integer.parseInt(s[0]);
 int y = Integer.parseInt(s[1]);
 int z = calc (x, y);
 System.out.println("Result = "+z);
 }
 catch(ArithmeticException e)
 {
 e.printStackTrace();
 }
 System.out.println("Thank you");
 }
 static int calc(int a, int b)
 {
 System.out.println("Welcome in calc");
 try
 {
 int c = a/b;
 if(c == 2)
 {
 System.exit(0);
 }
 return(c);
 }
 finally
 {
 System.out.println("This is finally block");
 }
 }
}
```

```
}
}

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest10>javac ExpTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest10>java ExpTest 40 20
Welcome in calc
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest10>_
```

### //ExpTest11

```
import java.io.*;
class ExpTest
{
 public static void main(String s [])
 {
 System.out.println("Welcome");
 try
 {
 int x = Integer.parseInt(s[0]);
 int y = Integer.parseInt(s[1]);
 int z = calc(x,y);
 System.out.println("Result = "+z);
 }
 finally
 {
 System.out.println("finally in the main");
 }
 System.out.println("Thank You");
 }
 static int calc(int a, int b)
 {
 System.out.println("Welcome in calc");
 try
 {
 int c = a/b;
 if(c==2)
 {
 System.exit(0); //terminate the program immediately
 }
 return(c);
 }
 finally
 {
 return(4);
 }
 }
}
```

```

}
E:\JAVA\Programs\Core Java Program\Exception Handeling\ExpTest11>java ExpTest 40 20
Welcome
Welcome in calc

```

**System.exit(0);** is to ways to return the control from any method.

| Ways                                                    | Exception of finally |
|---------------------------------------------------------|----------------------|
| 1. After compilation of method when return type is void | Yes                  |
| 2. Via return statement                                 | Yes                  |
| 3. When unchecked exception is propagated               | Yes                  |
| 4. <b>System.exit( );</b>                               | No                   |

### **throws keyword**

In java the checked exception does not have the capabilities of the automatic propagation rather we have to propagate that checked exceptions via throws keyword.

**//ThrowsUse1.java**

```

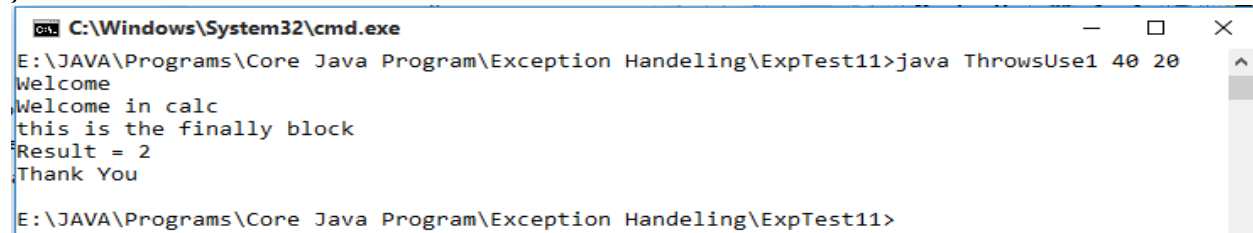
import java.io.*;
class ThrowsUse1
{
 public static void main(String s [])throws IOException
 {
 System.out.println("Welcome");
 try
 {
 int x = Integer.parseInt(s[0]);
 int y = Integer.parseInt(s[1]);
 int z = calc(x,y);
 System.out.println("Result = "+z);
 }
 catch(ArithmeticException e)
 {
 e.printStackTrace();
 }
 System.out.println("Thank You");
 }
 static int calc(int a, int b) throws IOException
 {
 System.out.println("Welcome in calc");
 try
 {
 if(b<=0)
 {
 throw new IOException("y cannot be zero");
 }
 int c = a/b;

```

```

 return(c);
 }
 finally
 {
 System.out.println("this is the finally block");
 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Exception Handling\ExpTest11>java ThrowsUse1 40 20
Welcome
Welcome in calc
this is the finally block
Result = 2
Thank You
E:\JAVA\Programs\Core Java Program\Exception Handling\ExpTest11>

```

## **Thumb rule about checked exception**

In case of checked exception we have to write the code either for their handling by using **catch( )** block or we have to write the code to propagate the checked exceptions by using **'throws keyword'**.

If we don't perform any one of these tasks then a compilation error will be generated.

It doesn't mean that a checked exception is raised at compile time rather the compiler explicitly wants to the **catch( )** block or **throws keyword** by the programmer.

[**Note:** - If there is any method that has the **throws keyword** in their signature must be invoked carefully that means within the **try catch( )** block or must provide **throws keyword** in the signature of the calling method.]

In the Java library, following methods are same methods that throw the checked exception from their signature.

### **Java.lang.InterruptedExcep**

Public void join( )throws InterruptedException  
[method of java.lang thread class]

### **Java.net.UnknownHostException**

Public static InetAddress.getByname(String hostName)throws UnknownHostException  
[method of the java.InetAddress class]

### **Java.io.IOException**

Public String readLine( )throws IOException  
[method of the java.io.BufferedReader class]

**Java.io.BufferedReader** class is the stream class that can be connected with the source of the data such as any file keyword etc.

ReaderLine method of this class is used to get the data from the source.

## **Program read the value from keyword**

```

//InputTest.java
import java.io.*;

```

```

class InputTest
{
 public static void main(String s[])throws IOException
 {
 BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
 System.out.print(" Enter your name : - ");
 String name = br.readLine();
 System.out.println(" Hello " +name);
 }
}
E:\JAVA\Programs\Core Java Program\Exception Handeling\InputTest>javac InputTest.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\InputTest>java InputTest
Enter your name : - XYZ
Hello XYZ

```

### Using the method of the java library

**Java.io.BufferedReader** class is the class which is capable to read the data from the stream (Data channel). `readLine( )` method is used to read the data from the stream.

#### **//InputTest1.java**

```

import java.io.*;
class InputTest
{
 public static void main(String s[])throws IOException
 {
 BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
 System.out.print(" Enter your name : - ");
 try
 {
 String name = br.readLine();
 System.out.println(" Hello " +name);
 }
 catch(IOException e)
 {
 e.printStackTrace();
 }
 }
}
E:\JAVA\Programs\Core Java Program\Exception Handeling\InputTest1>javac InputTest1.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\InputTest1>java InputTest1
Enter your name : - ABC
Hello ABC

```

### Custom exception

Custom exceptions are the user defines exceptions.

It can be the checked exception or unchecked exception.

**Java.lang.Exception**

↕  
MyException  
(checked custom exception)

**java.lang.RuntimeException**

↕  
MyException  
(unchecked custom exception)

### **Creating a custom exception named InvalidMarksException.**

**//InvalidMarksException.java**

```
package p1.myexp;
public class InvalidMarksException extends Exception
{
 String msg;
 public InvalidMarksException()
 {
 msg = " ";
 }
 public InvalidMarksException(String msg)
 {
 this.msg = msg;
 }
 public String toString()
 {
 return("InvalidMarksException : "+msg);
 }
}
```

**//Student.java**

```
import p1.myexp.InvalidMarksException;
class Student
{
 int rollno, marks;
 Student(int rollno)
 {
 this.rollno = rollno;
 }
 void exam(int marks)throws InvalidMarksException
 {
 if(marks < 0 || marks > 100)
 throw new InvalidMarksException("marks should within 0 - 100");
 this.marks = marks;
 }
 void show()
 {
 System.out.println("Rollno: "+rollno);
 System.out.println("Marks: "+marks);
 }
}
```



```
// School.java
import p1.myexp.InvalidMarksException;
class School
{
 public static void main(String s[])
 {
 int x = Integer.parseInt(s[0]);
 int y = Integer.parseInt(s[1]);
 Student st = new Student(x);
 try
 {
 st.exam(y);
 st.show();
 }
 catch(InvalidMarksException e)
 {
 e.printStackTrace();
 }
 }
}
```

```
C:\Windows\System32\cmd.exe

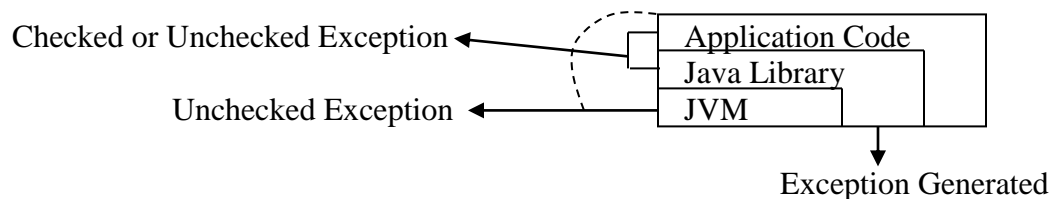
E:\JAVA\Programs\Core Java Program\Exception Handeling\School>javac -d . InvalidMarksException.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\School>javac Student.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\School>javac School.java
E:\JAVA\Programs\Core Java Program\Exception Handeling\School>java School 101 58
Rollno: 101
Marks: 58
```

**[Note: -** In case of checked exception we have to perform one of the two treatments.]

1. Either handles the exception by the try, catch block.
2. Propagate the exception by using throws keyword.

If now of these are performed then the compilation error will generated.

**[Note: -** Checked exception always is generated programmatically by using the throw keyword either from the user code or from the code of the library.]



### **Different between the checked and unchecked Exception**

|    | <b>Unchecked Exception</b>                                                                                                           | <b>Checked Exception</b>                                                                                                                                 |
|----|--------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. | Always be the child of runtime exception                                                                                             | These are the child of exception class.                                                                                                                  |
| 2. | That has the capacity of automatic propagation.                                                                                      | <b>throw</b> keyword is used to propagation them.                                                                                                        |
| 3. | Compilers never pay attention in the hanging of the unchecked exception.                                                             | Checked exception are known to the compiler that means compiler will generate the error if there is no try, catch or throw change for checked exception. |
| 4  | <b>Purpose:</b> - The purpose of unchecked exception is to take the treatment of exception handling from the environment optionally. | <b>Purpose:</b> - Is to intimate the calling environment about the exception.                                                                            |

# ARRAY

Array is the collection of similar data type.

All the elements in the array always are stored in the continuous location.

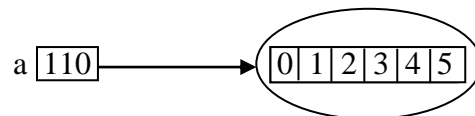
When there is requirement to store bulk of values of same type for the same purpose then we can use the array.

Benefit of array is to easy manage the data that means we can perform operation of the data easily such as sorting, searching etc.

In java all the arrays are always is the object that means we will create the array by using the new keyword.

**Syntax: -**

```
Int []a; ↗ reference variable of array
a = new int [5]; ↘ Creation of array object
```



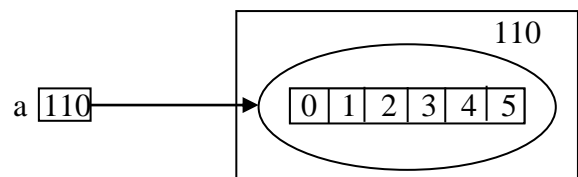
```
Ex: - int i, x = 0;
 int a[10];
 for(i = 0; i <= 10; i++)
 {
 x = x+10;
 a[i] = x;
 }
```

- For each and every array there is the separate proxy class created at the runtime by the JRE.
- Format of the name of these proxy classes.

```
int [] —————→ I[@Hexavalues
float[] —————→ F[@hexavalues
double[] —————→ D[@hexavalues
```

**[Note: -** Array elements always have the default values.]

```
int []a;
a = new int [5];
for(int i = 0; i < a.length; i++)
{
 System.out.println(a[i]);
}
```



## Some more valid syntax to declare the array in java

1. `int [ ]a = new int [5];` //single line declaration of array
2. `int a[ ] ;`

```

int []b;
3. int a[] = {5, 6, 7, 8, 9};
 Internal code
 Performed by overloaded
 "=" operator
 new int [4];
 a[0] = 5; a[1] = 6; a[2] = 7; a[3] = 8; a[4] = 9;

4. int a[];
 a = {5, 6, 7, 8, 9}; //Invalid
 a = new int[]{5, 6, 7, 8, 9}; //Valid

```

### Invalid Syntax

```

int a[5]; //Invalid
int a[] = new int[] //Invalid

```

**Question:** - create a class array until to perform the different operations on the array.

**//ArrayUtil.java**

```

class ArrayUtil
{
 static void input(int []a)
 {
 for(int i=0; i <a.length; i++)
 {
 a[i] = (int)(Math.random()*100); //to gnerate the value randomly
 }
 }
 static void output(int []a)
 {
 for(int i = 0; i < a.length; i++)
 {
 System.out.print(" "+a[i]);
 }
 }
}

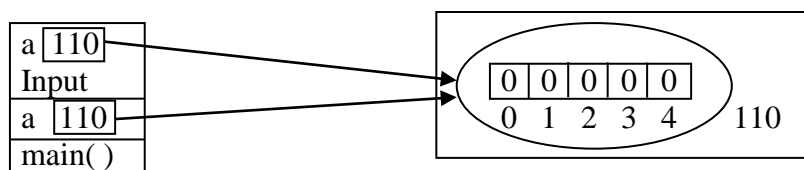
```

**// ArrayTest.java**

```

class ArrayTest
{
 public static void main(String []s)
 {
 int a[] = new int[5];
 ArrayUtil.input(a);
 ArrayUtil.output(a);
 }
}

```



```

C:\Windows\System32\cmd.exe

E:\JAVA\Programs\Core Java Program\Array>javac ArrayUtil.java

E:\JAVA\Programs\Core Java Program\Array>javac ArrayTest.java

E:\JAVA\Programs\Core Java Program\Array>java ArrayTest
5 81 11 41 28
E:\JAVA\Programs\Core Java Program\Array>

```

## Random function Signature

- **public static double random( )**  
                     0.6573 ← random vlaue

Random function randomly generate the numeric value within the within the range 0 to 1 (> 0 to 1<).

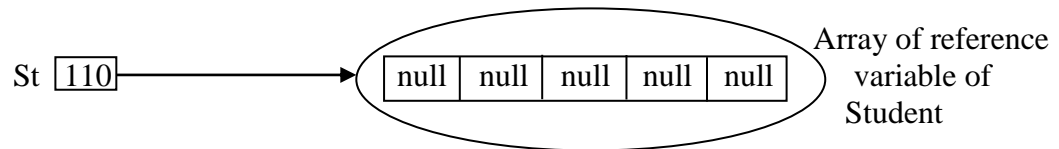
## Array of objects

```

Student st[];
St = new Student[5];

```

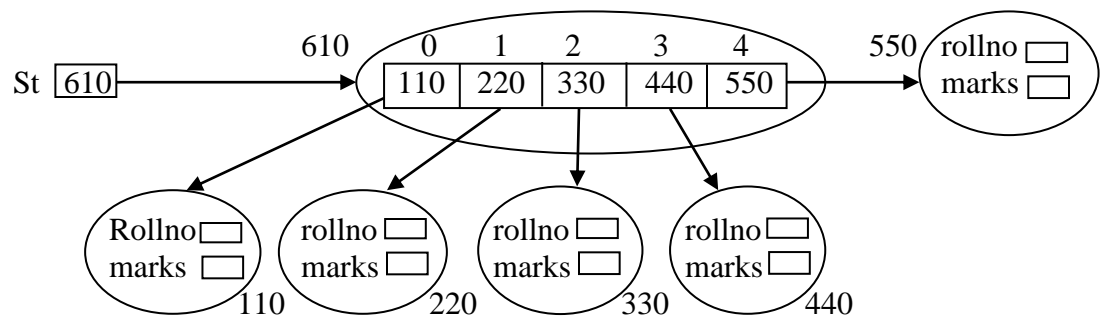
It will make the array of reference variable, not the array of objects.



```

for(int i=0; i<st.length; i++)
{
 st[i] = new Student();
}

```



```

st[i].admission(101);
st[i].admission(102);

```

## Two dimensional array (2D)/(Array of Arrays)

Array of Arrays means create the array of the multiple single dimensional Array.

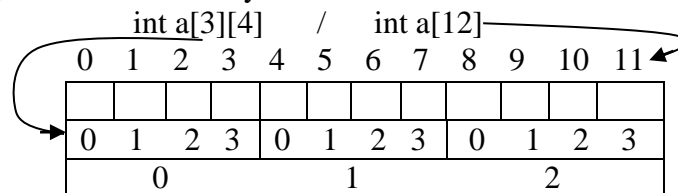
**Syntax:**

**Int a[][] =new int[3][4];**

(no. of rows) / (no of column)

(no of Arrays) / (no of element in each array)

In C language used the 2D Array.



//to start the elements

```
for(int i=0; i<a.length; i++)
```

```
{
```

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

```
 {
```

```
 a[i][j] = int (math.random()*100);
```

```
 }
```

```
}
```

//to print the array values

```
for(int i=0; i<a.length; i++)
```

```
{
```

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

```
 {
```

```
 System.out.println("\t" + a[i][j]);
```

```
 }
```

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

```
}
```

[**Note:** - In java we can create the array with different number of elements in each row.]

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

# AWT

## AWT (Abstract Window Toolkit)

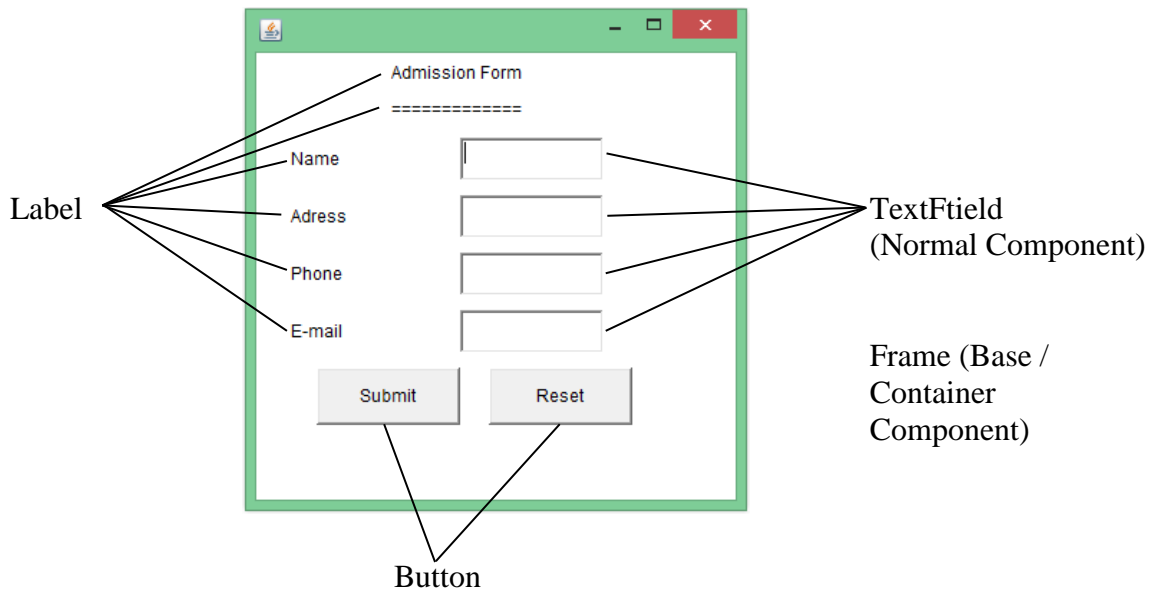
It is the first approach of java for window programming.  
AWT was introduced in the JDK1.1.

## java.awt package

**java.awt** package and there sub package provides the API (class and interface) for the programming of AWT.

## Two Types of development works are performed in AWT

1. Designing the GUI (Graphical User Interface) by using classes of **java.awt** package.
2. Making the GUI interface (Event Handling) by using classes and interfaces of **java.awt.event** package.



## java.lang package

- String
- System
- Object
- Integer
- Exception
- etc.

## java.awt package

- Component
- Button
- Frame
- TextField
- Label
- etc.

## Component Class is top most class

In the **java.awt** package there are the numbers of classes known as the component classes.

## There are two types of Component

1. Container (Base) Component
2. Normal Component

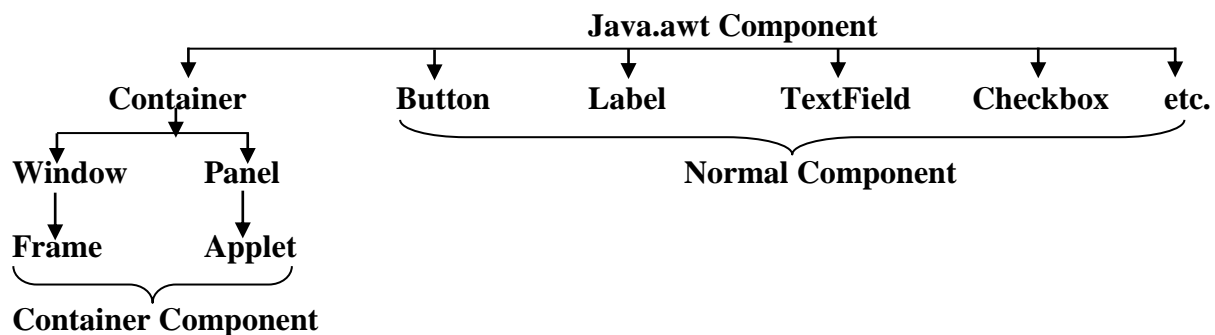
Container component provides the base for the normal components.

Normal components are used to design the GUI.

All the component classes are arranged in the hierarchy.

**java.awt** component class is the top most class for all the components.

## Hierarchy in the java.awt package



## Component class

Component class provides the basic functionality for all the components.

## Container class

Container class provides the basic functionality for the containership. It is the abstract class.

## Window

Window is the child of container class. It provides the facility of the **Title bar** and **border**.

## Frame class

Frame class is the child of the window and it provides the facility of **menu bar**.

## Panel

Panel is the hidden container. Panel never help there visibility. In order to make the panel visible we have to add them on to another container.

## Applet

Applet is the child of the panel. Applets are always gets visible on either on to the web browser on to the applet viewer.



## Methods of component class

- **public void setVisible(boolean b)**  
This method is use to make the component visible.
- **public void setSize(int weight, int height)**  
This method is use to set the size of the component.
- **public void setBounds(int x, int y, int weight, int height)**  
This method will set the cordinate and the size of the component.
- **public void setEnable(boolean b)**  
This method is use to enable and disable of the component.  
Disable component doesn't mean that it will not be displayed.

## Methods of the Container class

- **public void add(Component c)**  
This method will add the specified the component onto the container.
- **public void remove(Component c)**  
This method will remove the specified component.

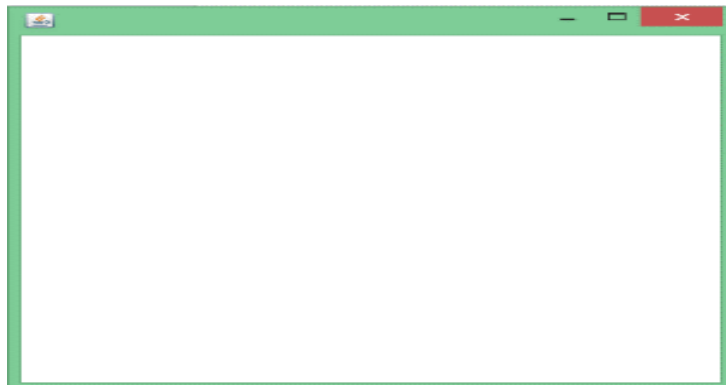
## Steps to create the Frame

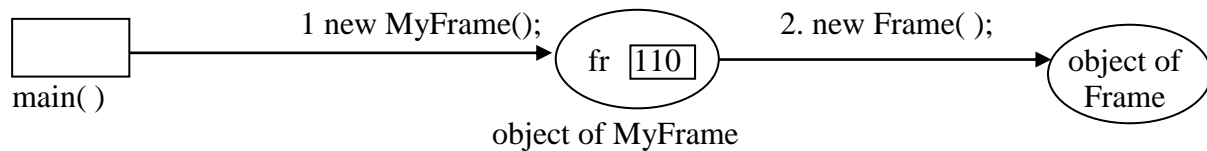
1. Create the object of frame class or its child class.
2. Set the size of the Frame.
3. Makes the frame visible.

## By creating object of Frame

```
//MyFrame.java
import java.awt.Frame;
class MyFrame
{
 Frame fr;
 MyFrame()
 {
 fr = new Frame();
 fr.setSize(400, 400);
 fr.setVisible(true);
 }
 public static void main(String s[])
 {
 new MyFrame();
 }
}
```

**Output -**





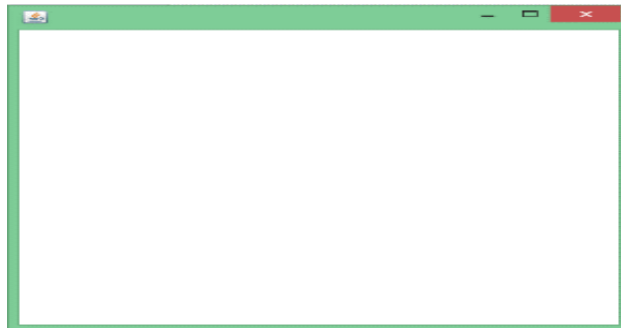
### **By creating the object of child class of frame**

```

//MyFrame.java
import java.awt.Frame;
class MyFrame1 extends Frame
{
 MyFrame1()
 {
 setSize(400, 400);
 setVisible(true);
 }
 public static void main(String s[])
 {
 new MyFrame1();
 }
}

```

**Output -**



### **There are two ways to display the component onto the Frame.**

1. Without Layout Manager
2. With Layout Manager

### **Layout Manager**

Layout manager are the predefined arrangements of the components onto the container. Each and every container such as Frame, Panel, and Applet etc. has the predefined Layout manager default.

**Java.awt** package provides the classes for the layout manager.

Instance of the classes represents the layout manager in the program **java.awt** package.

- BorderLayout
- GridLayout
- FlowLayout
- CardLayout
- etc.

**Without layout manager** the `setBounds()` method of the component class is use to set the x and y co-ordinates onto the components.

To remove the predefined layout from the container calling method will be used.

**Eg.**

```
fr.setLayout(null);
```

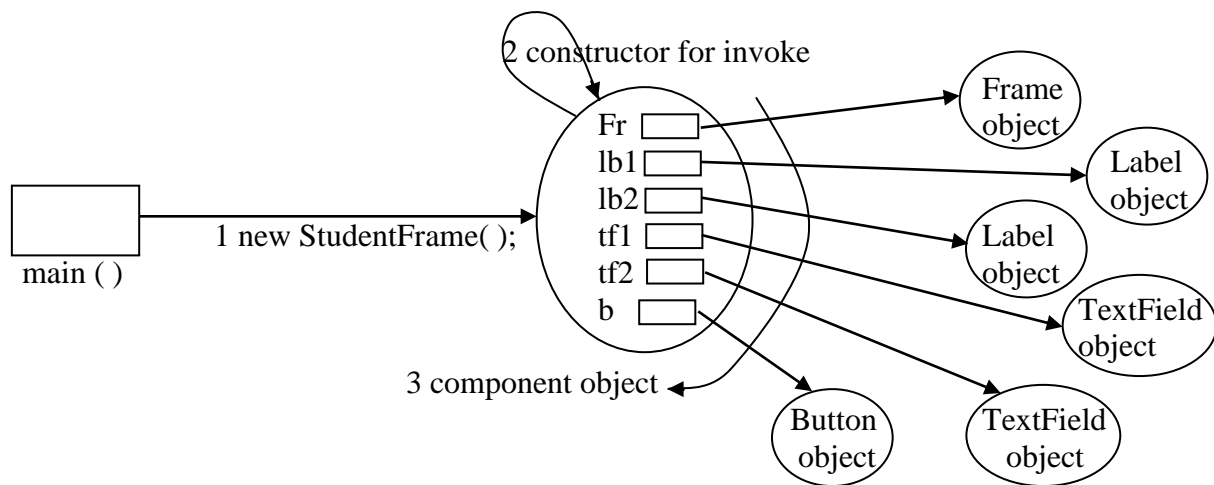
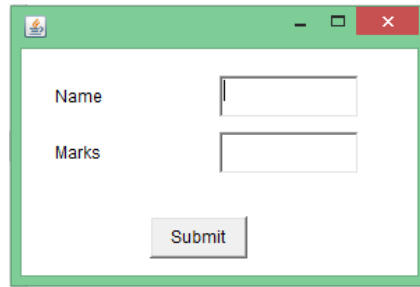
### **Steps to create and design the Frame without Layout Manager**

1. Create the object of frame or its child class.
2. Remove the predefined Layout.
3. Create the objects of components.
4. Set the bounds (coordinate and size) of the component.
5. Add the component onto the frame.
6. Set the size of the frame.
7. Makes the frame visible.

#### **//StudentFrame.java**

```
import java.awt.*;
class StudentFrame
{
 Frame fr;
 TextField tf1, tf2;
 Label lb1, lb2;
 Button b;
 StudentFrame()
 {
 fr = new Frame();
 fr.setLayout(null);
 tf1 = new TextField();
 tf2 = new TextField();
 lb1 = new Label("Name");
 lb2 = new Label("Marks");
 b = new Button("Submit");
 lb1.setBounds(30, 50, 100, 30);
 lb2.setBounds(30, 90, 100, 30);
 tf1.setBounds(150, 50, 100, 30);
 tf2.setBounds(150, 90, 100, 30);
 b.setBounds(100, 150, 70, 30);
 fr.add(lb1);
 fr.add(tf1);
 fr.add(lb2);
 fr.add(tf2);
 fr.add(b);
 fr.setSize(300, 200);
 fr.setVisible(true);
 }
 public static void main(String s[])
 {
 new StudentFrame();
 }
}
```

```
}
output-
```



```
//Admission.java
import java.awt.*;
class Admission
{
 Frame fr;
 TextField tf1, tf2, tf3, tf4;
 Label lb1, lb2, lb3, lb4, lb5, lb6;
 Button b1, b2;
 Admission()
 {
 fr = new Frame();
 fr.setLayout(null);
 tf1 = new TextField();
 tf2 = new TextField();
 tf3 = new TextField();
 tf4 = new TextField();
 lb1 = new Label("Admission Form");
 lb2 = new Label("=====");
 lb3 = new Label("Name");
 lb4 = new Label("Adress");
 lb5 = new Label("Phone");
 lb6 = new Label("E-mail");
 b1 = new Button("Submit");
 b2 = new Button("Reset");
 }
}
```

```

 lb1.setBounds(100,30,200,30);
 lb2.setBounds(100,60,200,20);
 lb3.setBounds(30,90,100,30);
 tf1.setBounds(150,90,100,30);
 lb4.setBounds(30,130,100,30);
 tf2.setBounds(150,130,100,30);
 lb5.setBounds(30,170,100,30);
 tf3.setBounds(150,170,100,30);
 lb6.setBounds(30,210,100,30);
 tf4.setBounds(150,210,100,30);
 b1.setBounds(50,250,100,40);
 b2.setBounds(170,250,100,40);
 fr.add(lb1); fr.add(lb2);
 fr.add(lb3); fr.add(tf1);
 fr.add(lb4); fr.add(tf2);
 fr.add(lb5); fr.add(tf4);
 fr.add(lb6); fr.add(tf3);
 fr.add(b1); fr.add(b2);
 fr.setSize(350,350);
 fr.setVisible(true);
 }
 public static void main(String s[])
 {
 new Admission();
 }
}

```

**Output -**

The screenshot shows a Java AWT window titled "Admission Form". The window has a green title bar with standard Windows controls (minimize, maximize, close). The main content area is white and contains the following elements:

- The text "Admission Form" centered at the top.
- A line of equals signs "===== " below the title.
- Four labels: "Name", "Adress", "Phone", and "E-mail", each followed by a text input field.
- Two buttons: "Submit" and "Reset" at the bottom.

**//Calculator.java**

```

import java.awt.Button;
import java.awt.Frame;
import java.awt.TextField;

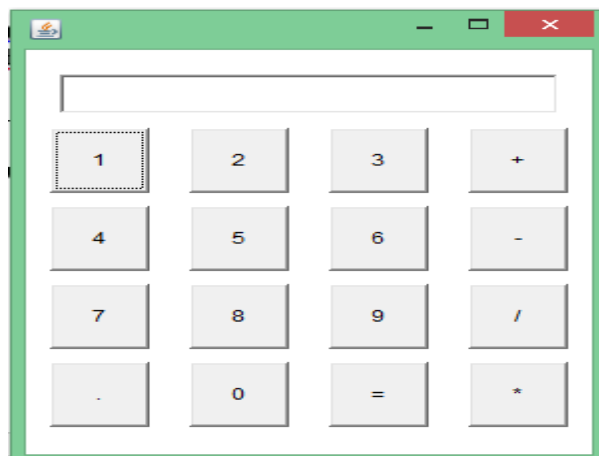
```

```

class Calculator
{
 Frame fr;
 int c=1;
 TextField tf;
 Button b;
 Calculator()
 {
 fr=new Frame();
 fr.setLayout(null);
 tf=new TextField();
 tf.setBounds(25,50,250,30);
 String value[]={"1","2","3","+","4","5","6","-","7","8","9","/",".", "0","=","*"};
 Button b[]=new Button[16];
 for(int i=0;i<16;i++)
 {
 b[i]=new Button(value[i]);
 fr.add(b[i]);
 }
 for(int i=0;i<16;i++)
 {
 fr.add(b[i]);
 }
 fr.add(tf);
 for(int i=0;i<4;i++)
 {
 c=c+60;
 b[i*4].setBounds(20,(30+c),50,50);
 b[i*4+1].setBounds(90,(30+c),50,50);
 b[i*4+2].setBounds(160,(30+c),50,50);
 b[i*4+3].setBounds(230,(30+c),50,50);
 }
 fr.setSize(300,350);
 fr.setVisible(true);
 }
 public static void main(String []args)
 {
 new Calculator();
 }
}

```

**Output -**



## Designing of the component through the layout manager

By default on to the frame “**BorderLayout**” is applicable.

By creating the object of any other layout manager class and passing their reference as the argument in the **setLayout()** method we can change the predefined the layout manager.

## FlowLayout

This is the default layout manager of the panel and their sub classes.

This layout manager arranges the component from left to right and top to down.

**setBounds()** method will not work in case of any layout manager caption over the components will designed their size.

**setPreferredSize()** method of the component class is use top set the size of the component explicitly.

**Ex: -**

```
tf1.setPreferredSize(new Dimension(100, 50));
```

↓                                      ↓  
(Reference of TexeField)      (class in the **java.awt** package)

## Constructor of the FlowLayout of the class

- **public FlowLayout( )**
- **public FlowLayout(Int align, int hgap, int vgap)**
- **public FlowLayout(int align)**

**Vgap** – Vertical gap

**Hgap** – Horizontal gap

Gap will be form in the pixels.

## There can be three type of alignment

1. center
2. left
3. right

## There are the static constants in the FlowLayout class for the alignment.

- **FlowLayout.CENTER**
- **FlowLayout.LEFT**
- **FlowLayout.Right**

**//StudentFrame.java**

```
import java.awt.*;
```

```
class StudentFrame1
```

```
{
```

```
 Frame fr;
```

```
 TextField tf1, tf2;
```

```
 Label lb1, lb2;
```

```
 Button b;
```

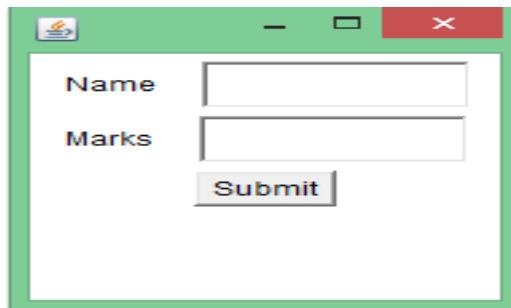
```
 Panel p;
```

```
 StudentFrame1()
```

```

{
 fr = new Frame();
 fr.setLayout(new FlowLayout());
 //fr.setLayout(null);
 p = new Panel();
 p.setLayout(new FlowLayout());
 tf1 = new TextField();
 tf2 = new TextField();
 lb1 = new Label("Name");
 lb2 = new Label("Marks");
 b = new Button("Submit");
 //no need to perform setBounds();
 tf1.setPreferredSize(new Dimension(100, 30));
 tf2.setPreferredSize(new Dimension(100,30));
 fr.add(lb1);
 fr.add(tf1);
 fr.add(lb2);
 fr.add(tf2);
 fr.add(b);
 fr.setSize(200, 300);
 fr.setVisible(true);
}
public static void main(String args[])
{
 new StudentFrame1();
}
}

```



There is the limitation of flowLayout is that as the size of container is change at the runtime the components will be rearranged.

For the flowLayout panel is the appropriate container as there border never be visible.

#### **//StudentFrame2.java**

```

import java.awt.*;
class StudentFrame2
{
 Frame fr;
 TextField tf1, tf2;
}

```



```

Label lb1, lb2;
Button b;
Panel p;
StudentFrame2()
{
 fr = new Frame();
 fr.setLayout(null);
 p = new Panel();
 p.setLayout(new FlowLayout());
 tf1 = new TextField();
 tf2 = new TextField();
 lb1 = new Label("Name");
 lb2 = new Label("Marks");
 b = new Button("Submit");
 tf1.setPreferredSize(new Dimension(100, 30));
 tf2.setPreferredSize(new Dimension(100, 30));
 p.add(lb1);
 p.add(tf1);
 p.add(lb2);
 p.add(tf2);
 p.add(b);
 p.setBounds(30, 50, 200, 300);
 fr.add(p);
 fr.setSize(400, 400);
 fr.setVisible(true);
}
public static void main(String s[])
{
 new StudentFrame2();
}
}

```



## **Event Handling**

- Changes in the state of components in the Event.
- Changes in the state of means changes in the look of components.
- Changes in the state occur by the user interaction or programmatically.

## **Java.awt.Event**

Event is nothing but changes in the state of GUI component is the Event.

In the **java.awt.Event** package there the number of Event class whose objects represents to the Event.

## **Event Class**

- **ActionEvent**
- **ItemEvent**
- **FocusEvent**
- **TextEvent**
- **MouseEvent**
- **WindowEvent**
- etc.

## **ActionEvent**

An instance of this class created when the button is clicked menu item is selected.

## **ItemEvent**

An instance of **ItemEvent** is created when the **CheckBox** or **RadioButton** gets clicked or item from the **ComboBox** selected.

## **FocusEvent**

By creating the object of this class application gets informed about gaining and lost of the keyboard focus.

## **KeyEvent**

An instance of this event class is created to intimate the program about the typing, pressing or releasing the key of the keyboard.

## **MouseEvent**

An object of **MouseEvent** class is created to intimate about the clicking, pressing or releasing, dragging etc. of the mouse.

## **WindowEvent**

An instance of **WindowEvent** is created when window is open, window is closed, activated, deactivated etc.

## **DelegateEvent Model**

In this model there are the two participants.

1. Source
2. Listener

## **Source**

Source is the component when is responsible for generating the event that means source triggers the event either due to the user interaction or programmatically.

## Listener

Listener is the object that has something for handling the event.

Listener are represented by the user define classes that has the **callback()** methods in which the response code for the event handling exist.

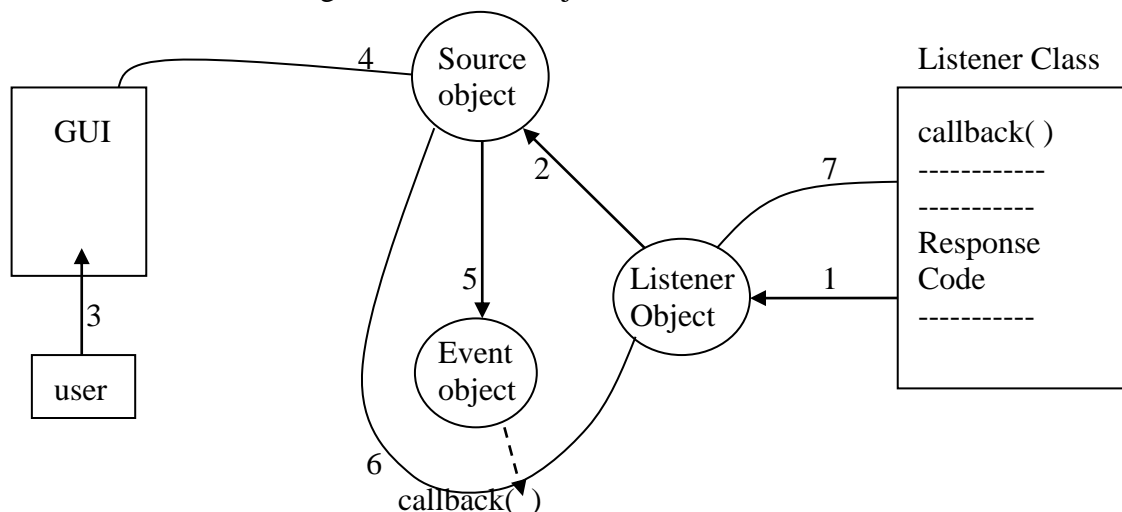
## Callback Method

**Callback** method are the methods whose name provided by the environment and the body of the method prepared by the programmer and then methods are called by the environment.

**main()** is the **callback** method.

[**Note:** - Listener object have to be registered with the source object in order to get the notification.]

As the event generated onto the source object creates the event object and invokes the **callback** method onto the registered listener object.



1. New object created
2. Listener object gets registered
3. User Interface with the view of source
4. New even object created
5. Source object gets notified
6. Execution of callback method
7. Callback() method invoked and the reference of event object is passed

Corresponding to each event there is one or more interfaces are available known as the listener interfaces.

Developers have to implement the listener corresponding to the event to which developer want to handle.

| Event Class        | Listener Interface  | Callback Method                                                                                                                                                                                                                                                                                                            |
|--------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ActionEvent</b> | ActionListener      | public void actionPerformed (ActionEvent e)                                                                                                                                                                                                                                                                                |
| <b>ItemEvent</b>   | ItemListener        | public void stateChanged(ItemEvent e)                                                                                                                                                                                                                                                                                      |
| <b>FocusEvent</b>  | FocusListener       | public void focusGained(FocusEvent e)<br>public void focusLost(FocusEvent e)                                                                                                                                                                                                                                               |
| <b>TextEvent</b>   | TextListener        | public void textValueChanged(TextEvent e)                                                                                                                                                                                                                                                                                  |
| <b>KeyEvent</b>    | KeyListener         | public void keyTyped(KeyEvent e)<br>public void keyPressed(KeyEvent e)<br>public void keyReleased(KeyEvent e)                                                                                                                                                                                                              |
| <b>MouseEvent</b>  | MouseListener       | public void mouseClicked(MouseEvent e)<br>public void mousePressed(MouseEvent e)<br>public void mouseReleased(MouseEvent e)<br>public void mouseEntered(MouseEvent e)<br>public void mouseExited(MouseEvent e)                                                                                                             |
|                    | MouseMotionListener | public void mouseMoved(MouseEvent e)<br>public void mouseDragged(MouseEvent e)                                                                                                                                                                                                                                             |
| <b>WindowEvent</b> | WindowListener      | public void windowOpened(WindowEvent e)<br>public void windowClosed(WindowEvent e)<br>public void windowClosing(WindowEvent e)<br>public void windowActivated(WindowEvent e)<br>public void windowDeactivated(WindowEvent e)<br>public void windowIconified(WindowEvent e)<br>public void windowDeiconified(WindowEvent e) |

//MyListener.java

```
import java.awt.event.*;
public class MyListener implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 System.out.println("Event is generated, now it can be handled");
 }
}
```

Source class provides the registration method to register the listener object.

### **Format of the registration method as follows**

- **public void add** \_\_\_\_\_ **( ref )**

**Ex:**

Registration method in the button (source) class

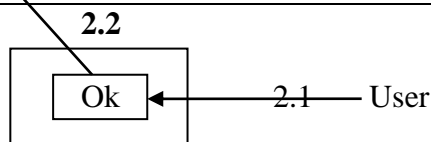
- **public void addActionListener(ActionListener al)**

Registration method in the Textfield (Source) class

- **public void addTextListener(TextListener tl)**

## Working flow of calling the callback method of Listener

| Java Library                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | User defined                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> class ActionEvent {     Component c;     ActionEvent(Component c)     {         this.c = c;     }     public Component getSource( )     {         Return(c);     }     ----- } Class Button {     ActionListener al;     Public void addActionListener(ActionListener al);     {         This.al=al;     }     &lt;Notification method&gt;     {         <u>2.3</u> ActionEvent e=new ActionEvent(this);         //Starting reference id of current source object.         <u>2.4</u> Al.actionPerformed(e);     } } </pre> | <pre> Class MyListener implements ActionListener {     Public void actionPerformed(ActionEvent e)     {         <u>2.5</u> -----e.getSource         -----     } } Class MyMain {     -----     Mymain( )     {         -----         1.1 Button b=new Button("ok");         MyListener ml=new MyListener( );         1.2 b.addActionListener(ml);     }     public static void main(String args[])     {         -----     } } </pre> |



1.1 Source object and Listener object created.

1.2 Listener object is registered with the source object.

2.1 **Button** is clicked by the user.

2.2 Notification method is invoked.

2.3 Event object is created and the ref id of the current source object is passed.

2.4 Callback method invoked and reference of event object is passed.

2.5 Execution of callback method started.

[**Note:** - One listener can be registered with the multiple sources.]

The event class object is use to identify the source to which the event is generated.

### **Method of the Event class**

**- public Component getSource()**

This method returns the reference of the source through which this event is generated.

**- Public String getActionCommand( )**

This method returns the caption of the source.

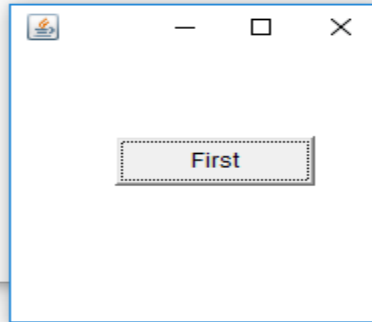
#### **//MyListener.java**

```
import java.awt.event.*;
public class MyListener implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 System.out.println("Event is generated, now it can be handled");
 }
}
```

#### **//EventTest.java**

```
import java.awt.*;
import java.awt.event.*;
class EventTest
{
 Frame fr;
 Button b1;
 EventTest()
 {
 fr = new Frame();
 fr.setLayout(null);
 b1 = new Button("First");
 MyListener list=new MyListener();
 b1.addActionListener(list);
 b1.setBounds(60, 80, 100, 30);
 fr.add(b1);
 fr.setSize(200, 200);
 fr.setVisible(true);
 }
 public static void main(String []args)
 {
 new EventTest();
 }
}
```

```
E:\JAVA\Programs\Core Java Program\AWT\EventTest>javac MyListener.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest>javac EventTest.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest>java EventTest
Event is generated, now it can be handled
```



### **Changes in the MyListener class**

**//MyListener.java**

```
import java.awt.event.*;
public class MyListener implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 String str = e.getActionCommand();
 if(str.equals("First"));
 {
 System.out.println("First Button is clicked");
 }
 if(str.equals("Second"));
 {
 System.out.println("Second Button is clicked");
 }
 }
}
```

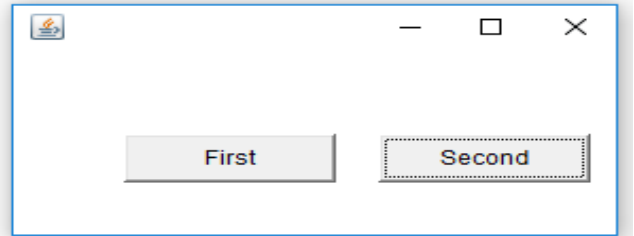
**//EventTest.java**

```
import java.awt.*;
import java.awt.event.*;
class EventTest
{
 Frame fr;
 Button b1,b2;
 EventTest()
 {
 fr = new Frame();
 fr.setLayout(null);
 b1 = new Button("First");
 b2 = new Button("Second");
 MyListener list=new MyListener();
 }
}
```

```

 b1.addActionListener(list);
 b2.addActionListener(list);
 b1.setBounds(60, 80, 100, 30);
 b2.setBounds(180, 80, 100, 30);
 fr.add(b1);
 fr.add(b2);
 fr.setSize(300, 300);
 fr.setVisible(true);
 }
 public static void main(String []args)
 {
 new EventTest();
 }
}
E:\JAVA\Programs\Core Java Program\AWT\EventTest1>javac MyListener.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest1>javac EventTest.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest1>java EventTest
First Button is clicked
Second Button is clicked

```



### **Changes in the EventTest.java**

```

//EventTest.java
import java.awt.*;
import java.awt.event.*;
class EventTest
{
 Frame fr;
 Button b1,b2;
 TextField tf;
 EventTest()
 {
 fr = new Frame();
 fr.setLayout(null);
 b1 = new Button("First");
 b2 = new Button("Second");
 tf = new TextField();
 MyListener listener=new MyListener();
 b1.addActionListener(listener);
 b2.addActionListener(listener);
 b1.setBounds(60, 80, 100, 30);
 b2.setBounds(180, 80, 100, 30);
 tf.setBounds(60,150,150,30);
 }
}

```



```

 fr.add(b1);
 fr.add(b2);
 fr.add(tf);
 fr.setSize(300, 300);
 fr.setVisible(true);
 }
 public static void main(String []args)
 {
 new EventTest();
 }
}

```

#### //MyListener.java

```

import java.awt.event.*;
public class MyListener implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
 String str = e.getActionCommand();
 EventTest et = new EventTest();
 if(str.equals("First"));
 {
 et.tf.setText("First Button is clicked");
 }
 if(str.equals("Second"));
 {
 et.tf.setText("Second Button is clicked");
 }
 }
}

```

```

E:\JAVA\Programs\Core Java Program\AWT\EventTest3>javac EventTest.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest3>java EventTest

```



**[Note: -** In the event handling there is the always be the common requirement to access the callback methods, so it recommended to make the designer class also as the listener class.]

## **Changes in the EventTest.java**

**//EventTest.java**

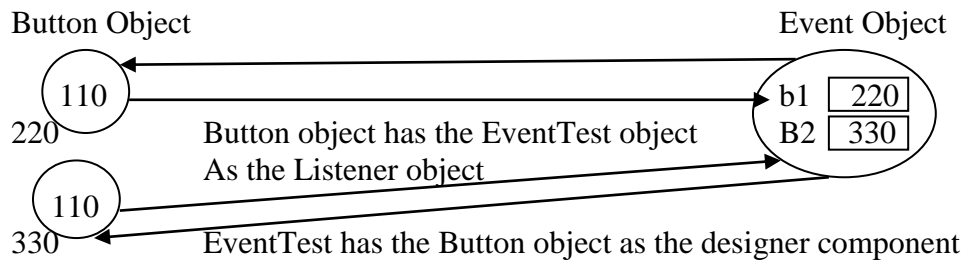
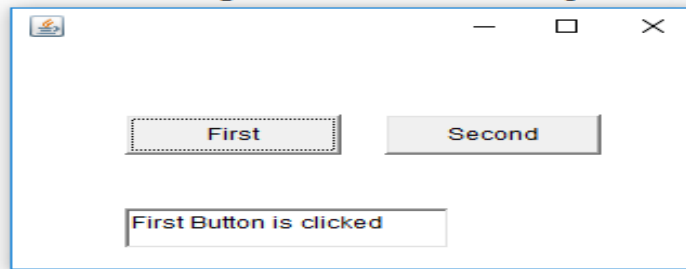
import java.awt.\*;

import java.awt.event.\*;

class EventTest implements ActionListener

```
{
 Frame fr;
 Button b1,b2;
 TextField tf;
 EventTest()
 {
 fr = new Frame();
 fr.setLayout(null);
 b1 = new Button("First");
 b2 = new Button("Second");
 tf = new TextField();
 b1.addActionListener(this);
 b2.addActionListener(this);
 //Registration of the listener with source
 b1.setBounds(60, 80, 100, 30);
 b2.setBounds(180, 80, 100, 30);
 tf.setBounds(60,150,150,30);
 fr.add(b1);
 fr.add(b2);
 fr.add(tf);
 fr.setSize(300, 300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 tf.setText("First Button is clicked");
 }
 if(e.getSource()==b2)
 {
 tf.setText("Second Button is clicked");
 }
 }
 public static void main(String []args)
 {
 new EventTest();
 }
}
```

```
E:\JAVA\Programs\Core Java Program\AWT\EventTest3>javac EventTest.java
E:\JAVA\Programs\Core Java Program\AWT\EventTest3>java EventTest
```



## Assignment

**//Calculator.java**

//Calculato.java

import java.awt.\*;

import java.awt.event.\*;

class Calculator //implements ActionListener

{

Frame fr;

Label num1,num2,result;

TextField txtn1,txtn2,txtr;

Button add,sub,mul,div;

Calculator()

{

fr=new Frame();

fr.setLayout(null);

txtn1=new TextField( );

txtn2=new TextField( );

txtr=new TextField( );

num1=new Label("First Number");

num2=new Label("Second Number");

result=new Label("Result");

add=new Button("Add");

sub=new Button("Sub");

mul=new Button("Mul");

div=new Button("Div");

add.addActionListener(this);

sub.addActionListener(this);

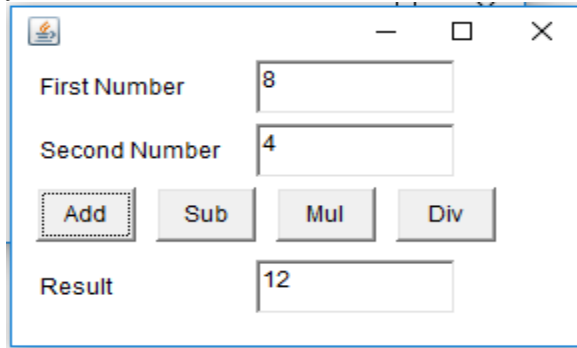
mul.addActionListener(this);

```

 div.addActionListener(this);
 num1.setBounds(20,30,100,30);
 txtn1.setBounds(130,30,100,30);
 num2.setBounds(20,60,100,30);
 txtn2.setBounds(130,60,100,30);
 result.setBounds(20,140,100,30);
 txtr.setBounds(130,140,100,30);
 add.setBounds(20,100,50,30);
 sub.setBounds(80,100,50,30);
 mul.setBounds(140,100,50,30);
 div.setBounds(200,100,50,30);
 fr.add(num1); fr.add(num2); fr.add(result);
 fr.add(txtn1); fr.add(txtn2); fr.add(txtr);
 fr.add(add); fr.add(sub); fr.add(mul); fr.add(div);
 fr.setSize(300,250);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 int x,y,res;
 x=Integer.parseInt(txtn1.getText());
 y=Integer.parseInt(txtn2.getText());
 if(e.getSource()==add)
 {
 res=x+y;
 txtr.setText(""+res);
 }
 if(e.getSource()==sub)
 {
 res=x-y;
 txtr.setText(""+res);
 }
 if(e.getSource()==mul)
 {
 res=x*y;
 txtr.setText(""+res);
 }
 if(e.getSource()==div)
 {
 res=x/y;
 txtr.setText(""+res);
 }
 }
 public static void main(String []args)
 {
 new Calculator();
 }

```

}



## **Implementing the window listener**

**//WindowEventTest.java**

import java.awt.\*;

import java.awt.event.\*;

class WindowEventTest implements WindowListener

{

Frame fr;

TextField tf;

WindowEventTest()

{

fr=new Frame();

fr.setLayout(null);

tf=new TextField();

tf.setBounds(30,50,200,30);

fr.add(tf);

fr.setSize(300,300);

fr.setVisible(true);

fr.addWindowListener(this);

}

public void windowOpened(WindowEvent e)

{

System.out.println("Window Open");

}

public void windowClosed(WindowEvent e)

{

System.out.println("Window is closed");

fr.dispose();

}

public void windowClosing(WindowEvent e)

{

System.out.println("Window close");

}

public void windowActivated(WindowEvent e)

{

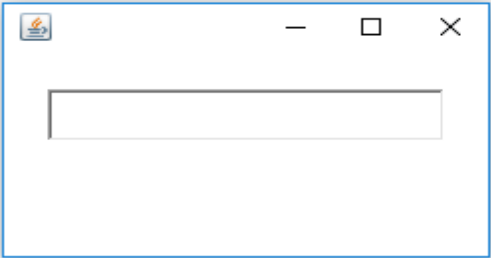
System.out.println("Window is activated");

}

```

public void windowDeactivated(WindowEvent e)
{
 System.out.println("Window is deactivated");
}
public void windowIconified(WindowEvent e)
{
 System.out.println("Window is minimized");
}
public void windowDeiconified(WindowEvent e)
{
 System.out.println("Window is maximized");
}
public static void main(String []args)
{
 new WindowEventTest();
}
}
E:\JAVA\Programs\Core Java Program\AWT\WindowEvent>javac WindowEventTest.java
E:\JAVA\Programs\Core Java Program\AWT\WindowEvent>java WindowEventTest
Window is activated
Window is deactivated
Window is activated
Window is deactivated
Window is activated

```



## Adaptor classes

In the **java.awt.event** package there are the adaptor classes corresponding to the entire listener interface that have more than one callback methods adaptor class implements to the listener interface and provides the default implementation to all the callback methods.

In the adaptor classes there are the body of all the callback body we can prepare the listener class also by extending the adaptor classes.

**Ex:**

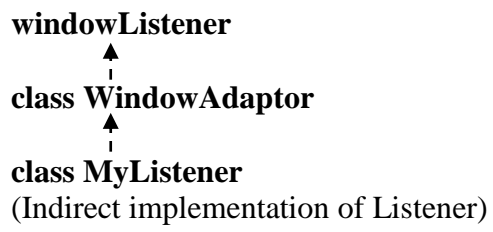
- **WindowAdaptor**
- **MouseAdaptor**
- **MouseMotionAdaptor**
- **KeyAdaptor**
- **FocusAdaptor**
- **Etc.**

**WindowListener**



**class MyListener**

(Direct implementation of listener Interface)



# SWING

It is extension of AWT, it was introduced in JDK 1.2 as the part of java foundation classes (JFC).

JFC means three things.

**JFC = AWT + Swing + java 2D**

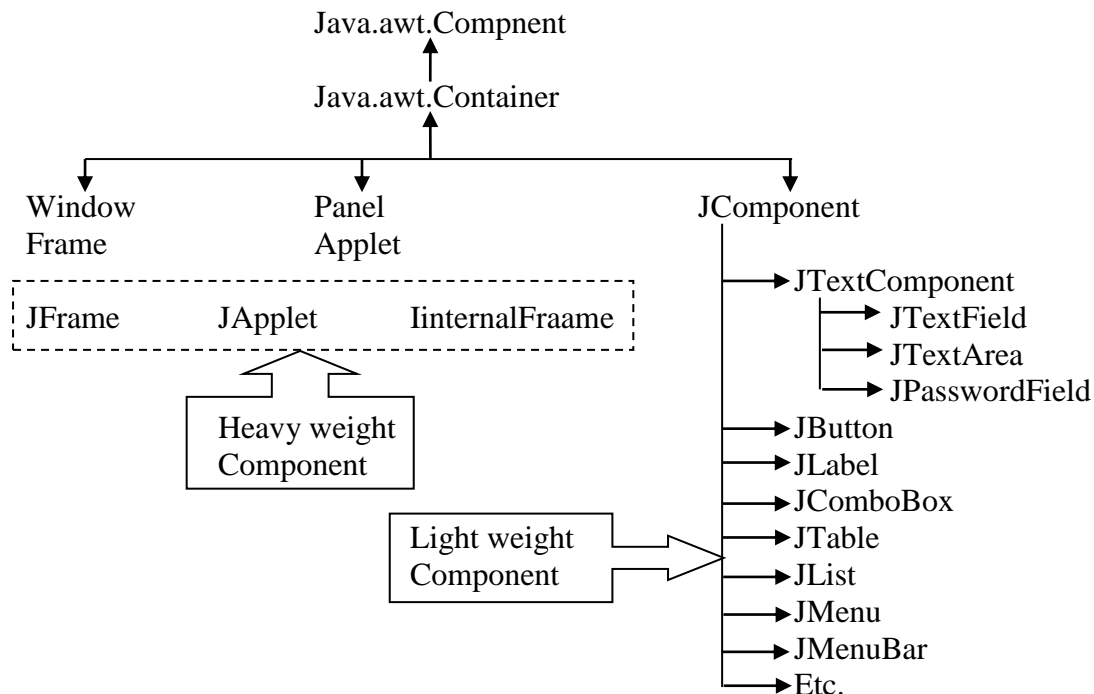
## Differences between the AWT and Swing (Advantages of the Swing over AWT)

1. Swing component are light weighted component with aspect of memory and execution time while AWT components are the heavy weight components.
2. At runtime AWT components take their definition from native Operating System (C & C++) while the Swing component purely developed in java.
3. Look and feel of AWT component very across the platform while the look and feel of swing component remains same across Operating System.
4. AWT provides advance components while the AWT provides basic components.
5. Swing utilized the MVC architecture while the AWT is just only the toolkit.

MVC →

## java.swing Package

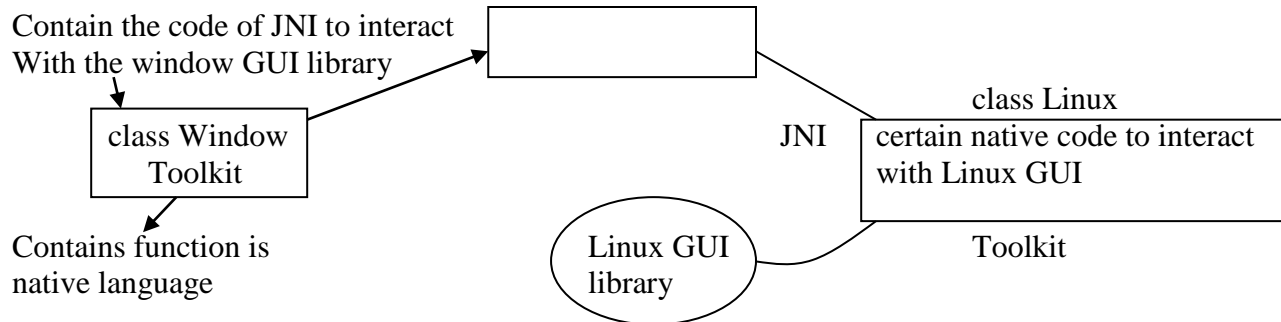
Provides API of Swing Hierarchy of Swing components



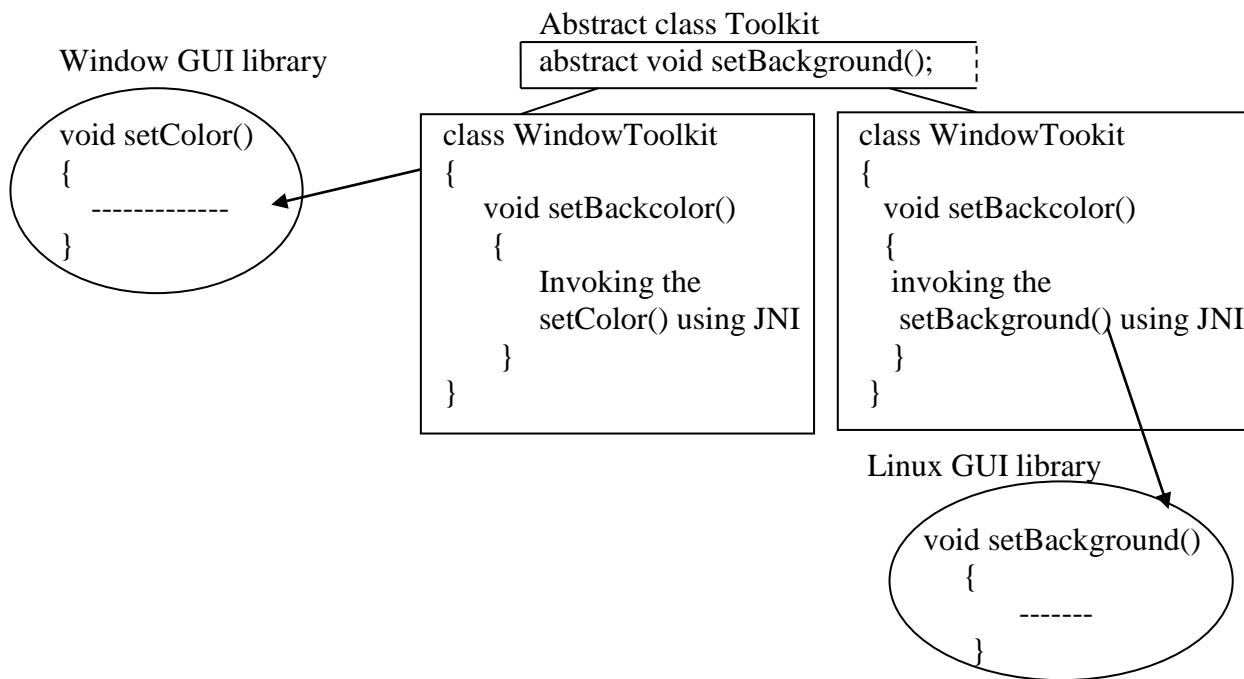
## **AWT stands for abstract?**

In the AWT library there the abstract class named toolkit at the runtime there implemented class created by the JRE according to the Operating System that implemented class will be the proxy class.

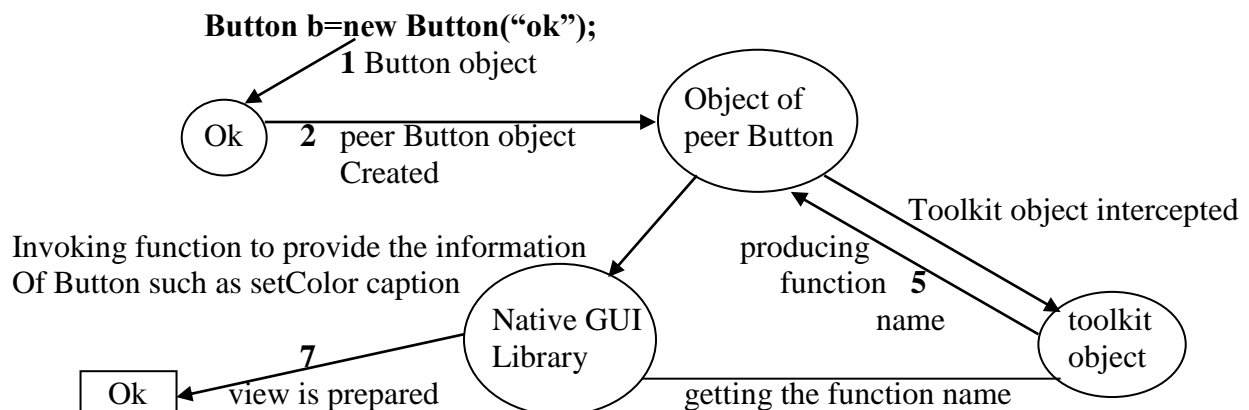




Let suppose window GUI library has the method with the name getColor(), setColor() and Linux GUI library has the method setBackground().



[Note: - Corresponding to each component of the AWT there is the extra class known as the peer class the use of peer class is to interact with GUI library of the Operating System through the toolkit class.]



## MVC (Model View and Controller)

This is the design according to that the whole functionality is divided into the three loosely coupled layer.

1. Model
2. View
3. Controller

### Model

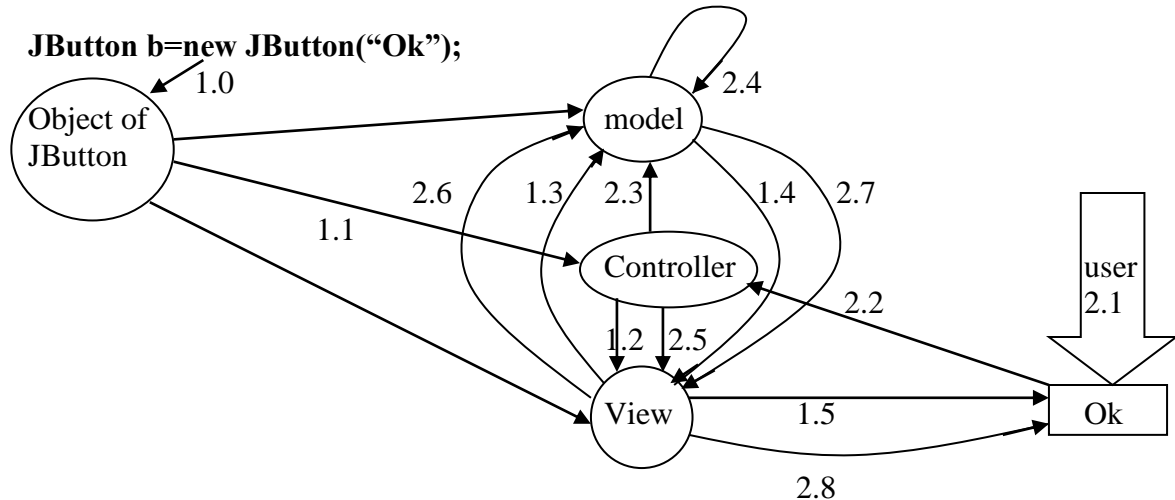
Contains the data use to prepare the view.

### View

It is responsible together the view by using the data provided by the model.

### Controller

It is responsible to interact with the view and model layer that means the controller is responsible to coordinate between the model views.



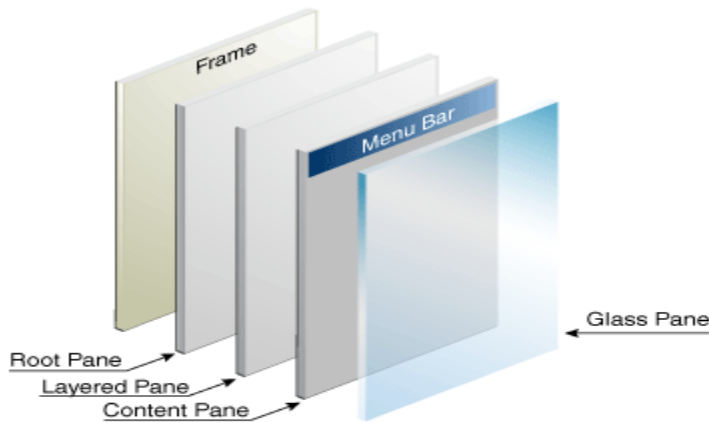
- 1.0 JButton object is created.
- 1.1 Model view and controller object are created.
- 1.2 View just notified to generate the view.
- 1.3 Model is requested to provide the data.
- 1.4 Data is provided to the view layer.
- 1.5 View is generated.
- 2.1 Button is clicked.
- 2.2 Controller is intercepted.
- 2.3 Model is notified to change the data.
- 2.4 Data is changed by the model.
- 2.5 View is notified to update the view.
- 2.6 Model is requested for new data.
- 2.7 Updated data is provided.
- 2.8 View gets updated.

## **javax.swing.JFrame**

### **JFrame**

It is container available in the swing API all the containers in the swing are heavy weighted that means they take their definition from the native Operating System at the runtime.

To retain the layered architecture on the swing is available.



### **Root Pane**

Provides the base that will interact with the native GUI library

### **Layered Pane**

Provides the depth

### **Content Pane**

All the component always be added on to the content pane.

### **Glass Pane**

Provides the facility of the drag and drop

[N0ote: - Up to the JDK 1.4 we can't directly add the components on to the JFrame rather we have to first of all get the content pane then we have to add the components onto the content pane.]

### **Up to the JDK 1.4**

```
Container c=fr.getContainer();
c.add(b);
```

To add any Button

### **JDK 1.5**

```
Fr.add(b);
```

To add Button indirectly onto the Content Pane

### **Method of JFrame**

- `public void setDefaultCloseOperation(int closingMade)`



```

 //relative path
new JLabel(new ImageIcon("d:\\temp\\abc.jpg"));
 //abstract path

```

- **public JLabel(String text, Icon I, int alignment)**

### **Method of JLabel**

- **public void setText(String str)**

It will set the text in the label

- **public String getText()**

This method will return the text of the JLabel

- **public void setItem(Icon i)**

This method is use to set the icon onto the JLabel.

- **public Icon getIcon()**

It will returns the reference of the Icon of the JLabel

- **public int getHorizontalAlignment()**

This method returns the horizontal alignment of JLabel which is currently set.

- **public void setHorizontalAlignment(int alignment)**

This method will set the horizontal alignment.

### **//JLabelTest.java**

```

import javax.swing.*;
import java.awt.event.*;
public class JLabelTest implements ActionListener
{
 JFrame fr;
 JLabel lb1;
 JButton b1, b2, b3;
 JLabelTest()
 {
 fr = new JFrame();
 fr.setLayout(null);
 lb1 = new JLabel();
 b1 = new JButton("First");
 b2 = new JButton("Second");
 b3 = new JButton("Third");
 lb1.setBounds(20,20,300,300);
 b1.setBounds(20,330,100,30);
 b2.setBounds(130,330,100,30);
 b3.setBounds(240,330,100,30);
 b1.addActionListener(this);
 b2.addActionListener(this);

```

```

 b3.addActionListener(this);
 fr.add(lb1); fr.add(b1);
 fr.add(b2); fr.add(b3);
 fr.setSize(400,450);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 lb1.setIcon(new ImageIcon("Apple(2).jpg"));
 }
 if(e.getSource()==b2)
 {
 lb1.setIcon(new ImageIcon("Apple(2)ja.jpg"));
 }
 if(e.getSource()==b3)
 {
 lb1.setIcon(new ImageIcon("Apple-Logo.jpg"));
 }
 }
 public static void main(String []args)
 {
 new JLabelTest();
 }
}

```



## **JTextField**

This component use to take the input from the end user this is to accept only the single line of text.

### **Constructor**

- **public JTextField()**
- **public JTextField(String str)**

- **public JTextField(int noOfColumns)**  
This constructor only be use in case of layout manager.
- **public JTextField(String s, int row, int column)**

### **Methods of JTextField**

- **public void setText(String str)**
- **public String getText()**
- **public String getSelectedText()**
- **public void selectAll()**

It is use to select all the TextField.

- **public void setSelset(int startIndex, int endIndex)**

This method will select the text from the specified start index to the specific end index.

- **public void copy()**

This method will copy the selected text and placed onto the clipboard.

- **public void paste()**

This method will paste the text into this text firdl at the current position of the cursor from the clipboard.

- **public void cut()**

This method will cut the selectied text and placed in the keyboard.

- **public void setFont(Font f)**

This method is use to set the font.

- **public void setdBackground(Color c)**

It is use to set background color

- **public void setForground(Color c)**

It is use to set forground color.

- **public void setSelectionTextColor(Color c)**

- **public void setSelectionColor(Color c)**

### **java.awt.Font class**

As instance of this is use to set the font

#### **Constructor**

- **public Font(String style, int font, int size)**

Ex:-

```
tf.setFont(new Font("ArialBalck", Font.BOLD, 25));
```

**Font.BOLD**  
**Font.ITALIC**  
**Font.PLAIN**

### **java.awt.Color class**

This class instance is use to represent the calor.

#### **Constructor**

- **public Color(int r, int g, int b)**  
value 0to255, 0to255, 0to255

**Ex: - new Color(0,0,255);            //pure blue**

**[Note: - In the color class there are the color static color objects are exists.]**

```
class Color
{
 public static final ColorRED=new Color(255,0,0);
 public static final Color red=new Color(255,0,0);
 public static final ColorGREEN=new Color(0,255,0);
}
```

**Some more color static object**

**Color.YELLOW**  
**Color.GRAY**  
**Color.PINK**  
**Color.ORANGE**  
**Color.etc**

**Ex:    tf.setBackground(new Color(255,0,0)); or**  
**tf.setBackgroun(Color.RED);**

### **JTextArea**

It is also the import component through this component multiple lines of text can be entered.

Constructor and methods are same as the JTextField.

#### **Additional Constructor**

- **public JTextArea(int noOfRows, int noOfColumns)**  
This constructor always be used in case of layout manager.

#### **Additional Methods**

- **public void append(String str)**  
This method is use to append the specified text in text of this TextArea.
- **public void setLineWrap(boolean b)**  
If argument value is true then the automatically horizontal scrollbar will be not append.



[**Note:** - In the swing there is no facility of scrollbar by default rather we have to add the scrollbar explicitly.]

### **javax.swing.JScrollPane class**

It is the container like component it has the horizontal scrollbar implicitly.

If we want to apply the scrollbar's on any component then we have to add that component onto the scroll pane.

#### **Constructor**

- **public JScrollPane(Component c)**
- **public JScrollPane(Component c, int hspolicy, int vsbpolicy)**

#### **Static constructors for policy**

**JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEED**

**JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS**

**JScrollPane.VERTICAL\_SCROLLBAR\_NEVER**

**JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEED**

**JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS**

**JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER**

#### **Method of JTextArea**

- **public void replaceSelection(String str)**

This method will replace the selected text with the specified text.

#### **//TestArea.java**

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.*;
```

```
class TextAreaTest implements ActionListener
```

```
{
```

```
 JFrame fr;
```

```
 JTextArea ta;
```

```
 JButton b;
```

```
 JScrollPane pane;
```

```
 TextAreaTest()
```

```
{
```

```
 fr=new JFrame();
```

```
 fr.setLayout(null);
```

```
 ta=new JTextArea(); //step 1
```

```
 pane=new
```

```
JScrollPane(ta,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS); //step 2
```

```
 pane.setBounds(30,30,200,200);
```

```
 b=new JButton("Click");
```

```
 b.setBounds(30,270,100,50);
```

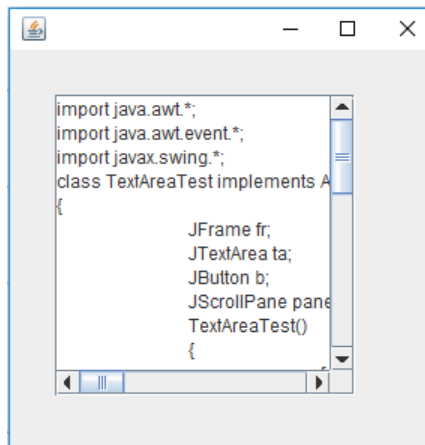
```
 fr.add(pane);
```

```
 fr.add(b);
```

```

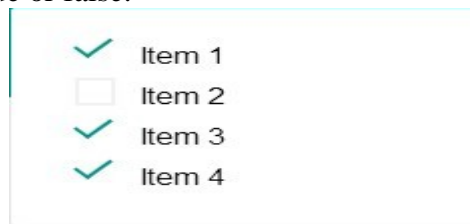
 b.addActionListener(this);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 ta.requestFocusInWindow();
 ta.select(3,5);
 }
 public static void main(String s[])
 {
 new TextAreaTest();
 }
}

```



## JCheckBox

This is also the input type component but it always takes the value in the form of Boolean value either true or false.



## Constructor

- **public JCheckBox(String str)**

It will make the checkbox with specified text

**Ex:**

**new JCheckBox("Java");**



- **public JCheckBox(String str, Icon i)**

It will add Image

### **Method of JCheckBox**

- **public void setSelected(boolean b)**

This method is use to make the checkbox selected.

- **public void getText()**

- **public void setSelectedIcon(Icon i)**

This will set the icon that will display when the checkbox is selected on the checkbox the action event and icon event both can be generated.

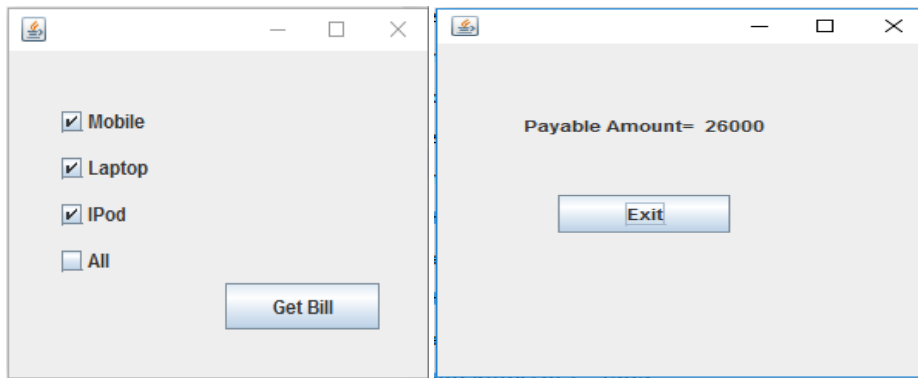
### **//JCheckBoxTest.java**

```
import java.awt.event.*;
import javax.swing.*;
class JCheckBoxTest implements ActionListener
{
 JFrame fr1,fr2;
 JCheckBox cb1,cb2,cb3,cb4;
 JLabel lb;
 JButton b1,b2;
 JCheckBoxTest()
 {
 fr1=new JFrame();
 fr2=new JFrame();
 fr1.setLayout(null);
 fr2.setLayout(null);
 cb1=new JCheckBox("Mobile");
 cb2=new JCheckBox("Laptop");
 cb3=new JCheckBox("IPod");
 cb4=new JCheckBox("All");
 lb=new JLabel();
 b1=new JButton("Get Bill");
 b2=new JButton("Exit");
 cb1.setBounds(30,30,100,30);
 cb2.setBounds(30,60,100,30);
 cb3.setBounds(30,90,100,30);
 cb4.setBounds(30,120,100,30);
 b1.setBounds(140,150,100,30);
 lb.setBounds(50,50,150,30);
 b2.setBounds(70,120,100,30);
 fr1.add(cb1);fr1.add(cb2);fr1.add(cb3);fr1.add(cb4); fr1.add(b1);
 fr2.add(lb) ; fr2.add(b2);
 cb4.addActionListener(this);
 b1.addActionListener(this);
 b2.addActionListener(this);
 fr1.setSize(300,300);
```

```

 fr2.setSize(250,250);
 fr1.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==cb4)
 {
 if(cb4.isSelected())
 {
 cb1.setSelected(true);
 cb2.setSelected(true);
 cb3.setSelected(true);
 }
 else
 {
 cb1.setSelected(false);
 cb2.setSelected(false);
 cb3.setSelected(false);
 }
 }
 if(e.getSource()==b1)
 {
 int amt=0;
 if(cb1.isSelected())
 amt+=5000;
 if(cb2.isSelected())
 amt+=20000;
 if(cb3.isSelected())
 amt+=1000;
 lb.setText("Payable Amount= "+amt);
 fr2.setVisible(true);
 }
 if(e.getSource()==b2)
 System.exit(0);
 }
 public static void main(String s[])
 {
 new JCheckBoxTest();
 }
}

```



## **JRadioButton**

This also the input component uses to take the input from the end user in the form of Boolean value.

JRadioButton are same JCheckBox but it can grouped.  
Constructor and method are same as JCheckBox

## **ButtonGroup class**

This class os use to group the radio buttons.

Ex:

```
JRadioButton rb1,rb2,rb3;
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);
bg.add(rb2);
bg.add(rb3);
br1.setBounds(-,-,-,-);
br2.setBounds(-,-,-,-);
br3.setBounds(-,-,-,-);
fr.add(rb1);
fr.add(rb2);
fr.add(rb3);
```

## **JComboBox**

This is also the input component that provides of list of values in front of end-user.

### **Constructor**

- **public JComboBox()**
- **public JComboBox(Object []values)**

Ex: `String city[]={ "Noida","Agra","Kanpur","Delhi" };  
JComboBox cb=new JComboBox(city);`

- **public JComboBox(Vector v)**

This constructor will take the collection of object that contains the value the ComboBox.

- **public JComboBox(ComboBoxModel model)**

ComboBoxModel is the interface and its implemented class is the DefaultComboBoxModel.

An instance of the ComboBoxModel represent to the values for the ComboBox.

**Ex: -**

```
DefaultComboBoxModel model=new DefaultComboBoxModel();
model.addElement("Noide");
model.addElement("Delhi");
model.addElement("Agra");
JComboBox cb=new JComboBox(model);
```

### **Method of JComboBox**

- **public Object getSelectedItem()**

This method will return the object selected on the comboBox.

- **public int getSelectedIndex()**

This method will return the index of selected item.

- **public void setSelectedIndex()**

This method will select the specified item.

- **public void setSelectedItem(Object item)**

- **public void setModel(ComboBoxModel model)**

This method will set the model.

- **public void setVisibleRowCount(int numberOfRows)**

This method will set the number of rows in the ComboBox that will be visible when the ComboBox gets open.

- **public void addItem(Object ob)**

This method will add the specified item in the ComboBox.

- **public void removeItem(Object ob)**

This method will remove the specified item from the comboBox.

### **Methods of DefaultComboBoxModel**

- **public void addElement(Object element)**

- **public void removeElementAt(int index)**

- **public void removeAllElement()**

- **public int getSize()**

This method returns the number of element contended by the ComboBox.

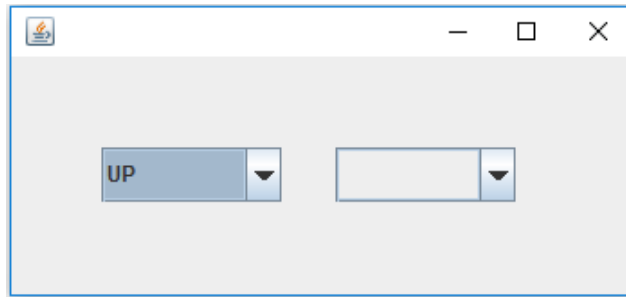
**//ComboBoxTest.java**

```
import javax.swing.*;
import java.awt.event.*;
```

```

class ComboBoxTest implements ActionListener
{
 JFrame fr;
 JComboBox cb1,cb2;
 DefaultComboBoxModel model1,model2,model3;
 ComboBoxTest()
 {
 fr=new JFrame();
 fr.setLayout(null);
 String city[]={ "Noida","Agra","Kanpur"};
 model1=new DefaultComboBoxModel(city);
 model2=new DefaultComboBoxModel();
 model2.addElement("Bhopal");
 model2.addElement("Jabalpur");
 model2.addElement("Indore");
 model3=new DefaultComboBoxModel();
 model3.addElement("Jaipur");
 model3.addElement("Ajmer");
 model3.addElement("Udaypur");
 String state[]={ "UP" , "MP" , "Raj" };
 cb1=new JComboBox(state);
 cb2=new JComboBox();
 cb1.setBounds(50,50,100,30);
 cb2.setBounds(180,50,100,30);
 fr.add(cb1);fr.add(cb2);
 cb1.addActionListener(this);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==cb1)
 {
 String st=(String)cb1.getSelectedItem();
 if(st.equals("UP"))
 cb2.setModel(model1);
 if(st.equals("MP"))
 cb2.setModel(model2);
 if(st.equals("Raj"))
 cb2.setModel(model3);
 }
 }
 public static void main(String s[])
 {
 new ComboBoxTest();
 }
}

```



## **JList**

### **java.swing.JList class**

JList is also input type component like comboBox in the JList multiple item can be view on to the GUI.

At a time user can select single item or the multiple items.

Constructor and Methods are same as the JComboBox

### **Additional Methods**

- **public void setSelectionMode(int mode)**

This method will set the selectionMode of the element on the ComboBox.

### **ListSelectionModel Interface**

ListSelectionModel interface provides the static constants for selection mode.

**ListSelectionModel.SINGLE\_SELECTION**

**ListSelectionModel.SINGLE\_INTERVAL\_SELECTION**

**ListSelectionModel.MULTIPLE\_INTERVAL\_SELECTION**

**Ex: -**

```
JList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION)
```

- **public Object getSelectionaValue()**
- **public Object[] getSelectedValues()**
- **public int getSelectedIndex()**
- **public int[] getSelectedIndexes ()**

### **javax.swing.event.ListSelectionEvent class**

This event is generated when the any list item is selected.

### **ListSelectionListener**

#### **Callback Method**

- **public void valueChanged(ListSelectionListener e)**

**//ListTest.java**

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.event.*;
```

```
class ListTest extends MouseAdapter implements ListSelectionListener , ActionListener
```



```

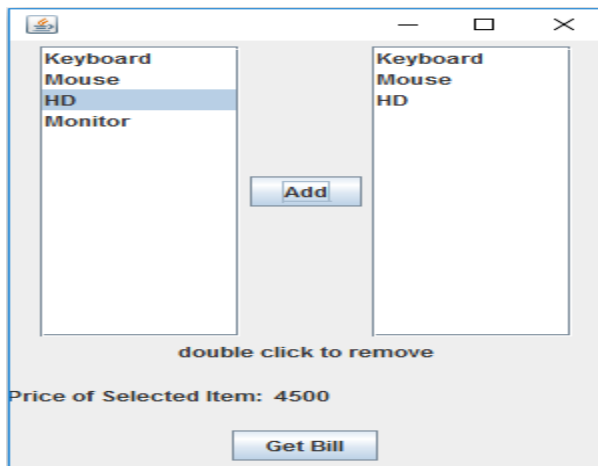
{
 JFrame fr1,fr2;
 JList list1,list2;
 JScrollPane pane1,pane2;
 DefaultListModel model1,model2;
 JButton badd,bget;
 JLabel lb1,lb2,lb3;
 int price[]={ 1500,800,4500,5000};
 String values[];
 ListTest()
 {
 fr1=new JFrame();
 fr2=new JFrame();
 fr1.setLayout(new FlowLayout());
 fr2.setLayout(new FlowLayout());
 values=new String[]{"Keyboard","Mouse","HD","Monitor"};
 model1=new DefaultListModel();
 for(int i=0 ; i<values.length ; i++)
 model1.addElement(values[i]);
 list1=new JList(model1);
 pane1=new JScrollPane(list1);
 model2=new DefaultListModel();
 list2 = new JList(model2);
 pane2=new JScrollPane(list2);
 pane1.setPreferredSize(new Dimension(100,250));
 pane2.setPreferredSize(new Dimension(100,250));
 badd=new JButton("Add");
 bget=new JButton("Get Bill");
 lb1=new JLabel("double click to remove");
 lb2=new JLabel();
 lb3=new JLabel();
 lb2.setPreferredSize(new Dimension(300,50));
 lb3.setPreferredSize(new Dimension(300,50));
 fr1.add(pane1);
 fr1.add(badd);
 fr1.add(pane2);
 fr1.add(lb1);
 fr1.add(lb2);
 fr1.add(bget);
 badd.addActionListener(this);
 bget.addActionListener(this);
 list1.addListSelectionListener(this);
 list2.addMouseListener(this);
 fr2.add(lb3);
 fr1.setSize(400,400);
 fr1.setVisible(true);
 fr2.setSize(300,300);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==badd)

```

```

 {
 Object values[]=list1.getSelectedValues();
 for(int i=0 ; i<values.length ;i++)
 {
 model2.addElement(values[i]);
 }
 }
 if(e.getSource()==bget)
 {
 int i=0,bill=0;
 while(true)
 {
 try
 {
 String str=(String)model2.getElementAt(i);
 int index=model1.indexOf(str);
 bill=bill + price[index];
 i++;
 }
 catch(ArrayIndexOutOfBoundsException ee)
 {
 break;
 }
 }
 lb3.setText("Total Paybale amount: "+bill);
 fr2.setVisible(true);
 }
 }
 public void valueChanged(ListSelectionEvent e)
 {
 int i=list1.getSelectedIndex();
 lb2.setText("Price of Selected Item: "+price[i]);
 }
 public void mouseClicked(MouseEvent e)
 {
 if(e.getSource()==list2)
 {
 System.out.println("mouse clicked");
 if(e.getClickCount()==2)
 {
 model2.removeElement(list2.getSelectedValue());
 }
 }
 }
 public static void main(String s[])
 {
 new ListTest();
 }
}

```



## **JTable**

JTable is use to show the report on to the GUI in the tabular format that means in form of row and columns.

## **Constructor**

- **public JTable(int noOfRows, int noOfColumn)**
- **public JTable(TableModel model)**
- **public JTable(String [][]values, String []heading)**

## **Methods**

- **publicObject getValueAt(int rowNo, int colon)**
- **public void setValueAt(String value, int rowNo, int colNo)**
- **public void setModel(TableModel model)**

[**Note:** - Table heading will only be visible when the JTable will be added onto the JScrollPane otherwise we have to create the separate object of JTable header.]

## **//TabelTest.java**

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;
class TableTest implements ActionListener
{
 JTable table;
 JScrollPane pane;
 JTextField tf1,tf2,tf3,tf4;
 JButton b1,b2;
 JLabel lb1,lb2,lb3,lb4;
 DefaultTableModel model;
 JFrame fr1,fr2;
 TableTest()
 {
```

```

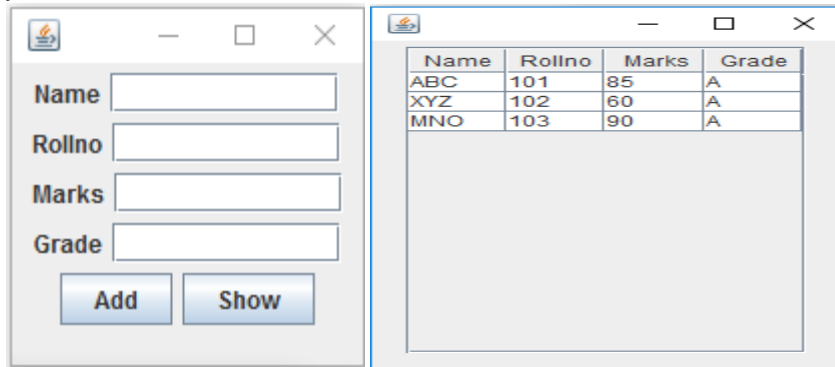
fr1=new JFrame();
fr2=new JFrame();
fr1.setLayout(new FlowLayout());
fr2.setLayout(new FlowLayout());
lb1=new JLabel("Name");
lb2=new JLabel("Rollno");
lb3=new JLabel("Marks");
lb4=new JLabel("Grade");
tf1=new JTextField(10);
tf2=new JTextField(10);
tf3=new JTextField(10);
tf4=new JTextField(10);
b1=new JButton("Add");
b2=new JButton("Show");
fr1.add(lb1); fr1.add(tf1);
fr1.add(lb2); fr1.add(tf2);
fr1.add(lb3); fr1.add(tf3);
fr1.add(lb4); fr1.add(tf4);
fr1.add(b1); fr1.add(b2);
model=new DefaultTableModel();
model.addColumn("Name");
model.addColumn("Rollno");
model.addColumn("Marks");
model.addColumn("Grade");
table=new JTable(model);
pane = new JScrollPane(table);
pane.setPreferredSize(new Dimension(200,250));
fr2.add(pane);
fr1.setSize(300,300);
fr2.setSize(300,300);
fr1.setVisible(true);
b1.addActionListener(this);
b2.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
 if(e.getSource()==b1)
 {
 String str[]=new String[4];
 str[0]=tf1.getText();
 str[1]=tf2.getText();
 str[2]=tf3.getText();
 str[3]=tf4.getText();
 model.addRow(str);
 tf1.setText("");tf2.setText("");
 tf3.setText("");tf4.setText("");
 }
}

```

```

 if(e.getSource()==b2)
 fr2.setVisible(true);
 }
 public static void main(String s[])
 {
 new TableTest();
 }
}

```



## **DialogBoxes**

DialogBoxes are the intermediate window that is use to show the intermediate on to the in front of user.

### **There are three types of DialogBoxes**

1. Builtin DialogBox
2. Custom DialogBox
3. File DialogBox

## **Builtin DialogBox**

Builtin DialogBox are also three types

1. Message DailogBox
2. Confirmation DialogBox
3. Input DialogBox

## **javax.swing.JOptionPane class**

javax.swing.JOptionPane class provides the static method to show the different type of Builtin DialogBox

- **public static void showMessageDialog(Component parent, String title, String message, int msgType)**

## **Static constant for the message types**

**JOptionPane.INFORMATION\_MESSAGE**  
**JOptionPane.QUESTION\_MESSAGE**  
**JOptionPane.WARNING\_MESSAGE**

The first argument is the parent from on which this DialogBox will be appeared.

- **Public static int showConfiemDialog(Component parent, Sting title, String message, int optionType)**

#### **Static constant for the Option types**

**JOptionPane.YES\_NO\_OPTION**  
**JOptionPane.YES\_NO\_CANCEL\_OPTION**  
**JOptionPane. OK\_CANCEL\_OPTION**

#### **Static constant for the Option types that will be return by this method**

**JOptionPane.YES\_OPTION**  
**JOptionPane.NO\_OPTION**  
**JOptionPane. OK\_OPTION**  
**JOptionPane. CANCEL\_OPTION**

This method will show the confirmation dialogBox.

The return value of this method denotes the option selected by the user.

- **public static String showInputDialog(Component parent, String defaultMessage, String message)**

This method will show the inputDialogBox.

If user clicked on the ok Button then this method returns the text entered by the user.

If cancel button is clicked then this method returns the null.

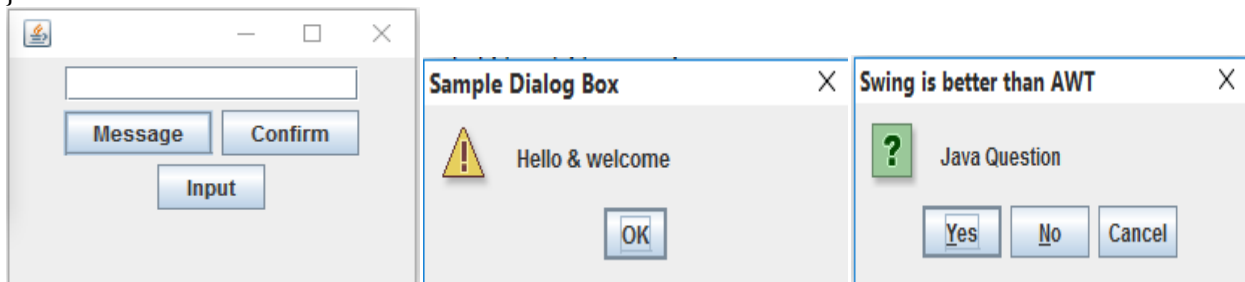
#### **//DialogBoxTest.java**

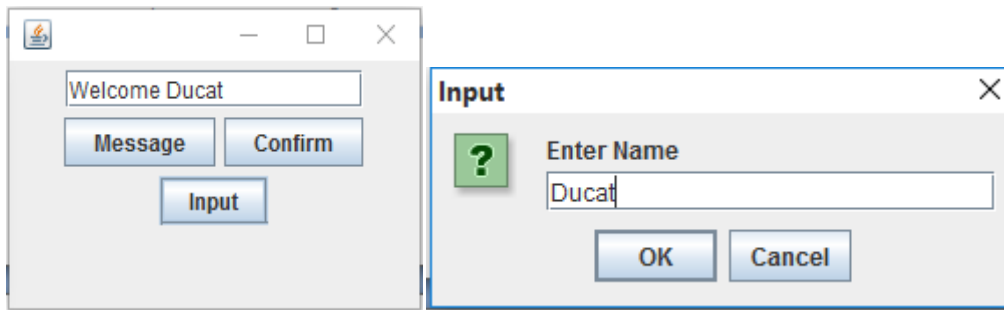
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class DialogBoxTest implements ActionListener
{
 JFrame fr;
 JButton b1,b2,b3;
 JTextField tf;
 DialogBoxTest()
 {
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 b1=new JButton("Message");
 b2=new JButton("Confirm");
 b3=new JButton("Input");
 tf=new JTextField(15);
 fr.add(tf);
 fr.add(b1);
 fr.add(b2);
 fr.add(b3);
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 }
}
```

```

 fr.setSize(200,250);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 String name=tf.getText();
 JOptionPane.showMessageDialog(fr,"Hello & welcome "+name,"Sample
Dialog Box", JOptionPane.WARNING_MESSAGE);
 }
 if(e.getSource()==b2)
 {
 int x=JOptionPane.showConfirmDialog(fr,"Java Question","Swing is
better than AWT" ,JOptionPane.YES_NO_CANCEL_OPTION);
 if(x==JOptionPane.YES_OPTION)
 tf.setText("Correct answer");
 if(x==JOptionPane.NO_OPTION)
 tf.setText("Incorrect answer");
 if(x==JOptionPane.CANCEL_OPTION)
 tf.setText("you should know it");
 }
 if(e.getSource()==b3)
 {
 String name=JOptionPane.showInputDialog(fr,"Enter Name","Enter your
name");
 if(name!=null)
 tf.setText("Welcome "+name);
 else
 tf.setText("Welcome unknown");
 }
 }
 public static void main(String s[])
 {
 new DialogBoxTest();
 }
}

```





## CustomDialogBox

Through the custom DialogBox are used to unused to custom DialogBox.

## javax.swing.JDialog class

javax.swing.JDialog class is used to prepare the custom DialogBox.

[Note: - DialogBoxes never have minimized and maximize button their Titlebar.]

### **//CustomDialogBox.java**

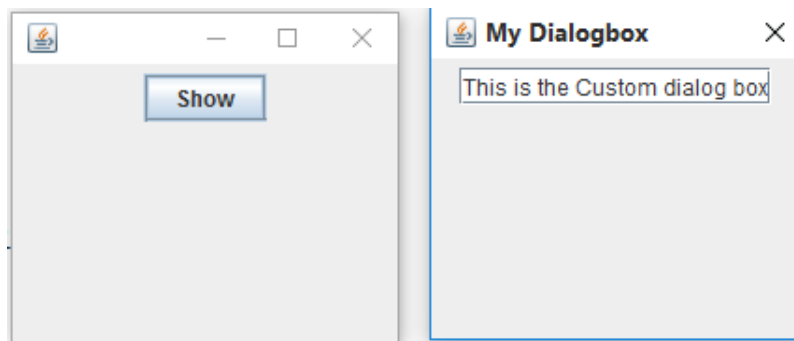
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class CustomDialogTest implements ActionListener
{
 JFrame fr;
 JDialog d;
 JButton b;
 JTextField tf;
 CustomDialogTest()
 {
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 d=new JDialog(fr,"My Dialogbox",true);
 d.setLayout(new FlowLayout());
 tf=new JTextField("This is the Custom dialog box");
 b=new JButton("Show");
 b.addActionListener(this);
 fr.add(b);
 d.add(tf);
 d.setSize(200,200);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 d.setVisible(true);
 }
 public static void main(String s[])
 {
 }
```



```

{
 new CustomDialogTest();
}

```



## FileDialogBox

JFileChooser class in the javax.swing package provides the method to show the FileDialogBoxes.

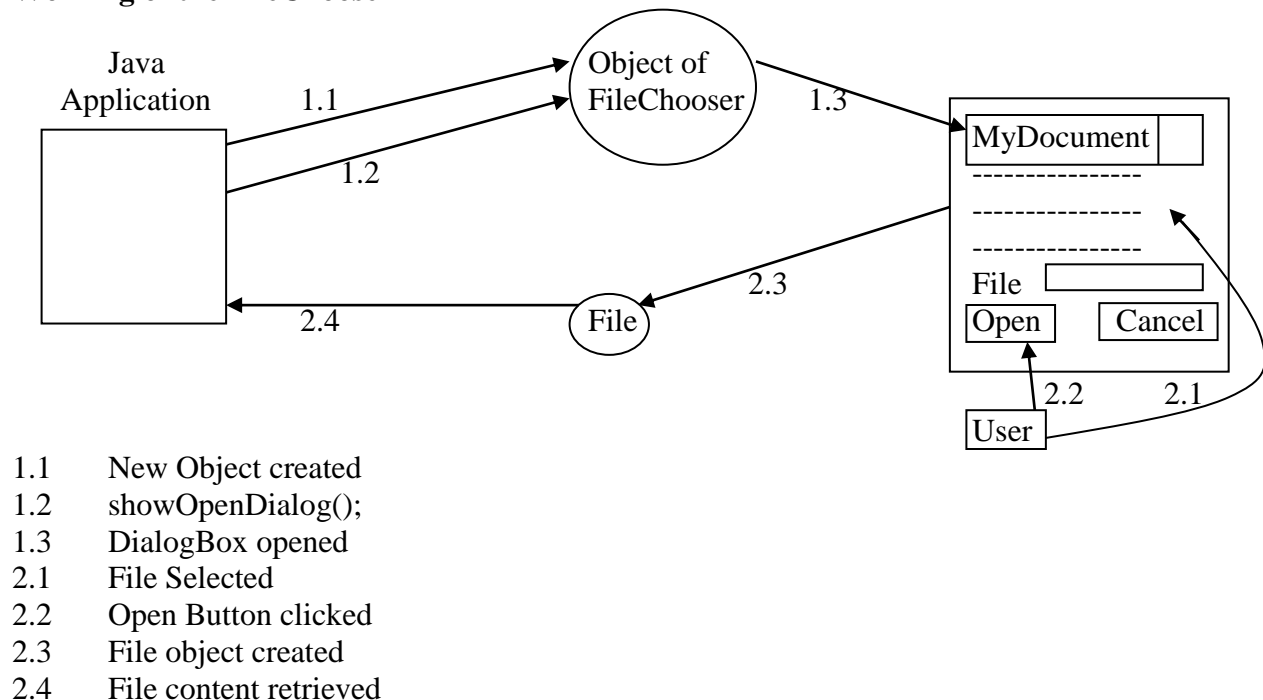
**There are two Types of FileDialogBoxes**

1. Open DialogBox
2. Save DialogBox

## Method

- **public int showOpenDialog(Frame parent)**
- **public int showSaveDialog(Frame parent)**

**Working of the FileChooser**



### //FileInputOutputTest.java

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class FileInputOutputTest implements ActionListener
{
 JFrame fr;
 JTextArea ta;
 JScrollPane pane;
 JButton b1,b2;
 FileInputOutputTest()
 {
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 ta=new JTextArea(20,30);
 pane=new JScrollPane(ta);
 b1=new JButton("Open");
 b2=new JButton("Save");
 b1.addActionListener(this);
 b2.addActionListener(this);
 fr.add(pane);
 fr.add(b1);
 fr.add(b2);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 try
 {
 JFileChooser fch=new JFileChooser();
 int x=fch.showOpenDialog(fr);
 if(x==JFileChooser.APPROVE_OPTION)
 {
 File f=fch.getSelectedFile();
 BufferedReader br=new BufferedReader(new FileReader(f));
 while(true)
 {
 String str=br.readLine();
 if(str==null)
 break;
 ta.append(str+"\n");
 }
 }
 }
 }
 }
}
```

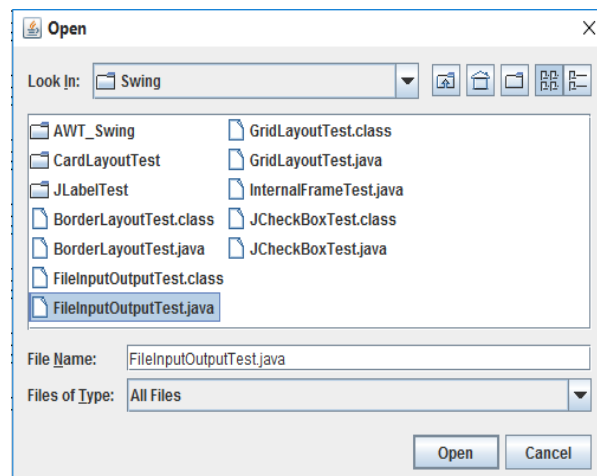
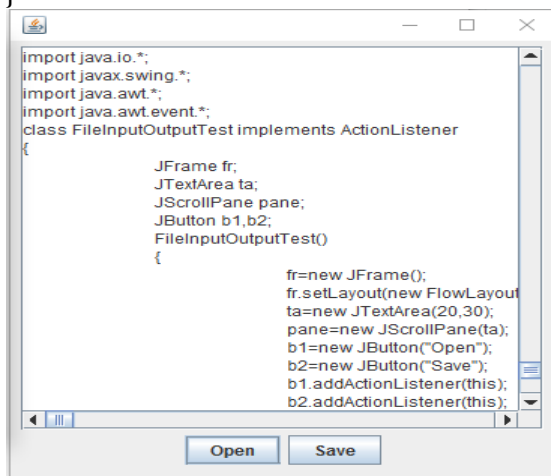
```

 }
 catch(Exception ee)
 {
 ee.printStackTrace();
 }
}
if(e.getSource()==b2)
{
 try
 {
 JFileChooser fch=new JFileChooser();
 int x=fch.showSaveDialog(fr);
 if(x==JFileChooser.APPROVE_OPTION)
 {
 File f=fch.getSelectedFile();
 String str=ta.getText();
 PrintWriter pw=new PrintWriter(f);
 pw.println(str);
 pw.flush();
 }
 }
 catch(Exception ee)
 {
 ee.printStackTrace();
 }
}

}

public static void main(String s[])
{
 new FileInputOutputTest();
}
}

```



## LayoutManagers

layoutManager are the predefined arrangements of the component on to the container.

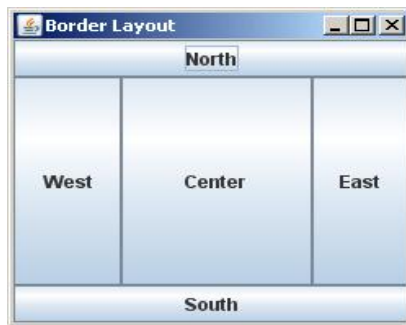
Some layoutManagers are

- **FlowLayout**
- **BorderLayout**
- **GridLayout**
- **CardLayout**

## BorderLayout

BorderLayout is the default layout of the frame and its subclasses.

BorderLayout arrange the components in the five different regions north, south, west and center.



In one region only one component can be placed.

There is the overloaded to add the component in specific region.

- **public void add(Component c, int region)**

## Static constants for the region

- **BorderLayout.NORTH**
- **BorderLayout.SOUTH**
- **BorderLayout.EAST**
- **BorderLayout.WEST**
- **BorderLayout.CENTER**

## Constructors of BorderLayout

- **public BorderLayout();**
- **public BorderLayout(int horizontalSpace, int verticalSpace)**

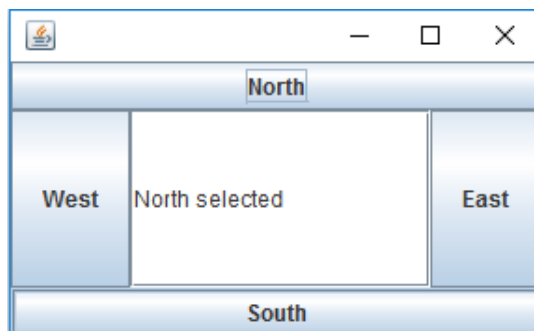
## //BorderLayoutTest.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class BorderLayoutTest implements ActionListener
{
 JFrame fr;
 JButton b1,b2,b3,b4;
```

```

 JTextField tf;
 BorderLayoutTest()
 {
 fr=new JFrame();
 fr.setLayout(new BorderLayout());
 b1=new JButton("North");
 b2=new JButton("South");
 b3=new JButton("East");
 b4=new JButton("West");
 tf=new JTextField();
 fr.add(b1,BorderLayout.NORTH);
 fr.add(b2,BorderLayout.SOUTH);
 fr.add(b3,BorderLayout.EAST);
 fr.add(b4,BorderLayout.WEST);
 fr.add(tf,BorderLayout.CENTER);
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b4.addActionListener(this);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 tf.setText("North selected");
 if(e.getSource()==b2)
 tf.setText("South selected");
 if(e.getSource()==b3)
 tf.setText("East selected");
 if(e.getSource()==b4)
 tf.setText("West selected");
 }
 public static void main(String s[])
 {
 new BorderLayoutTest();
 }
}

```



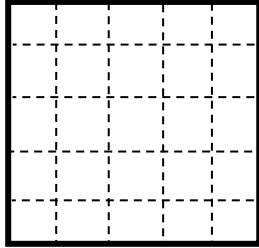
## **GridLayout**

GridLayout arrange the components in the form of grids one grid can contain only component.

Each grid always has equal size.

Grids automatically filled with the left to right and with the top to bottom manner.

**GridLayout**



## **Constructor**

- **public GridLayout()**
- **public GridLayout(int noOfRows, int noOfColumns)**

In case of 1<sup>st</sup> constructor only the single Grid with one row and one column will be created.

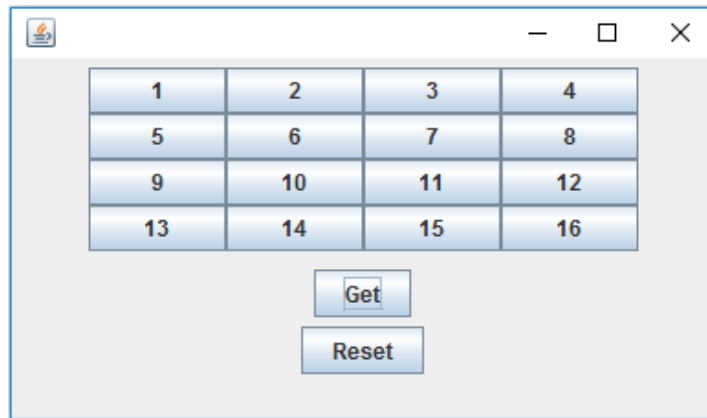
## **//GridLayoutTest.java**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class GridLayoutTest implements ActionListener
{
 JFrame fr;
 JPanel p1,p2;
 JButton b[]=new JButton[16];
 JButton b1,b2;
 int count=0;
 GridLayoutTest()
 {
 fr=new JFrame();
 p1=new JPanel();
 p2=new JPanel();
 p1.setLayout(new GridLayout(4,4));
 fr.setLayout(new FlowLayout());
 for(int i=0 ; i<16 ; i++)
 {
 b[i]=new JButton(""+(i+1));
 b[i].setVisible(false);
 p1.add(b[i]);
 }
 b1=new JButton("Get");
 b2=new JButton("Reset");
 b1.addActionListener(this);
```

```

 b2.addActionListener(this);
 p2.add(b1);
 p2.add(b2);
 p1.setPreferredSize(new Dimension(300,100));
 p2.setPreferredSize(new Dimension(100,100));
 fr.add(p1);
 fr.add(p2);
 fr.setSize(400,400);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 while(count<16)
 {
 int x=(int)(Math.random()*16);
 System.out.println(x);
 if(b[x].isVisible())
 continue;
 b[x].setVisible(true);
 count++;
 }
 break;
 }
 if(e.getSource()==b2)
 {
 for(int i=0 ; i<16 ; i++)
 b[i].setVisible(false);
 count=0;
 }
 }
 public static void main(String s[])
 {
 new GridLayoutTest();
 }
}

```

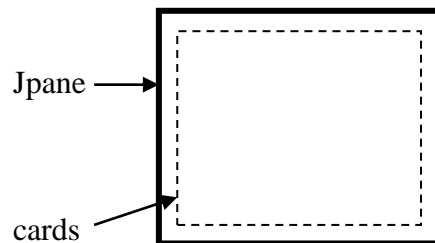


## CardLayout

It arranges the component in the form of back of cards.

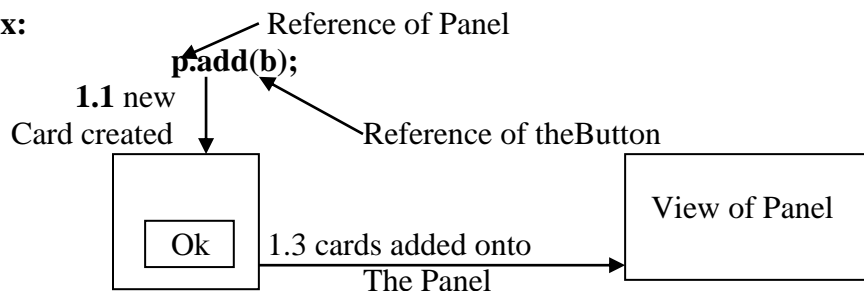
Each card can contains the only one component and at the onetime only one card can be visible.

The card layout can't be applicable onto the frame or their subclasses card layout applicable on to the pane.



When ever we add the new component onto the panel that has the CardLayout then automatically the new card will be created and the new component will be added into it.

**Ex:**



There is the overloaded add method to add the component onto the contains that has the CardLayout.

## Signature

- `public void add(Component c, String cardName)`

## Methods of CardLayout

- `public void show(String cardName, Component parent)`

This will show the specified card.



- **public void first(Component parent)**

This method will show the first card.

- **public void last(component c)**

This method show last card

- **public void next()**

- **public void previous()**

#### **//CardLayoutTest.java**

```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
class CardLayoutTest implements ActionListener
{
 JFrame fr;
 JButton b1,b2,b3,b4;
 JLabel lb1,lb2,lb3,lb4;
 JPanel p1,p2;
 CardLayout cd;
 CardLayoutTest()
 {
 cd=new CardLayout();
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 p1=new JPanel();
 p2=new JPanel();
 p2.setLayout(cd);
 b1=new JButton("First");
 b2=new JButton("Prev");
 b3=new JButton("Next");
 b4=new JButton("Last");
 lb1=new JLabel(new ImageIcon("1.jpg"));
 lb2=new JLabel(new ImageIcon("2.jpg"));
 lb3=new JLabel(new ImageIcon("3.jpg"));
 lb4=new JLabel(new ImageIcon("4.jpg"));
 p1.add(b1);p1.add(b2);p1.add(b3);p1.add(b4);
 p2.add(lb1,"first");
 p2.add(lb2,"second");
 p2.add(lb3,"third");
 p2.add(lb4,"fourth");
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b4.addActionListener(this);
 fr.add(p1);
 }
}
```

```

 fr.add(p2);
 fr.setSize(1024,800);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==b1)
 {
 cd.first(p2);
 }
 if(e.getSource()==b2)
 cd.previous(p2);
 if(e.getSource()==b3)
 cd.next(p2);
 if(e.getSource()==b4)
 cd.last(p2);
 }
 public static void main(String s[])
 {
 new CardLayoutTest();
 }
}

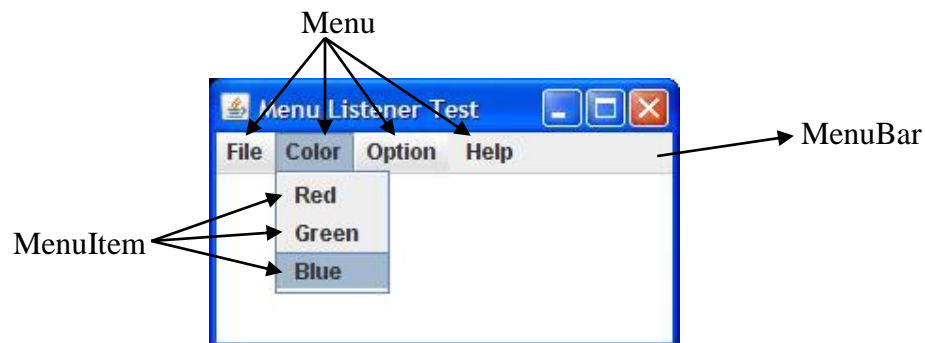
```



### Working with the Menus

In the javax.swing package there are the three classes for the Menus

1. JMenuBar
2. JMenu
3. JMenuItem



## **JMenuBar**

This class represents the member onto the frame.

### **Constructor**

- **public JMenuBar()**

[Note: - setJMenuBar() of the JFrame class is use to set the JMenuBar onto the JFrame.]

### **Method of JMenuBar**

- **public void add(JMenu m)**  
It will add the JMenu onto the MenuBar.

## **JMenu class**

An instance of JMenu class is used to represent the menu onto the Frame.

- **public JMenu(String title)**

### **Method of JMenu**

- **public void add(JMenu item)**  
This will add the MenuItem on to the menu
- **public void setMnemonic(int key)**  
This method is use to set the shortcut key with the combination of “Alt” key.  
There are the static constants for the key in the KeyEvent class.

## **KeyEvent class**

KeyEvent.VK\_A  
KeyEvent.VK\_B  
KeyEvent.VK\_C  
KeyEvent.VK\_D  
Etc

## **JMenuItem**

An instance of this class is represents the MenuItem that will added onto the JMenu.

### **Constructor**

- **public JMenuItem()**
- **public JMenuItem(String title)**
- **public JMenuItem(Icon i)**

[Note: - add() method of the JMenu is use to add the MenuItem onto the menu.]

### **Method of JMenuItem**

- **public void setAcceleration(KeyStroke k)**  
This method is use to set the shortcut on the MenuItem.  
KeyStroke class has the static factory method to create and provide there object.

- `public static KeyStroke getKeyStroke(int key, int modifier)`

### Static constants for key

`KeyEvent.VK_A`  
`KeyEvent.VK_B`  
 Etc

### Static constant for the modifier

`KeyEvent.SHIFT_MASK`  
`KeyEvent.CTRL_MASK`  
`KeyEvent.ALT_MASK`

**Ex: -** `KeyStroke ks=KeyStroke.getKeyStroke(KeyEvent.VK_A, KeyEvent.SHIFT_MASK);`

`mi.setAcceleration(ks);`

[**Note: -** On the MenuItem actionEvent is generated as the MenuItem is clicked.]

### Steps to Applied the Menu on JVM

1. Create the object to MenuBar
2. Create the object of JMenu
3. Create the object of JMenuItem
4. Add the MenuItem on the JMenu
5. Add the Menu on to the MenuBar
6. Set the MenuBar on to the JFrame

### //MenuTest.java

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class MenuTest implements ActionListener
{
 JFrame fr;
 JMenuBar bar;
 JMenu m;
 JMenuItem mi1,mi2,mi3;
 JLabel lb;
 MenuTest()
 {
 fr=new JFrame();
 fr.setLayout(null);
 bar=new JMenuBar();
 m=new JMenu("Color");
 mi1=new JMenuItem("Red");
 mi2=new JMenuItem("Green");
 mi3=new JMenuItem("Blue");
 mi1.addActionListener(this);
 mi2.addActionListener(this);
```

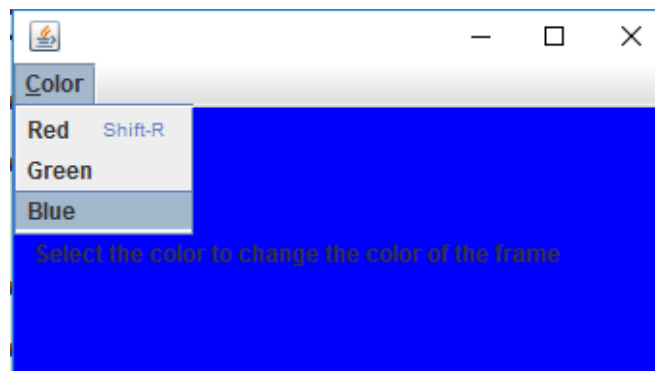
```

 mi3.addActionListener(this);
 m.add(mi1);m.add(mi2);m.add(mi3);
 bar.add(m);
 fr.setJMenuBar(bar);
 lb=new JLabel("Select the color to change the color of the frame");
 lb.setBounds(10,50,300,50);
 fr.add(lb);
 m.setMnemonic(KeyEvent.VK_C);

 mi1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,KeyEvent.SHIFT_MASK
));

 fr.setSize(400,400);
 fr.setVisible(true);
 }
 public void actionPerformed(ActionEvent e)
 {
 Container cr=fr.getContentPane();
 if(e.getSource()==mi1)
 cr.setBackground(Color.red);
 if(e.getSource()==mi2)
 cr.setBackground(Color.green);
 if(e.getSource()==mi3)
 cr.setBackground(Color.blue);
 }
 public static void main(String s[])
 {
 new MenuTest();
 }
}

```



### **m.addSeparator();**

m.addSeparator is used to add the separator in the menu in between the menu items.

### **JPopupMenu**

JPopupMenu are the menu that will be displayed upon the any component as the result of any KeyEvent.

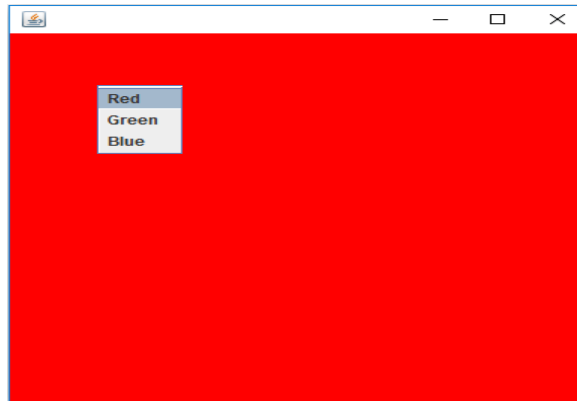
### //JPopupMenu

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class PopupMenuTest extends MouseAdapter implements ActionListener
{
 JFrame fr;
 JPopupMenu menu;
 JMenuItem m1,m2,m3;
 JTextField f1;
 PopupMenuTest()
 {
 fr=new JFrame();
 fr.setLayout(null);
 f1.setBounds(40,50,50,80);
 m1=new JMenuItem("Red");
 m2=new JMenuItem("Green");
 m3=new JMenuItem("Blue");
 menu=new JPopupMenu();
 menu.add(m1);menu.add(m2);menu.add(m3);
 m1.addActionListener(this);
 m2.addActionListener(this);
 m3.addActionListener(this);
 fr.addMouseListener(this);
 fr.setSize(400,400);
 fr.setVisible(true);
 }
 public void mouseClicked(MouseEvent e)
 {
 if(e.getButton()==3)
 {
 Point p=e.getPoint();
 int x=(int)p.getX();
 int y=(int)p.getY();
 menu.show(fr , x, y); // to show the popup menu
 }
 }
 public void actionPerformed(ActionEvent e)
 {
 Container c=fr.getContentPane();
 if(e.getSource()==m1)
 c.setBackground(Color.red);
 if(e.getSource()==m2)
 c.setBackground(Color.green);
 if(e.getSource()==m3)
 c.setBackground(Color.blue);
 }
}
```

```

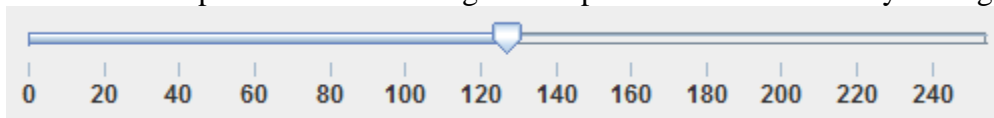
public static void main(String s[])
{
 new PopupMenuTest();
}
}

```



## JSlider class

This is the component that is use to get the input from the end-user by sliding.



## Constructor

- **public JSlider()**

It will make the slide object and with the one and 100 as the minimum and maximum value with the default slider position at the 50.

- **public JSlider(int max, int min, int value)**

This constructor is use to JSlider object with the specified minimum, maximum and default value.

## Method

- **public int getValue()**

This method returns the current value of the slider.

- **public void setValue(int val)**

This method is use to set the value of the slider.

- **public void setPointTicks(boolean b)**

This method if the argumented value is true then Ticks will be displayed on the slider but before the calling of this method the calling method must be invoked.

- **public void setMinorTicksSpacing(int x)**

- **public void setPaintLabels(boolean b)**

- **public void setSnapToTicks(boolean b)**

[**Note:** - On the JSlider as the value getsChanged immediately javax.swing.event.ChangeEvent; accrued.]

## **Listener is ChangeListener (I)**

### **Callback Method**

- **public void stateChanged(ChangeEvent e)**

#### **//SliderTest.java**

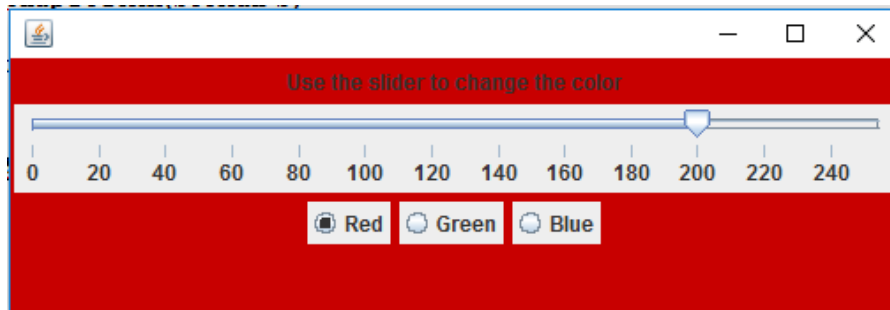
```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
class SliderTest implements ChangeListener
{
 JFrame fr;
 JSlider slider;
 JLabel lb;
 JRadioButton rb1,rb2,rb3;
 SliderTest()
 {
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 slider=new JSlider(0,255);
 slider.setMajorTickSpacing(20);
 slider.setPaintTicks(true);
 slider.setPaintLabels(true);
 slider.setPreferredSize(new Dimension(500,50));
 lb=new JLabel("Use the slider to change the color");
 rb1=new JRadioButton("Red");
 rb2=new JRadioButton("Green");
 rb3=new JRadioButton("Blue");
 ButtonGroup bg=new ButtonGroup();
 bg.add(rb1); bg.add(rb2);
 bg.add(rb3);
 slider.addChangeListener(this);
 fr.add(lb); fr.add(slider);
 fr.add(rb1);fr.add(rb2);fr.add(rb3);
 fr.setSize(600,300);
 fr.setVisible(true);
 }
 public void stateChanged(ChangeEvent e)
 {
 int x=slider.getValue();
 Container c=fr.getContentPane();
 if(rb1.isSelected())
 c.setBackground(new Color(x,0,0));
 if(rb2.isSelected())
 c.setBackground(new Color(0,x,0));
 }
}
```



```

 if(rb3.isSelected())
 c.setBackground(new Color(0,0,x));
 }
 public static void main(String s[])
 {
 new SliderTest();
 }
}

```



### javax.swing.Timer class

An instance of this class is used to set the timer in the swing application.

Timer object always be registered with the ActionListener and after the specified time automatically invokes the ActionEvent.

### Constructor

- **public Timer(int Miliseconds, ActionListener al)**

### Method of the Timer

- **public void start()**

This method will start to generate the ActionEvent

- **public void stop()**

This method will stop generation of the ActionEvent.

### //TimerTest.java

```

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class TimerTest implements ActionListener
{
 JFrame fr;
 JButton b1,b2;
 JLabel lb1,lb2;
 Timer t;
 Panel p1,p2;
 TimerTest()
 {
 fr=new JFrame();
 fr.setLayout(new FlowLayout());
 }
}

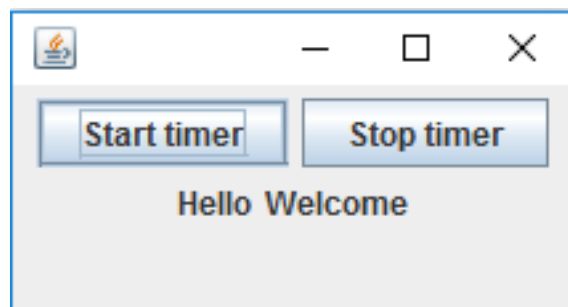
```

```

 b1=new JButton("Start timer");
 b2=new JButton("Stop timer");
 lb1=new JLabel("Hello");
 lb2=new JLabel("Welcome");
 t=new Timer(500 , this);
p1=new Panel();
 fr.add(b1); fr.add(b2);
 fr.add(lb1); fr.add(lb2);
Toolkit t=Toolkit.getDefaultToolkit();
Dimension d=t.getScreenSize();
int a=(int)d.getWidth();
int b=(int)d.getHeight();
 fr.setSize(a,b);
 fr.setVisible(true);
 b1.addActionListener(this);
 b2.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource()==t)
 {
 String temp=lb1.getText();
 lb1.setText(lb2.getText());
 lb2.setText(temp);
 }
 if(e.getSource()==b1)
 t.start();
 if(e.getSource()==b2)
 t.stop();
 }

 public static void main(String s[])
 {
 new TimerTest();
 }
}

```



## **JColorChooser**

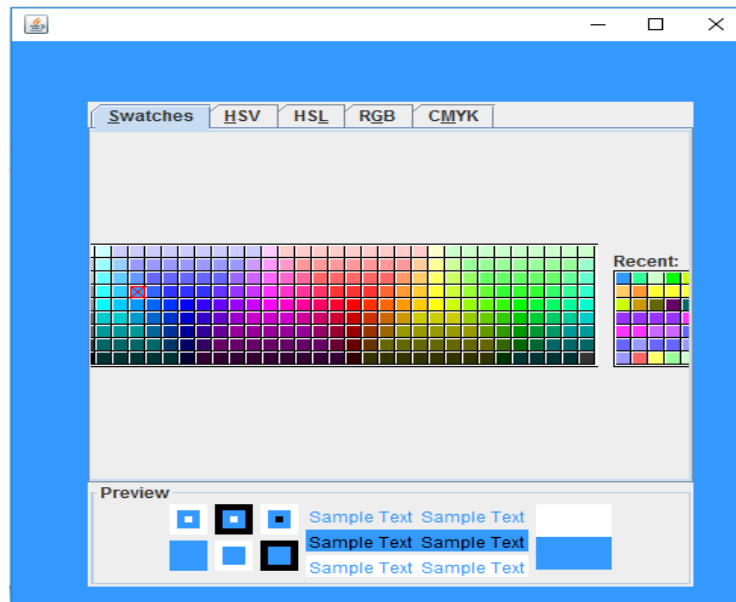
This is component that provides the color pane from that we can select the color.

An instance of JColorChooser automatically always associated with the colorSelectionModel instance and we have to register the change listener with that model.

As the color gets selected on the colorChooser an immediately change event is generated and by the following method of the colorSelectionModel we can get the selected color.

### **//ColorTest.java**

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.colorchooser.*;
import java.awt.*;
class ColorChooserTest implements ChangeListener
{
 JFrame fr;
 JColorChooser ch;
 ColorSelectionModel model;
 ColorChooserTest()
 {
 fr=new JFrame();
 fr.setLayout(null);
 ch=new JColorChooser(); // step 1
 model = ch.getSelectionModel(); // step 2
 model.addChangeListener(this); // step 3
 ch.setBounds(50,50,400,400);
 fr.add(ch);
 fr.setSize(500,500);
 fr.setVisible(true);
 }
 public void stateChanged(ChangeEvent e)
 {
 Color c=model.getSelectedColor();
 fr.getContentPane().setBackground(c);
 }
 public static void main(String s[])
 {
 new ColorChooserTest();
 }
}
```



### Signature

- **public Color getSelectedColor()**

### JInternalFrame

JInternalFrame is the container that is always be open within the frame.

### Steps to prepare the InternalFrame

1. Create to prepare the InternalFrame
2. Create the object of DesktopPane
3. Set the desktopPane as the contentPane on the Frame
4. Add the InternalFrame on to the DesktopPane.

### Constructor of JInternalFrame

- **public JInternalFrame(String title)**
- **public JInternalFrame(String title, boolean closeable, boolean resizable, boolean maximizable, boolean minimizable)**

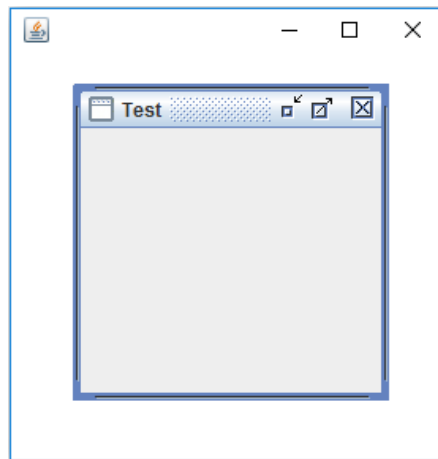
### //InternalFrameTest.java

```
import javax.swing.*;
class InternalFrameTest
{
 JFrame fr;
 JInternalFrame ifr;
 InternalFrameTest()
 {
 fr=new JFrame();
 fr.setLayout(null);
 //step 1
 ifr=new JInternalFrame("Test" ,true , true ,true ,true);
 ifr.setBounds(50,50,200,200);
 }
}
```

```

 ifr.setVisible(true);
 // step 2
 JDesktopPane pane=new JDesktopPane();
 //step 3
 fr.setContentPane(pane);
 //step 4
 pane.add(ifr);
 fr.setSize(300,300);
 fr.setVisible(true);
 }
 public static void main(String s[])
 {
 new InternalFrameTest();
 }
}

```



## **JTree**

JTree is the component that is used to display the content in the tree structure format.

DefaultMutableTreeNode class in the Javax.swing package interface of this class works as the node of the tree.

## **Constructor**

- **public JTree(DefaultMutableTreeNode Root)**

### **//JTreeTest.java**

```

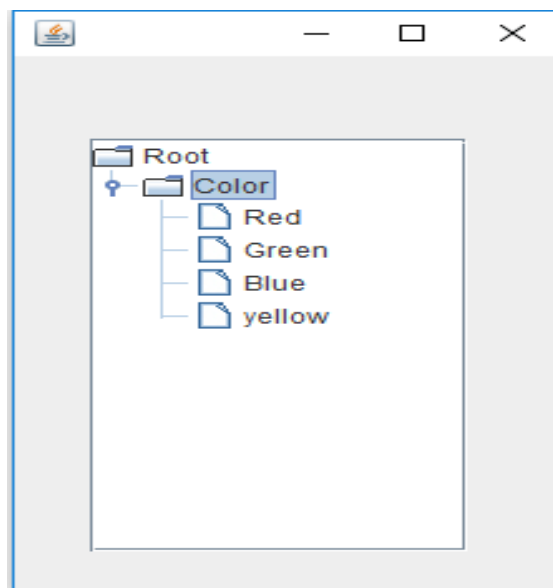
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;
class TreeTest
{
 JFrame fr;
 JScrollPane pane;
 JTree tree;
 TreeTest()
 {

```

```

fr=new JFrame();
fr.setLayout(null);
DefaultMutableTreeNode root=new DefaultMutableTreeNode("Root");
DefaultMutableTreeNode color=new DefaultMutableTreeNode("Color");
DefaultMutableTreeNode red=new DefaultMutableTreeNode("Red");
DefaultMutableTreeNode green=new DefaultMutableTreeNode("Green");
DefaultMutableTreeNode blue=new DefaultMutableTreeNode("Blue");
DefaultMutableTreeNode yellow=new DefaultMutableTreeNode("yellow");
tree=new JTree(root);
pane=new JScrollPane(tree);
color.add(red);
color.add(green);
color.add(blue);
color.add(yellow);
root.add(color);
//tree.setRoot(root);
pane.setBounds(30,50,150,250);
fr.add(pane);
Toolkit t=Toolkit.getDefaultToolkit();
Dimension d=t.getScreenSize();
int width=(int)d.getWidth();
int height=(int)d.getHeight();
fr.setSize(width,height);
fr.setVisible(true);
}
public static void main(String s[])
{
 new TreeTest();
}
}

```



## **JProgressBar**

This is the component which is used to show the progress of the any task.

### **Constructor**

- **public JProgressBar(int min, int max)**
- **public JProgressBar()**

By using the default constructor the progressbar will have their default value one and 100 as the min value and max value.

### **Method**

- **public void setValue(int value)**

Up to the specified value the progressbar will be filled.

- **public void setString(String str)**

By using this method the argument string will be displayed on to the progress bar.

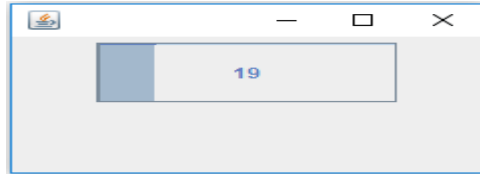
### **//ProgressBar.java**

```
import javax.swing.*;
import java.awt.*;
class ProgressTest
{
 JFrame fr1,fr2;
 JProgressBar pbar;
 ProgressTest()
 {
 fr1=new JFrame();
 fr1.setLayout(new FlowLayout());
 fr2=new JFrame();
 pbar=new JProgressBar();
 pbar.setPreferredSize(new Dimension(150,50));
 pbar.setStringPainted(true);
 fr1.add(pbar);
 fr1.setSize(250,150);
 fr1.setVisible(true);
 fr2.setSize(300,300);
 for(int i=1 ; i<=100 ; i++)
 {
 try
 {
 Thread.sleep(200);
 }
 catch(InterruptedException e)
 {
 }
 pbar.setValue(i);
 pbar.setString(""+i);
 }
 fr1.setVisible(false);
 }
}
```

```

 fr2.setVisible(true);
 }
 public static void main(String s[])
 {
 new ProgressTest();
 }
}

```



### **To get the current Desktop resolution**

```

Toolkit t=Toolkit.getDefaultToolkit();
Dimension d=t.getScreenSize();
int width=(int)d.getWidth();
int height=(int)d.getHeight();

fr.setSize(Width, Height);

```

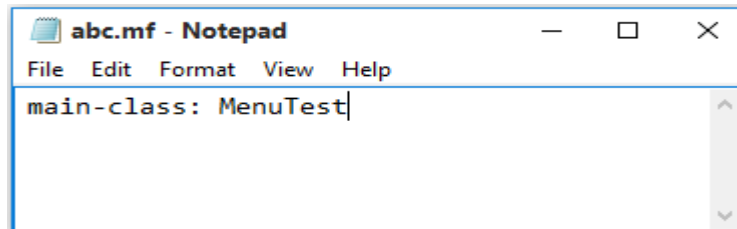
### **Create the executable jar of java Application**

#### **Jar Tool**

Jar tool is use to make the executable jar file

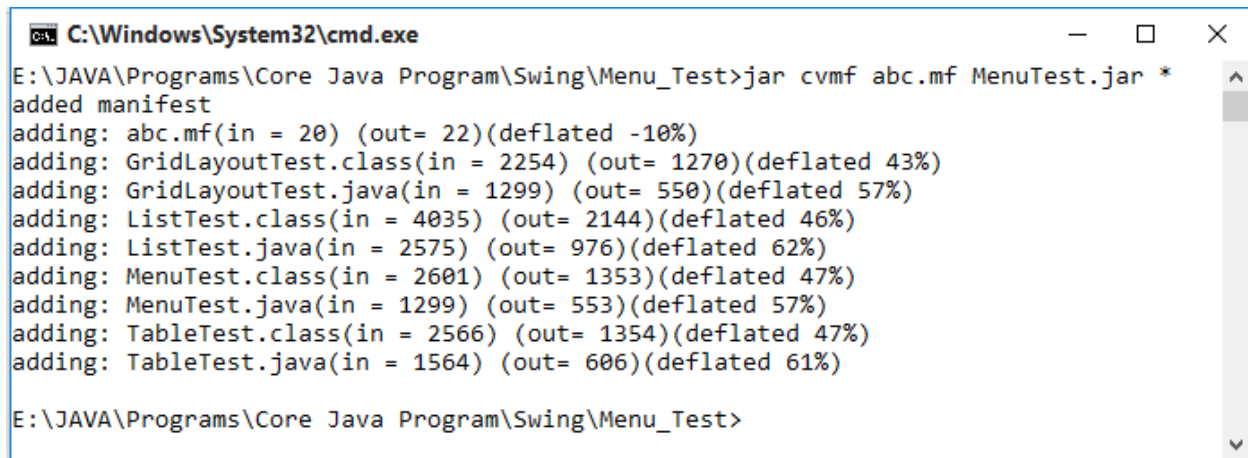
**Step1:** - Place the **.class** file of java application in any separate folder.

**Step2:** - Create the Menu\_Test file and save it with **.mf** extension in the Menu\_Test file just write the name of main class.



**Step3:** - Open the Dos-Prompt and use the following command **jar cvmf**





```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Swing\Menu_Test>jar cvmf abc.mf MenuTest.jar *
added manifest
adding: abc.mf(in = 20) (out= 22)(deflated -10%)
adding: GridLayoutTest.class(in = 2254) (out= 1270)(deflated 43%)
adding: GridLayoutTest.java(in = 1299) (out= 550)(deflated 57%)
adding: ListTest.class(in = 4035) (out= 2144)(deflated 46%)
adding: ListTest.java(in = 2575) (out= 976)(deflated 62%)
adding: MenuTest.class(in = 2601) (out= 1353)(deflated 47%)
adding: MenuTest.java(in = 1299) (out= 553)(deflated 57%)
adding: TableTest.class(in = 2566) (out= 1354)(deflated 47%)
adding: TableTest.java(in = 1564) (out= 606)(deflated 61%)

E:\JAVA\Programs\Core Java Program\Swing\Menu_Test>
```

This executable file will run the public jre.

## **APPLET**

Applets are the java that extends the functionality of web page.

Applets program are resides onto the web server and transmitted over the network to be executed on to the web browser.

Applets are the managed objects.

A managed object is the object whose life cycle is managed by the environment.

In case of applet the lifecycle is managed by the web browser life cycle of object means the creation methods onto it and finally description of the object object.

### **java.applet.Applet class**

This class provides the life cycle methods that are invoked by the web browser timely onto the applet objet.

**Life-cycle() method is the applet class as follows.**

- **public void init()**

This method is invoked by the web browser only once just after the creation of the applet object.

- **public void start()**

This method is invoked by the web browser after the init() method and subsequently each time when the applet gets pre loaded.

- **public void paint(Graphics g)**

This method always be invoked after the start() method and subsequently each time when the content on the applet redrawn.

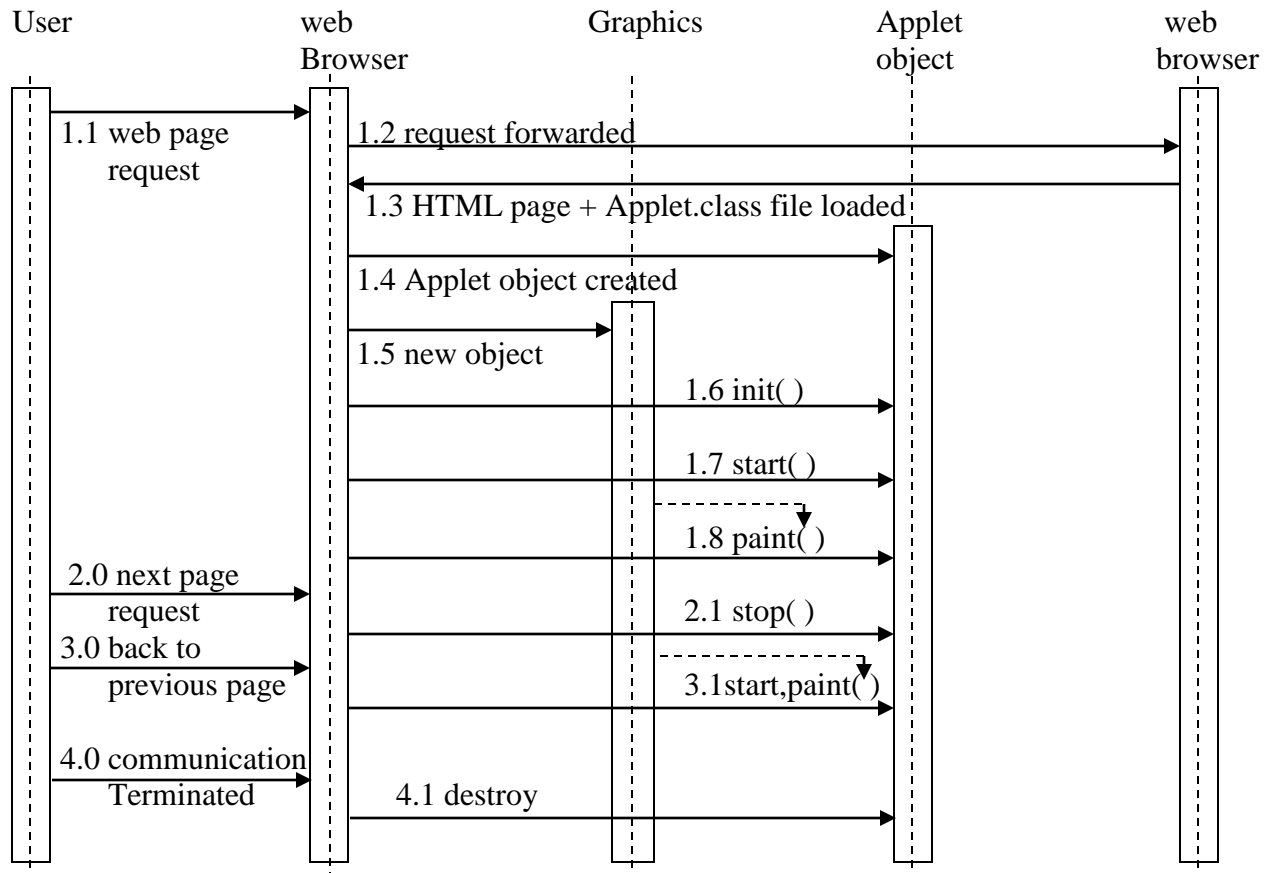
- **public void stop()**

This method is invoked each time when the execution of the applet gets paused.

- **public void destroy()**

This method invoked by the web browser just before the restriction of the applet object.

## Sequence Diagram of Applet lifecycle



[**Note:** - All the life-cycle() method are the callback method because their names always be provide by the enviroment.]

### //MyApplet.java

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class MyApplet extends Applet implements ActionListener
{
 Button b1,b2;
 int x;
 public void init()
 {
 b1=new Button("Rect");
 b2=new Button("Circle");
 b1.addActionListener(this);
 b2.addActionListener(this);
 add(b1);
 add(b2);
 }
}

```

```

public void paint(Graphics g)
{
 if(x==1)
 {
 g.setColor(Color.RED);
 g.fillRect(30,70,150,150);
 }
 if(x==2)
 {
 g.setColor(Color.GREEN);
 g.fillOval(30,70,150,150);
 }
}
public void actionPerformed(ActionEvent e)
{
 if(e.getSource()==b1)
 {
 x=1;
 }
 if(e.getSource()==b2)
 {
 x=2;
 }
 repaint();
 // to invoke the paint() explicitly
}
}

```

**[Note: - Applet class must always be declared as public class.]**

Applet classes never contains the main() method so the class can't be run directly through the java tool.

In the web browsers java plug-in are exist to load the Applet class and make the applet objects.

In order to load the Applet class in the web browser just make the html file and make the entry of Applet class into it.

#### **//MyAppletTest.html**

```

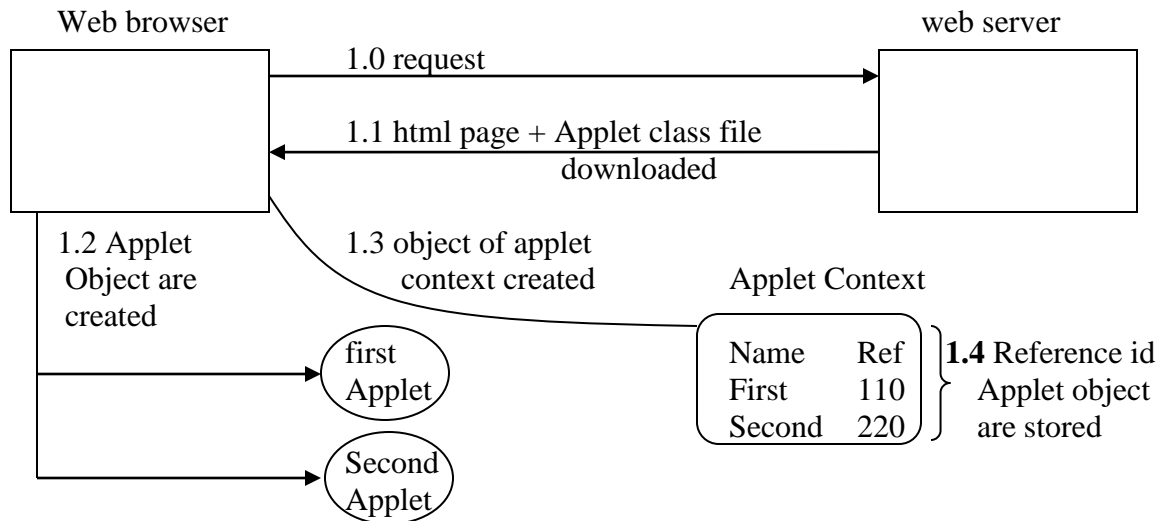
<html>
<body>
 <center>This is my applet</center>
 <applet code="MyApplet.class" width="300" height="300">
 </applet>
</body>
</html>

```

## InterApplet Communication

One Applet can get reference id of another Applet object so that the one Applet can be controlled by another applet.

In web browser there is an object of applet context that contains the reference id of all the applet object created by the java plug-in from the web browser.



## Method of java.applet.Applet class

- **public AppletContext getAppletContext()**

This method returns the reference id of AppletContext of the object

## Method of AppletContext

- **public Applet getApplet(String appletName)**

This method returns the reference id of applet object whose name specified with the argument.

[**Note:** - Name of the applet has to be specified in the html file within the applet tag.]

## Limitation of Applet

Through the applet program we can never use the user resources outside the boundary of the JRE. That means the applet program we can never access the file from file and database.

## Applet Viewer

Applet viewer is the tool in the JDK that is used to execute the applet program.

Without making the HTML file we can also run the applet program by making the applets in the .java file entry of the comment.

## JDK

1. for each loop
2. var-args
3. enum
4. static import
5. AutoBoxing
6. Annotation
7. Generics
8. Co-variant return type
9. Concurrency API

### For each Loop

For each loop is introduced in the JDK 1.5 to read the value from the array effectively. This is new feature of java file that provides the different way to read the values from array.

**Ex: -**

```
int a[] = {2, 3, 4, 5, 7, 9};
for(int i=0; i<a.length; i++)
{
 System.out.println(a[i]);
}
```

This is the normal for loop to read the array values

```
int a[]={5,6,7,8,9};
for(int i:a)
{
 System.out.println(a[i]);
}
```

This is the for each loop

### Limitation of for each Loop

Limitation of for each loop that we can't read the value by jumping the location. We can't read the array of objects.

**Ex: -**

```
Student st[]={new Student(101,85), new Student(102,75), new Student(103,65)};
for(Student s:st)
{
 System.out.println(s.report());
}
```

Foreach loop to get the value from two dimensional array.

**Ex: -**

```
int [][]a={{2,4,3},{4,5,6,7},{8,4,5,6,7,9}};
for(int i[]:a)
{
 for(int j:i)
```

```

 {
 System.out.println(j);
 }
 }
}

```

## **Var-args (Variable Length Argument)**

This is the new feature introduced in the JDK 1.5 to make the argument list of any method with the variable length that means in any method we can pass the arguments according to the requirement.

Reduce the code of complexity.

**//Test.java**

```

class Test
{
 static void m1(int ...x)
 {
 System.out.println("m1 invoked");
 }
 public static void main(String args[])
 {
 m1(5);
 m1(5,6);
 m1(5,6,7);
 }
}

```

```
E:\JAVA\Programs\Core Java Program\JDK\Var-args>javac Test.java
```

```
E:\JAVA\Programs\Core Java Program\JDK\Var-args>java Test
```

```

m1 invoked
m1 invoked
m1 invoked

```

**//Demo.java**

```

class Demo
{
 public void m1(int ...x)
 {
 System.out.println("m1 invoked");
 }
 public static void main(String args[])
 {
 m1(5);
 m1(5,6);
 m1(5,6,7);
 }
}

```

```

E:\JAVA\Programs\Core Java Program\JDK\Var-args>javac Demo.java
Demo.java:9: error: non-static method m1(int...) cannot be referenced from a static context
 m1(5);
 ^
Demo.java:10: error: non-static method m1(int...) cannot be referenced from a static context
 m1(5,6);
 ^
Demo.java:11: error: non-static method m1(int...) cannot be referenced from a static context
 m1(5,6,7);
 ^
3 errors

```

[Note: - Because all value goes into x and no any value is rest for y.

## Rules for declaration the var-args

1. var-args always must be declared in the right most in any method.
2. There can be only one var-args in any method declaration.

**Ex: -**

```

class Test1
{
 //static void m1(float f, int ...x, int ...y) //Invalid
 //static void m1(int ...x, float f) //Invalid
 static void m1(float f, int ...x) //valid
 {
 System.out.println("m1 invoked");
 }
 public static void main(String []args)
 {
 m1(5,6);
 m1(5,6,7);
 }
}

```

[Note: - Internally for String the value in the var-args type argument the array object will be created.]

```

class Test2
{
 static void m1(int ...x)//valid
 {
 System.out.println("m1 invoked");
 }
 public static void main(String []args)
 {
 m1(5,6,7,2,9);
 }
}

```

1.1 new array object created

1.2 Reference of array is passed

The diagram illustrates the internal process of var-args. In the `main` method, the call `m1(5,6,7,2,9);` is shown. An arrow labeled '1.1 new array object created' points from the arguments to a box representing an array containing the values 5, 6, 7, 2, and 9. Another arrow labeled '1.2 Reference of array is passed' points from this array box to the `int ...x` parameter of the `m1` method.



In the same way as we get the values from the array get the values from the var-args.

```
class Test3
{
 static void m1(int ...x)//vlaid
 {
 for(int i : x)
 {
 System.out.println(i);
 }
 }
 public static void main(String []args)
 {
 m1(5,6,7,2,9);
 }
}
```

E:\JAVA\Programs\Core Java Program\JDK\Var-args>java Test3

5  
6  
7  
2  
9

**Question: -** Can we pass the array in the var-args type argument?

**Answer: -** Yes

```
class Test4
{
 static void m1(int ...x)//vlaid
 {
 for(int i:x)
 {
 System.out.println(i);
 }
 }
 public static void main(String []args)
 {
 m1(new int []{5,6,7,2,9});
 }
}
```

E:\JAVA\Programs\Core Java Program\JDK\Var-args>javac Test4.java

E:\JAVA\Programs\Core Java Program\JDK\Var-args>java Test4

5  
6  
7  
2  
9

[**Note:** - JRE passes the array of string in the argument of the public static void main(String []s) so we can also accept the String array object in var-args type arguments.]

#### CMD

```
D:\java>java Demo
1. JRE started
```

#### JRE

```
String[]
2. Array prepared
main() is invoked
PSVM(String... args)
```

### Var-args type argument for the array

```
class Test5
```

```
{
 static void m1(int []...x) //vleid
 {
 for(int i[:x)
 {
 for(int j:i)
 {
 System.out.println(j);
 }
 }
 }
 public static void main(String ...s)
 {
 int a[]={5, 6, 7};
 int b[]={3,4,5,7,9};
 m1(a, b);
 }
}
```

```
E:\JAVA\Programs\Core Java Program\JDK\Var-args>javac Test5.java
```

```
E:\JAVA\Programs\Core Java Program\JDK\Var-args>java Test5
```

```
5
6
7
3
4
5
7
9
```

### Var-args and Method overloading

#### Exact match VS var-args

```
class Test6
```

```
{
 static void m1(int ...x)//vleid
 {
```

```

 System.out.println("First m1 invoked");
 }
 static void m1(int a, int b)
 {
 System.out.println("Second m1 invoked");
 }
 public static void main(String ...s)
 {
 m1(5,6);
 }
}
E:\JAVA\Programs\Core Java Program\JDK\Var-args>javac Test6.java

E:\JAVA\Programs\Core Java Program\JDK\Var-args>java Test6
Second m1 invoked

```

[Note: - Exact match beats the var-args.]

### **Wedding VS var-args**

```

class Test7
{
 static void m1(int ...x)//v'laid
 {
 System.out.println("First m1 called");
 }
 static void m1(long l)
 {
 System.out.println("Second m1 called");
 }
 public static void main(String ...s)
 {
 m1(5);
 }
}

```

[Note: - Wedding beats var-args.]

### **Static Import**

This is the new feature introduced in the JDK 1.5 according to this feature static member of any class can be imported so that without using the class name directly we can use static member of that class.

```

import static java.lang.System;
class ImportTest
{
 public static void main(String ...args)
 {
 out.println("directly using the output-stream");
 }
}

```

```

 }
}
import static java.lang.system.*;

```

When we use it that it is expended that in the last class name is comes.

### **enum (Enumeration)**

This is the new feature in the JDK 1.5 use to prepare the final static constants.

**enum** is the keyword to declare the enumeration.

After the compilation the .class file of enumeration also will be generated.

All the enumeration is converted into the class after the compilation.

#### **Example:**

```
enum Book{java, oracle sql}
```

1. Save this : - Book.java
2. Compile : - javac Book.java

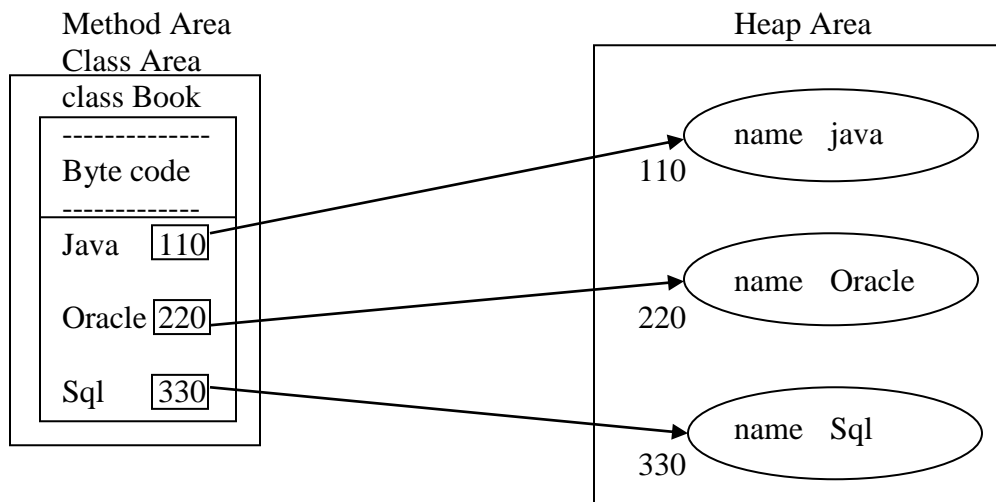
Open the .class file into the cavaj

### **Internal class corresponding to the enumeration**

```

//Internal
public final class Book
{
 static public final Book java = new Book("java");
 static public final Book oracle = new Book("oracle");
 static public final Book sql = new Book("sql");
 String name;
 public Book(String name)
 {
 this.name = name;
 }
 public String toString()
 {
 return(name);
 }
}

```



```
class Student
{
 public static void main(String s[])
 {
 Book b = Book.java;
 System.out.println("Selected Book = "+b);
 }
}
```

```
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum2>javac book.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum2>javac Student.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum2>java Student
Selected Book = java
```

**//Student.java**

```
class Student
{
 String name;
 Book b;
 Student(String name, Book b)
 {
 this.name = name;
 this.b = b;
 }
 public String toString()
 {
 return(name+ ":" +b);
 }
}
```

**//School.java**

```

class School
{
 public static void main(String s[])
 {
 Student st = new Student("Abc",Book.java);
 System.out.println(st);
 }
}
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum3>javac Book.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum3>javac Student.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum3>javac School.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum3>java School
Abc:java

```

### **Customize Enumeration**

```

enum Book
{
 java(550),oracle(450),sql(300);
 int price;
 Book(int price)
 {
 this.price=price;
 }
 int getPrice()
 {
 return(price);
 }
}

```

### **Internally created enumeration class**

```

public final class Book
{
 static public final Book java = new Book("java",550);
 static public final Book oracle = new Book("oracle", 450);
 static public final Book sql = new Book("sql",300);
 String name;
 int price;
 public Book(String name, int price)
 {
 this.name = name;
 this.price = price;
 }
 public String toString()

```

```

 {
 return(name);
 }
 public int getPrice()
 {
 return(price);
 }
 }
}

```

#### **//Student.java**

```

class Student
{
 String name;
 Book b;
 Student(String name, Book b)
 {
 this.name = name;
 this.b = b;
 }
 public String toString()
 {
 return("Name: "+name+"\n Book:"+b+"\n Price :"+b.getPrice());
 }
}

```

#### **//School.java**

```

class School
{
 public static void main(String s[])
 {
 Student st = new Student("Java",Book.java);
 System.out.println(st);
 }
}

```

```

E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum4>javac Book.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum4>javac Student.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum4>javac School.java
E:\JAVA\Programs\Core Java Program\JDK\Enumeration\enum4>java School
Name: Java
Book:java
Price :550

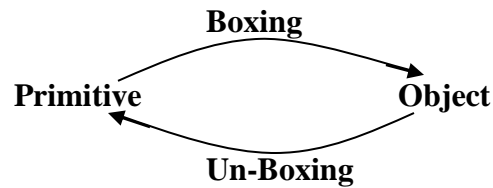
```

### **Wrapper class**

In the java.lang package there are the classes corresponding to each primitive type these class are known as the wrapper classes.

**Ex: -** Integer, Byte, short, long, character, float, double, Boolean.

Converting the primitive type into the object type is known as the **Boxing** and converting the object type into the primitive type known as the **un-boxing**.



### Example of Boxing

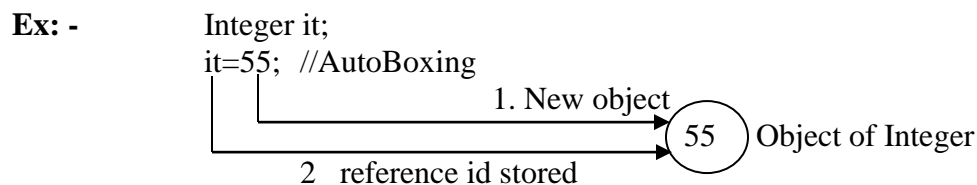
```
int x;
x=10;
Integer it=new Integer(x); //Boxing
```

### Example of Un-Boxing

```
int x;
x=10;
Integer it=new Integer(x); //Boxing
int y;
y=it.intValue(); //Un-Boxing
```

### AutoBoxing

In the JDK 1.5 that is the concept make autoBoxing.  
AutoBoxing means automatic boxing and automatic Un-boxing.



```
Student st;
st=5420; //wrong
```

```
class Test
{
 public static void main(String ...args)
 {
 Integer it;
 it=55;
 }
}
```



### Practical example of the AutoBoxing

```
class Demo
{
 static void m1(Integer it)
 OR
 //static void m1(Object ob)
 OR
 //static void m1(long lg)
 {
 System.out.println("Welcpme in m1");
 }
}
class Test
{
 public static void main(String s[])
 {
 Demo.m1(55);
 }
}
```

2. Reference id passed

1. New Integer

[**Note:** - If we pass any primitive value in any methods argument and there is the reference variable in the parameter in the method so AutoBoxing will be performed.]

### Method overloading with AutoBoxing

#### AutoBoxing VS Exact Match

```
class Demo
{
 static void m1(int x)
 {
 System.out.println("First m1");
 }
 static void m1(Integer it)
 {
 System.out.println("Second m1");
 }
}
class Test
{
 public static void main(String s[])
 {
 Demo.m1(55);
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing2>javac Demo.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing2>javac Test.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing2>java Test
First m1
```

### **AutoBoxing VS var-args**

```
class Demo
{
 static void m1(int ...x)
 {
 System.out.println("First m1");
 }
 static void m1(Integer it)
 {
 System.out.println("Second m1");
 }
}
class Test
{
 public static void main(String s[])
 {
 Demo.m1(55);
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing3>javac Demo.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing3>javac Test.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing3>java Test
Second m1
```

AutoBoxing beats var-args

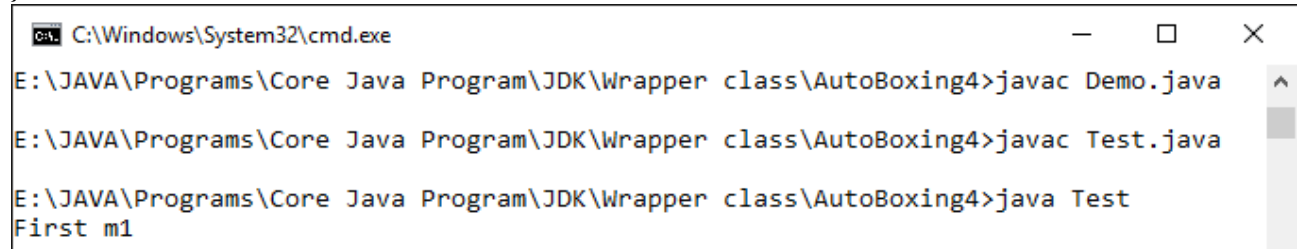
### **AutoBoxing VS widening**

```
class Demo
{
 static void m1(long lg)
 {
 System.out.println("First m1");
 }
 static void m1(Integer it)
 {
 System.out.println("Second m1");
 }
}
```

```

 }
}
class Test
{
 public static void main(String s[])
 {
 Demo.m1(55);
 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing4>javac Demo.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing4>javac Test.java
E:\JAVA\Programs\Core Java Program\JDK\Wrapper class\AutoBoxing4>java Test
First m1

```

AutoBoxing beats widening

## **Constructor of Wrapper classes**

Accept the character class there are two types of constructors in each wrapper class.

1. With the String type
2. With the primitive type

**Ex: -**

```

public Integer(String str)throws NumberFormatException
public Integer(int x)

```

The string type constructor can generate the NumberFormatException

**Ex: -**

```

Integer it1=new Integer("45");
Integer itt2=new Integer("12abc"); //NumberFormatException
Integer it3=new Integer(55); //Primitive type constructors will be used

```

## **Method of Wrapper classes**

- **public static in parseInt(String str)throws NumberFormatException**

This method is used to convert the String type value in the primitive but if in the argumented String there is the non-numeric value then **NumberFormatException** will be generated.

**Ex: -**

```

int x=Integer.parseInt("55");
int y=Integer.parseInt("abc123"); //NumberFormatException

```

- **public int intValue( )**

This method is use to get the value content by the wrapper object.

- **public static Integer valueOf(int val)**

This method will create the wrapper object corresponding to the specified to integer value.

**Ex: -** Integer it=Integer.valueOf(50);

- **public static Integer valueOf(String str)throws NumberFormatException  
//overloaded method**

This method will create and returns the object corresponding to the specified String represents numeric value corresponding to the specified String represent numeric value.

- **public static String toString(int x)**

This method will return the String representation of the argument integer value.

|          |          |                       |
|----------|----------|-----------------------|
| <b>P</b> | <b>→</b> | <b>Primitive</b>      |
| <b>S</b> | <b>→</b> | <b>String</b>         |
| <b>W</b> | <b>→</b> | <b>Wrapper Object</b> |

|          |          |          |                                           |
|----------|----------|----------|-------------------------------------------|
| <b>P</b> | <b>→</b> | <b>S</b> | <b>static toString( )</b>                 |
| <b>P</b> | <b>→</b> | <b>W</b> | <b>valueOf(int ), Integer(int )</b>       |
| <b>S</b> | <b>→</b> | <b>P</b> | <b>parseInt( )</b>                        |
| <b>S</b> | <b>→</b> | <b>W</b> | <b>valueOf(String ), Integer(String )</b> |
| <b>W</b> | <b>→</b> | <b>P</b> | <b>intValue( )</b>                        |
| <b>W</b> | <b>→</b> | <b>S</b> | <b>toString( )</b>                        |

This is the new feature according to that in the inheritance we can override the method by changing their return type as the sub type of parent class method.

**Ex: -**

```
class A
{

}
class B extends A
{
 Dog/m1() //valid override
 {

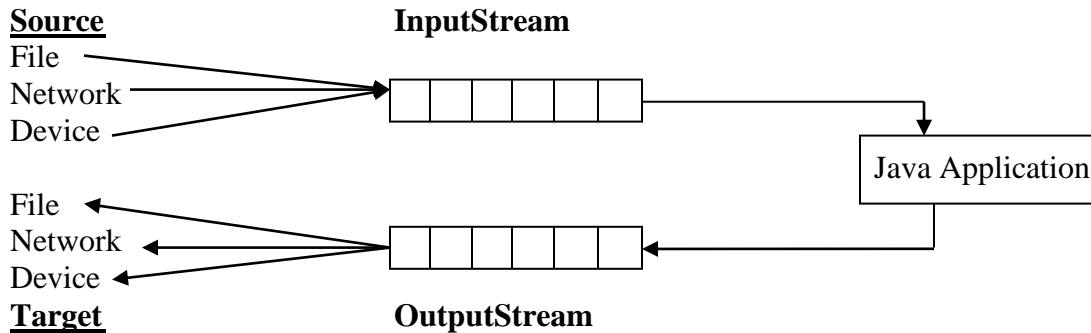
 }
}
```

## Java Input / Output Streams

**Input/Output** in java is totally the stream based.

This stream is the channel used for the transmission.

A data stream is the channel through which data is transmitted between the sources to the destination.



### Benefit of the Stream over the conditional Input / Output

1. **Abstraction**
2. **Flexibility**
3. **Efficiency**
4. **java.io package**

#### 1. **Abstraction**

Stream hides the internal complexity of actual input/output.

#### 2. **Flexibility**

Through the stream we can perform the input/output in the different ways.

#### 3. **Efficiency**

Stream also provides the additional services over the basic input/output.

#### 4. **java.io package**

Provides the classes known as the Stream classes and the instances of these Stream classes represent to the Stream in the java application.

**In the java.io package there are the two types of Stream classes**

1. **byte Stream** : - Follows the 8 bit Sequence
2. **Character Stream** : - Follows the 16 bit Sequence

### InputStream and OutputStream classes

These are the stream classes all the byte type input/output stream classes are the sub class of these two classes.

## **Method of the InputStream classes**

- **public int read()**

This method is to read the next available byte from the Stream.

If all the byte already has been read then this method will return the -1 (-one).

- **public int read(byte []b)**

This method will read all the bytes from InputStraem and copied into the specified byte array.

This method will return the number of bytes it reads.

- **public int available()**

This method returns the number of bytes available in the Stream.

## **Method of the OutpputStream**

- **public void write(int x)**

This method will write the specified byte into the Stream.

- **public void write(byte []b)**

This method will write the argumented type into the stream.

## **Sub classes of InputStream**

- **FileInputStream**
- **BufferedInputStream**
- **DataInputStream**
- **ObjectInputStream**
- **PipedInputStream**
- **ByteArrayInputStream**
- **SequenceInputStream**

## **Sub classes of OutputStream**

- **FileOutputStream**
- **BufferedOutputStream**
- **DataOutputStream**
- **ObjectOutputStream**
- **PipedOutputStream**
- **ByteArrayOutputStream**
- **SequenceOutputStream**

## **FileInputSteram and FileOutputStream classes**

These are the stream classes which are used to perform the Input/Output from the file.

### **Constructor: -**

- **public FileOutputStream(String fname)**

This constructor will always create new file with the specified name.

A file is already exist this constructor will override to that file.

- **public FileOutputStream(String fname, boolean b)**

Boolean argument specifies the append mode that means if it is true then the existing file will not be overridden.


- **public FileInputStream(String fname) throws FileNotFoundException**

This constructor will create the file Input if file doesn't exist then this constructor will generate the Exception.

#### //FileOutputStream.java

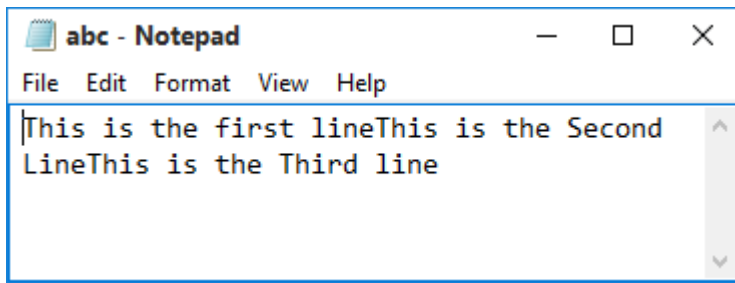
```
import java.io.FileOutputStream;
class FileOutputStreamTest
{
 public static void main(String []args)
 {
 try
 {
 FileOutputStream fout=new FileOutputStream("abc.txt");
 String str[]={ "This is the first line", "This is the Second Line", "This is the
Third line"};

 byte b[]=str[0].getBytes();
 fout.write(b);
 fout.flush();
 b = str[1].getBytes();
 fout.write(b);
 fout.flush();
 b = str[2].getBytes();
 fout.write(b);
 fout.flush();
 fout.close();
 System.out.println("Content stored into the File");
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

 C:\Windows\System32\cmd.exe

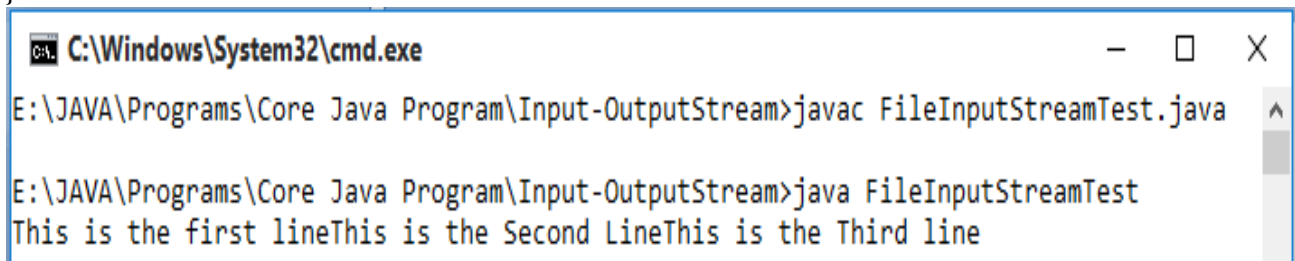
E:\JAVA\Programs\Core Java Program\Input-OutputStream>javac FileOutputStreamTest.java

E:\JAVA\Programs\Core Java Program\Input-OutputStream>java FileOutputStreamTest  
Content stored into the File



### //FileInputStreamTest.java

```
import java.io.FileInputStream;
class FileInputStreamTest
{
 public static void main(String []args)
 {
 try
 {
 FileInputStream fin=new FileInputStream("abc.txt");
 while(true)
 {
 int x=fin.read();
 if(x == -1)
 break;
 System.out.print((char)x);
 }
 fin.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```



Performance byes this program is slow because the loop will iterate up to the number of byte are available in the stream.

### Assignment:

Read the bytes from the file and count how many alphabets, digits, space and special symbol are exists.



## //Count.java

// Read the bytes from the file and count how many alphabets, digits, space and special symbol are exist.

import java.io.\*;

class Assignment

```
{
 public static void main(String s[])
 {
 int vc= 0, dc = 0, cc = 0, sc=0, oc=0, alphabates;
 try
 {
 FileInputStream fin = new FileInputStream("abc.txt");
 while(true)
 {
 int x= fin.read();
 if(x == -1)
 {
 break;
 }
 if((x>=65 && x<=90)||(x>=97 && x<=122))
 {
 if(x=='A' || x=='a' || x=='E' || x=='e' || x=='O' || x=='o' ||
x=='I' || x=='i' || x=='U' || x=='u')
 {
 vc++;
 }
 else
 {
 cc++;
 }
 }
 if(x>=48 || x<=57)
 {
 dc++;
 }
 else if(x==32)
 {
 sc++;
 }
 else
 {
 oc++;
 }
 }
 alphabates = vc + cc;
 System.out.println("Vowels are: "+vc);
 System.out.println("Consonants are: "+cc);
 System.out.println("Digits are: "+dc);
 }
 }
}
```

```

 System.out.println("Spaces are: "+sc);
 System.out.println("Special symbols are: "+oc);
 System.out.println("Alphabates are: "+alphabates);
 fin.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

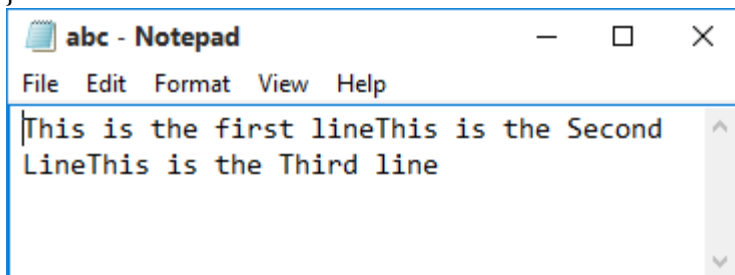
```

### **//FileInputStream1.java**

```

import java.io.FileInputStream;
class FileInputStreamTest1
{
 public static void main(String []args)
 {
 try
 {
 FileInputStream fin=new FileInputStream("abc.txt");
 int size=fin.available();
 byte b[]=new byte[size];
 fin.read(b);
 //byte are copied into the byte[] from the stream
 String str=new String(b);
 System.out.println(str);
 fin.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-OutputStream>javac FileInputStreamTest1.java
E:\JAVA\Programs\Core Java Program\Input-OutputStream>java FileInputStreamTest1
This is the first lineThis is the Second LineThis is the Third line
```

## **Character Stream**

Reader and Writer classes are the top most classes for all the Character Stream.

### **There sub classes as follows**

#### **Reader**

- **FileReader**
- **BufferedReader**
- **InputStreamReader**
- **CharArrayReader**

#### **Writer**

- **FileWriter**
- **BufferedWriter**
- **OutputStreamWriter**
- **CharArrayWriter**
- **PrintWriter**

### **Method of the Reader class**

- **public int read()**

This method will read the ASCII code of the next available character.  
It returns -1 if all the character is already being repeat

- **public int read(char ch[ ])**

This method will read all the character from the stream into the specified character array and return the number of character returned.

[Note: - There is no available method]

### **Method of the Writer class**

- **public void write(int ch)**

This method will write the specified character into the stream in the form of ASCII code.

- **public void write(char [ ]ch)**

This method will write the specified character array into the stream.

- **public void write(String str)**

This method will write the character of the specified stream into the stream.

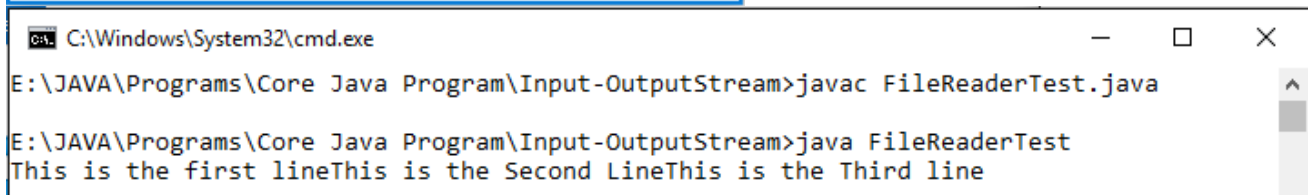
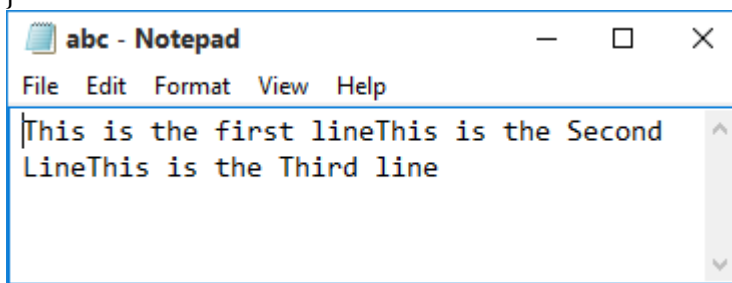
## FileReader and FileWriter classes

These classes are the character stream version of the FileInputStream and FileOutputStream.

**Ex: -** To read the File context from the FileReader.

**//FileReaderTest.java**

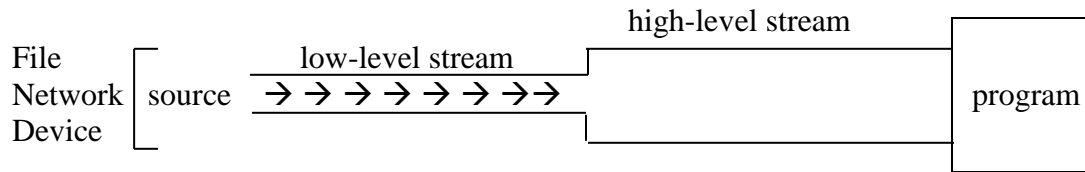
```
import java.io.*;
class FileReaderTest
{
 public static void main(String []s)
 {
 try
 {
 FileReader fr=new FileReader("abc.txt");
 while(true)
 {
 int x=fr.read();
 if(x==-1)
 break;
 System.out.print((char)x);
 }
 System.out.println();
 fr.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```



## **BufferedReader and BufferedWriter classes**

These classes are the high-level streams.

A high-level stream is the stream that never contents directly to any file, device and network rather they always be connected with the low-level stream or other high-level stream.



The purposes of high level stream are to provide the some additional functionality over the normal input output.

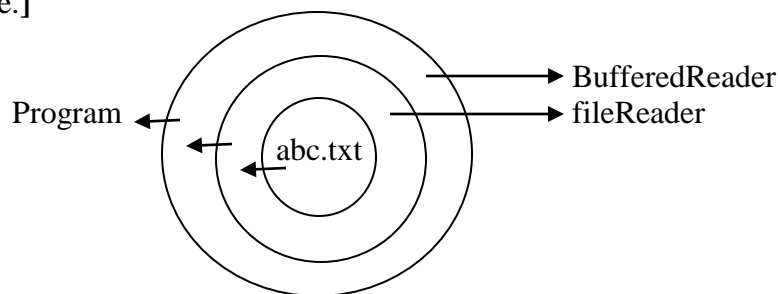
High level stream are the external layers for the Input/Output.

**BufferedReader** provides the method to read the control from the source line by line.

### **Method of BufferedReader**

- **public String readLine() throws IOException**

[**Note:** - All the methods in the Input Output API throws the IOException to read the control form the file line by line.]



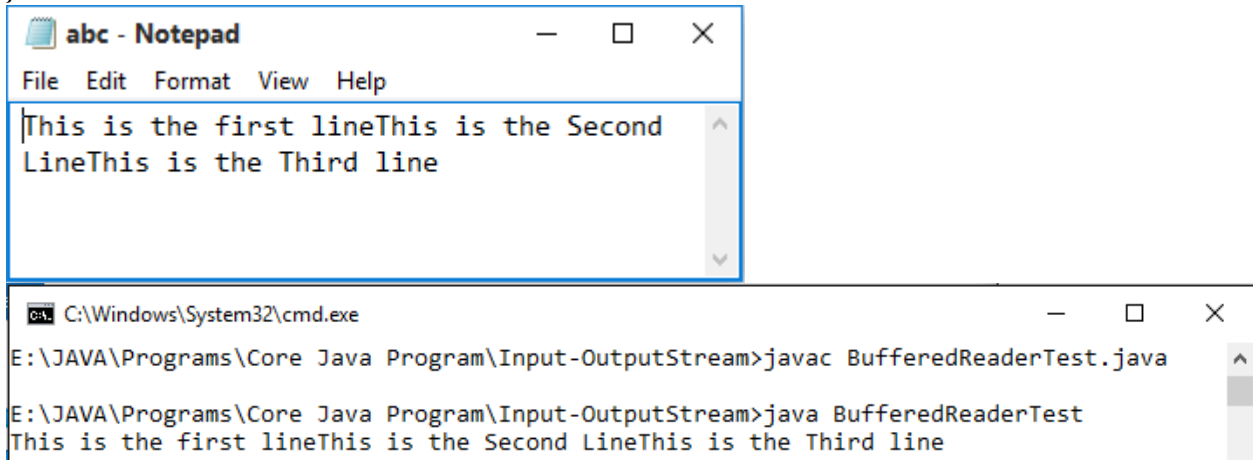
### **//BufferedReaderTest.java**

```
import java.io.*;
class BufferedReaderTest
{
 public static void main(String []s)
 {
 try
 {
 FileReader fr=new FileReader("abc.txt");
 BufferedReader br=new BufferedReader(fr);
 while(true)
 {
 String str=br.readLine();
 if(str==null)
 break;
 System.out.print(str);
 }
 System.out.println();
 }
 }
}
```

```

 fr.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}

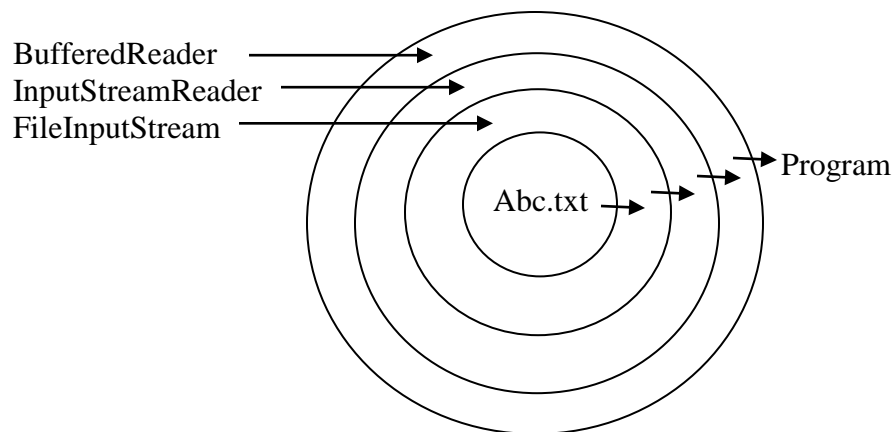
```



## InputStreamReader and OutputStreamReader classes

**InputStreamReader** is used to convert the byte stream to character stream and **OutputStreamWriter** class is used to the character stream into byte stream.

To read the file content through the **FileInputStream** line by line.



```

FileInputStream fin=new FileInputStream("abc.txt");
BufferedReader br=new BufferedReader(new InputStreamReader(fin));

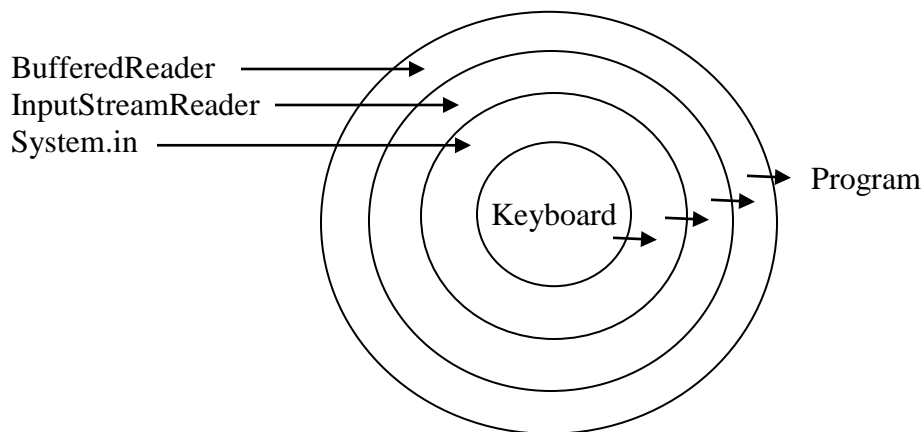
```

At the many places there are the readymade byte type streams and to read the content line by line from there we have to connect them with the BuffereReader via InputStreamReader.

**Ex: - System.in**

Instance of the InputStream(Byte type Stream) connected with the keyboard

## Reading the content line by line from the keyboard



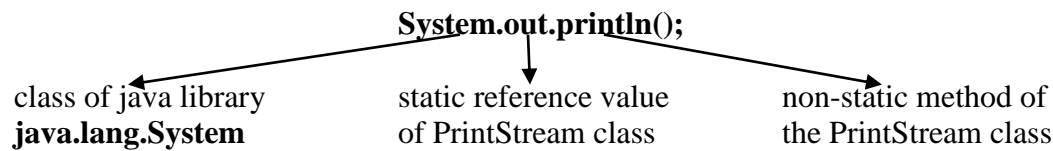
**//SystemInTest.java**

```
import java.io.*;
class SystemInTest
{
 public static void main(String [] args)
 {
 try
 {
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
 System.out.println("Enter the number");
 String str=br.readLine();
 System.out.println(str);
 System.out.println("Hello "+str);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-OutputStream\SystemStream>javac SystemInTest.java
E:\JAVA\Programs\Core Java Program\Input-OutputStream\SystemStream>java SystemInTest
Enter the number
Ducat
Ducat
Hello Ducat
```

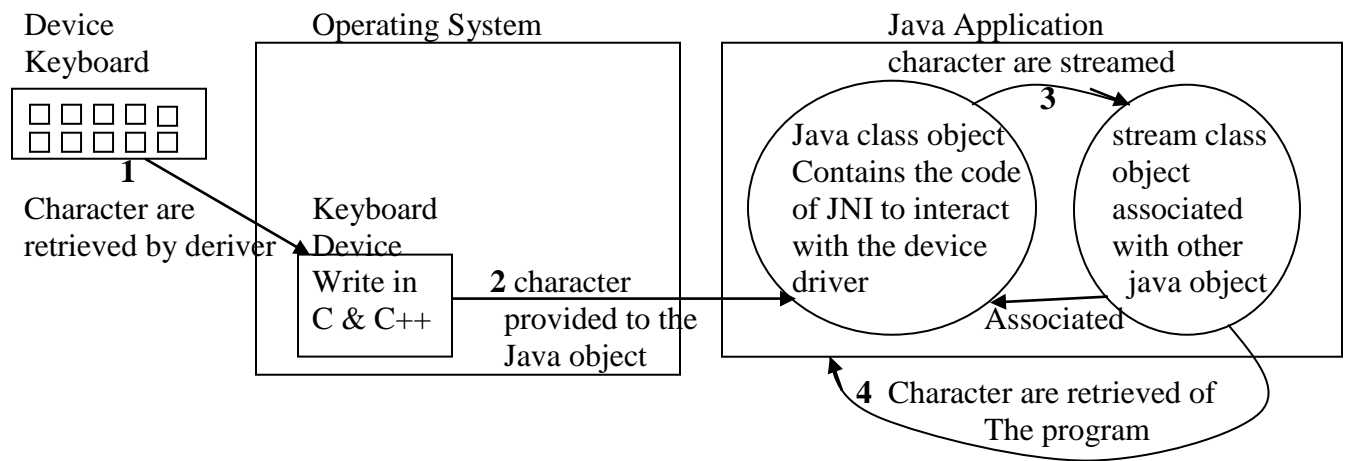
**[Note: - There are three streams already prepared in the System class statically.]**

1. **System.out** [Object of **PrintStream**]
2. **System.err** [Object of **PrintStream**]
3. **System.in** [Object of **PrintStream**]



class final class **System**

```
{
 public static final PrintStream out = null;
 public static final PrintStream err = null;
 public static final PrintStream in = null;
}
```



There are the static methods in the **System** class by using that the default readymade stream can be modified that means these stream can be connected other device, file or network.

### Method of the **System** class

- **public static void setOut(PrintStream out)**
- **public static void setErr(PrintStream err)**
- **public static void setIn(InputStream in)**

### //SystemStreamChangeTest.java

```
import java.io.*;
class SystemStreamChangeTest
{
 public static void main(String []s)
 {
 try
 {
 FileInputStream fin = new FileInputStream("file1.txt");
 PrintStream out = new PrintStream("file2.txt");
```






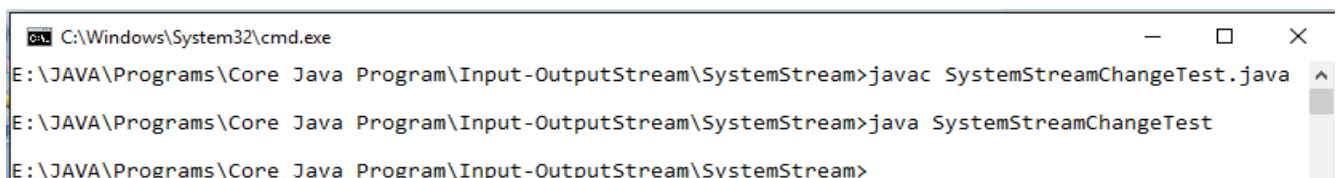
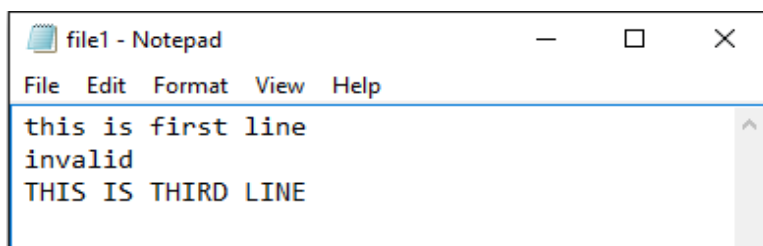
```

 PrintStream err = new PrintStream("fileErr.txt");
 //System.in = fin; //Error
 System.setIn(fin);
 System.setOut(out);
 System.setErr(err);
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 String str;
 while((str == br.readLine()) != null)
 {
 if(str.length()<10)
 System.err.println(str);
 else
 System.out.println(str);
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}






```

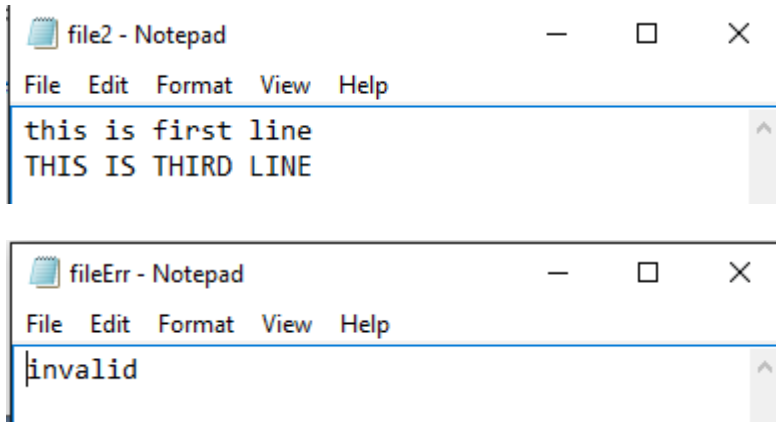
#### Folder Structure

| Name                                                                                                             | Date modified      | Type          | Size |
|------------------------------------------------------------------------------------------------------------------|--------------------|---------------|------|
|  file1                        | 23-Feb-16 12:18 AM | Text Document | 1 KB |
|  SystemStreamChangeTest.class | 22-Jun-16 10:06 AM | CLASS File    | 2 KB |
|  SystemStreamChangeTest       | 23-Feb-16 12:21 AM | JAVA File     | 1 KB |



After run the program file2 and fileErr is created. Folder structure

| Name                                                                                                           | Date modified      | Type          | Size |
|----------------------------------------------------------------------------------------------------------------|--------------------|---------------|------|
|  file1                        | 23-Feb-16 12:18 AM | Text Document | 1 KB |
|  file2                        | 22-Jun-16 10:09 AM | Text Document | 1 KB |
|  fileErr                      | 22-Jun-16 10:09 AM | Text Document | 1 KB |
|  SystemStreamChangeTest.class | 22-Jun-16 10:08 AM | CLASS File    | 2 KB |
|  SystemStreamChangeTest       | 23-Feb-16 12:21 AM | JAVA File     | 1 KB |



## DataInputStream and DataOutputStream

These are the high level stream and provide a way to read and write the direct primitive in the streams.

### Method of DataOutputStream

- **public void writeInt(int x)**
- **public void writeChar(char ch)**
- **public void writeFloat(float f)**
- **public void writeUTF(String str)**

### Method of DataInputStream

- **public int reading()**
- **public char readChar()**
- **public float readFloat()**
- **public String readUTF()**

If all the data has been read then the read() method will generate the **EOFException**.

### //DataIOTest.java

```
import java.io.*;
class DataIOTest
{
 public static void main(String s[])
 {
 try
 {
```

```

 FileOutputStream fout=new FileOutputStream("student.txt");
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
 DataOutputStream dout=new DataOutputStream(fout);
 while(true)
 {
 System.out.println("Entre the name, rollno, marks and grade");
 String name=br.readLine();
 int rollno=Integer.parseInt(br.readLine());
 float marks=Float.parseFloat(br.readLine());
 char grade=br.readLine().charAt(0);
 dout.writeUTF(name);
 dout.writeInt(rollno);
 dout.writeFloat(marks);
 dout.writeChar(grade);
 dout.flush();
 System.out.println("do u want to store more records(y/n)");
 String option=br.readLine();
 if(option.equals("n"))
 break;
 }
 dout.close();
 FileInputStream fin=new FileInputStream("student.txt");
 DataInputStream din=new DataInputStream(fin);
 System.out.println("\n\n entered records as follows:");
 while(true)
 {
 try
 {
 String name=din.readUTF();
 int rollno=din.readInt();
 float marks=din.readFloat();
 char grade=din.readChar();

 System.out.println(name+"\t"+rollno+"\t"+marks+"\t"+grade);
 }
 catch(Exception e)
 {
 break;
 }
 }
 din.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
} // main method close
}

```

## **ByteArrayInputStream and ByteArrayOutputStream**

These Stream facilitated for the development for the InputStream and OutputStream for the other modules for the program

### **Constructor**

- **public ByteArrayInputStream(byte [ ]b)**

This constructor to get the InputStream object and the specified byte Array works as the buffer.

### **Method of ByteArrayInputStream**

- **public void read(byte [ ]b)**

This method will read the byte from the buffer of this ByteArrayInputStream and writes the bytes into the specify array.

### **Constructor**

- **public ByteArrayOutputStream(byte [ ]b)**

### **Method of ByteArrayOutputStream**

- **public void write(byte [ ]b)**

This method will write the bytes into this OutputStream from the specified byte array.

- **public void writeTo(OutputStream out)**

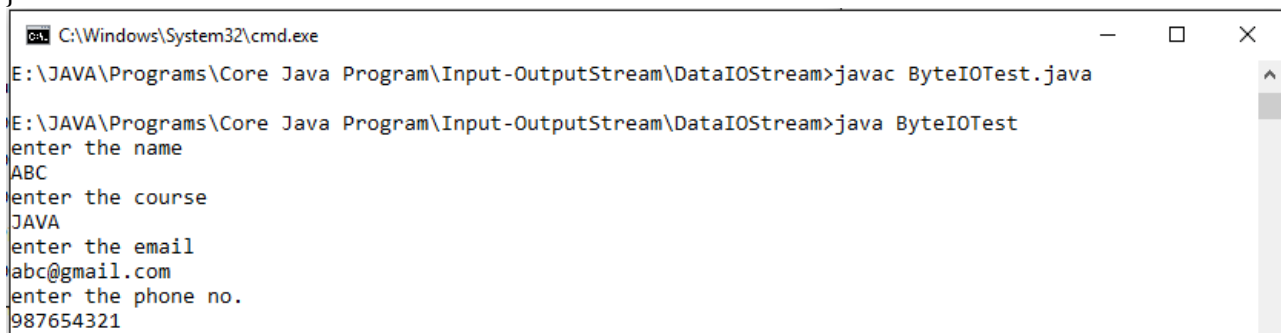
This method will write bytes into the specified OutputStream from this ByteArrayOutputStream.

```
// MyReader.java
import java.io.*;
class MyReader
{
 static InputStream myRead(String msg)
 {
 try
 {
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
 System.out.println(msg);
 String str=br.readLine();
 byte b[]=str.getBytes();
 ByteArrayInputStream in=new ByteArrayInputStream(b);
 // bytes are loaded into BArrayInputStream
 return(in);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 return(null);
 }
}
```

```

}
// Main class ByteIOTest.java
class ByteIOTest
{
 public static void main(String s[])
 {
 try
 {
 FileOutputStream fout=new FileOutputStream("stud.txt");
 ByteArrayOutputStream bout=null;
 String msgs[]={ "enter the name","enter the course","enter the
email","enter the phone no."};
 int i=0;
 while(i<msgs.length)
 {
 InputStream in = MyReader.myRead(msgs[i]);
 byte b[]=new byte[in.available()];
 in.read(b);
 // bytes are copied from the InputStream into the b[]
 bout = new ByteArrayOutputStream();
 bout.write(b);
 // bytes copied into ByteArrayOutputStream from byte[]
 bout.writeTo(fout);
 bout.flush();
 // bytes are copied into the FileOutputStream
 fout.write("\n".getBytes());
 fout.flush();
 i++;
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

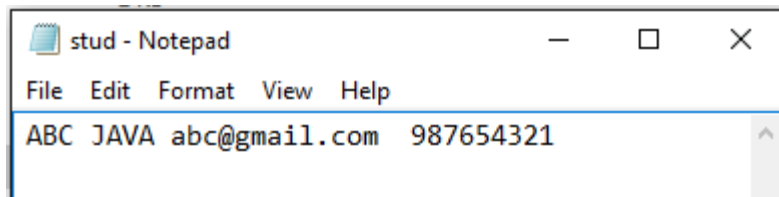
```



```

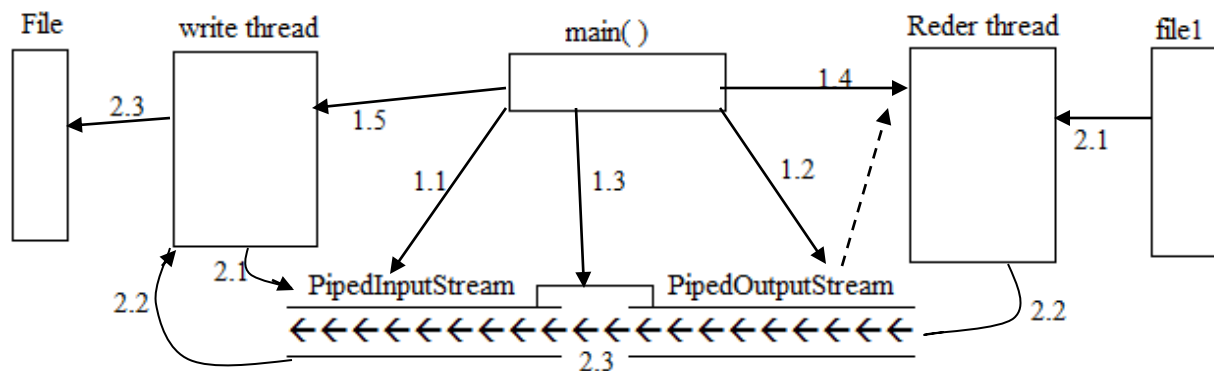
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-OutputStream\DataIOStream>javac ByteIOTest.java
E:\JAVA\Programs\Core Java Program\Input-OutputStream\DataIOStream>java ByteIOTest
enter the name
ABC
enter the course
JAVA
enter the email
abc@gmail.com
enter the phone no.
987654321

```



## PipedInputStream and PipedOutputStream classes

These streams always are used in the multithreading environment before using these streams they have to be connected with each other when ever any content is written in the PipedInputStream then that content automatically moved into the PipedOutputStream.



## **Working of PipedInput/OutputStream in multithreaded environment**

- 1.1. PipedOutputStream object created.
- 1.2. PipedInputStream object Created.
- 1.3. Stream are connected each other.
- 2.1. File content read. (ReaderThread side)
- 2.2. File content writer in the PipedOutputStream.
- 2.3. File content moved towards PipedInputStream. (WriterThread side)

- 2.1. Read method invoked.

If the bytes are already written in the output stream then retrieved otherwise wait for the loading of bytes.

- 2.2. Bytes returned.
- 2.3. Bytes are written into the file.

### **//PipedIOTest.java**

```
import java.io.*;
class ReaderThread extends Thread
{
 String fname;
 PipedOutputStream pout;
 ReaderThread(String fname,PipedOutputStream pout)
 {
 this.fname=fname;
```

```

 this.pout=pout;
 start();
 }
 public void run()
 {
 try
 {
 FileInputStream fin=new FileInputStream(fname);
 BufferedReader br=new BufferedReader(new InputStreamReader(fin));
 PrintWriter out=new PrintWriter(pout);
 System.out.println("reader thread started");
 while(true)
 {
 String str = br.readLine();
 if(str==null)
 break;
 out.println(str);
 out.flush();
 }
 System.out.println("data is copied into the pipe");
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
class WriterThread extends Thread
{
 String fname;
 PipedInputStream pin;
 WriterThread(String fname,PipedInputStream pin)
 {
 this.fname=fname;
 this.pin=pin;
 start();
 }
 public void run()
 {
 try
 {
 PrintWriter out=new PrintWriter(fname);
 BufferedReader br=new BufferedReader(new InputStreamReader(pin));
 System.out.println("writer thread started");
 while(true)
 {
 try

```

```

 {
 String str=br.readLine();
 out.println(str);
 out.flush();
 }
 catch(Exception e)
 {
 break;
 }
 }
 System.out.println("contents are copied into file");
}
catch(Exception e)
{
 e.printStackTrace();
}
}
}
class PipedIOtest
{
 public static void main(String s[])throws IOException
 {
 PipedInputStream pin=new PipedInputStream();
 PipedOutputStream pout=new PipedOutputStream(pin);
 ReaderThread rt=new ReaderThread(s[0],pout);
 WriterThread wt=new WriterThread(s[1],pin);

 }
}

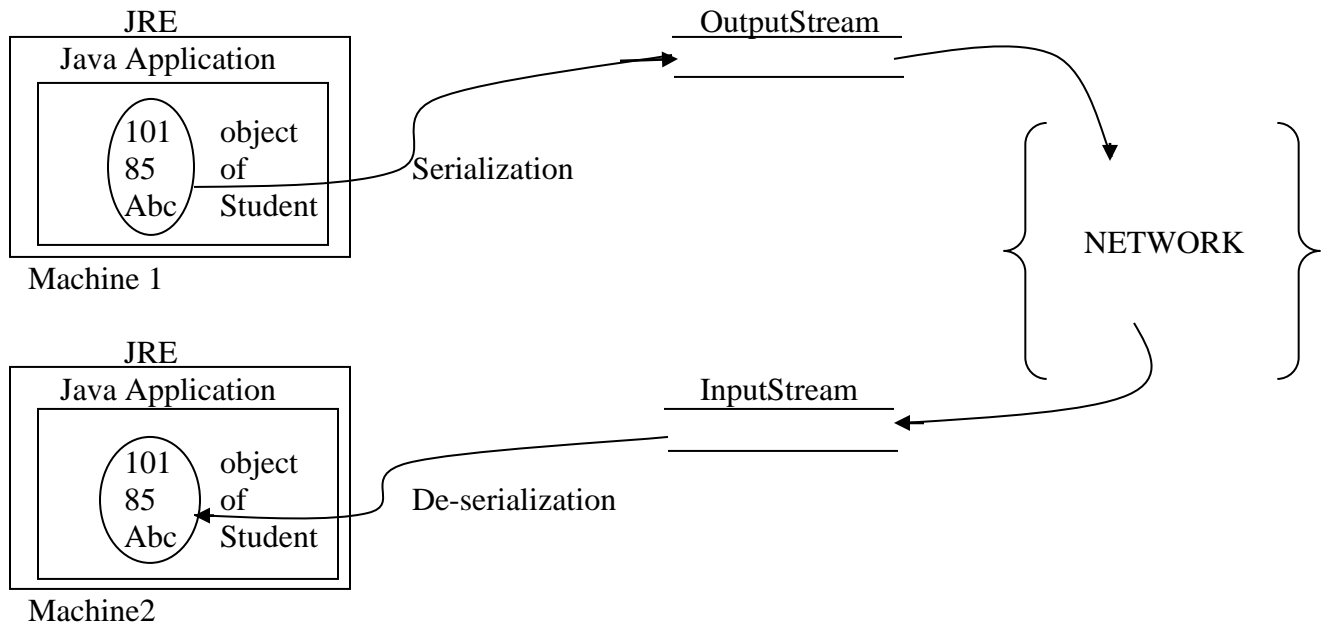
```



## Serialization

It is the way to write the objects into the stream in such a way that objects further can be reconstructed.

Whenever the object is written onto the stream in actual the object data is written along with the class information.



By default in java objects are not capable to be serialization only the objects that implements to the **java.io.Serializable** interface can only be written onto the outputStream.

**Java.io.Serializable** interface is the marker interface.

A marker interface is the interface that does not have any method they are just used to mark the class for special task.

### //Student.java

```
import java.io.*;
class Student implements Serializable
{
 int rollno,marks;
 String name;
 Student(String name,int rollno,int marks)
 {
 this.name = name;
 this.rollno = rollno;
 this.marks = marks;
 }
 public String toString()
 {
 return(name+" "+rollno+" "+marks);
 }
}
```

## ObjectInputStream and ObjectOutputStream class

These classes are use to write and read the objects onto the stream.  
These are high level streams

- **public void writerObject(Object ob)**

**writeObject( )** of ObjectOutputStream class is use to write the object onto the stream.

- **public void readObject( )**

**readerObject( )** of the ObjectInputStream class is use to read the object from the stream.

It generate end of file exception if when all the objects have been read.

**// SerializableTest.java**

```
import java.io.*;
class SerializableTest
{
 public static void main(String s[])
 {
 try
 {
 FileOutputStream fout = new FileOutputStream("Student.txt");
 ObjectOutputStream out = new ObjectOutputStream(fout);
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 while(true)
 {
 System.out.println("Enter the name, rollno, marks");
 String name = br.readLine();
 int rollno = Integer.parseInt(br.readLine());
 int marks = Integer.parseInt(br.readLine());
 Student st = new Student(name,rollno,marks);
 out.writeObject(st);
 out.flush();
 System.out.println("do you want to insert more record(y/n)");
 String option = br.readLine();
 if(option.equals("n"))
 break;
 }
 fout.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>javac SerializableTest.java

E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>java SerializableTest
Enter the name, rollno, marks
abc
101
85
do you want to insert more record(y/n)
n

```

### // DeserializableTest.java

```

import java.io.*;
class DeserializableTest
{
 public static void main(String s[])
 {
 try
 {
 FileInputStream fin = new FileInputStream("Student.txt");
 ObjectInputStream in = new ObjectInputStream(fin);
 while(true)
 {
 try
 {
 Student st = (Student)in.readObject();
 System.out.println(st);
 }
 catch(Exception e)
 {
 //e.printStackTrace();
 break;
 }
 }
 in.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>javac DeserializableTest.java

E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>java DeserializableTest
abc 101 85

```

## **Transient Keyword**

This keyword can be used with the non-static variable the serialization value of transient type can never be written onto the stream during the serialization.

### **// SerializableTest1.java**

```
import java.io.*;
class SerializableTest1
{
 public static void main(String s[])
 {
 try
 {
 FileOutputStream fout = new FileOutputStream("Student.txt");
 ObjectOutputStream out = new ObjectOutputStream(fout);
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 while(true)
 {
 System.out.println("Enter the name, rollno, marks");
 transient String name;
 int rollno = Integer.parseInt(br.readLine());
 int marks = Integer.parseInt(br.readLine());
 Student st = new Student(name,rollno,marks);
 out.writeObject(st);
 out.flush();
 System.out.println("do you want to insert more record(y/n)");
 String option = br.readLine();
 if(option.equals("n"))
 break;
 }
 fout.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

**Question:- why primitive data is not in the readable form.**

**Answer: -** Because data save in RAM as image either binary format, image or any text format, image binary format image is not in the readable form.

### **Signature of ReadObject()**

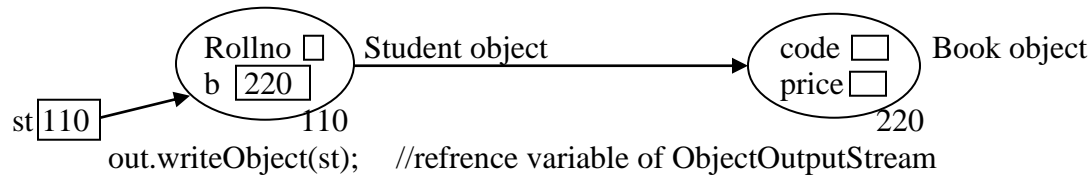
- **public Object ReadObject()**
- **public void writeObject(Object ob)**

When append mode is true in case of externalization then same file is override that means is written is same file.

In case of serialization we cannot append the objects in the same file or existing file if we do so then at the time of retrieval of objects exception stream corrupted will be generated.

[**Note:** - Transient keyword and static variables both can never be written / stored onto the stream.]

[**Note:** - Serialization with the association in case of association when never we write the composite object into the stream along with that the numbers of object also will be written in the stream.]



[**Note:** - The invocation of writeObject() method will write the data of student object as well as book object.]

[**Note:** - The book object also must be the type of serialization.]

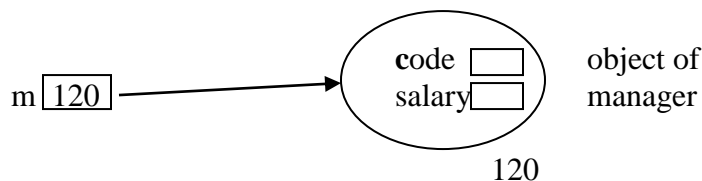
### Serialization in case of inheritance

If we write the child class object into the stream then that child class must be declared as serializable class and in that case the parent class need not be declared as serializable class.

```

class Employee
{
 int code;
}
class Manager extends Employee implements serializable
{
 int salary;
}

```



Non-static data members of parent class can also stored into the child object.

out.writeObject(m);  
 ↗ Reference variable of manager  
 ↘ Reference variable of ObjectOutputStream

### Externalization

It is enhancement version of serialization through the externalization we can customize the process of serialization.

### **java.io.Externalization(I)**

This is the child interface of serialization interface this interface provides two methods.

1. public void writeExternal(ObjectOutput obj)
2. public void readExternal(ObjectInput in)

These are the callback() method that automatically invoked when the input from the readObject() and writeObject() of the ObjectInputStream and ObjectOutputStream.

### **//ExternalizableTest.java**

```
import java.io.*;
class Student implements Externalizable
{
 String name;
 int rollno, marks;
 public Student()
 {
 }
 Student(String name, int rollno, int marks)
 {
 this.name = name;
 this.rollno = rollno;
 this.marks = marks;
 }
 public void readExternal(ObjectInput in)
 {
 try
 {
 name = (String)in.readObject();
 rollno = in.readInt();
 marks = in.readInt();
 name = name.toUpperCase();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
 public void writeExternal(ObjectOutput out)throws IOException
 {
 out.writeObject(name);
 out.writeInt(rollno);
 if(marks>=35 && marks<40)
 marks+=5;
 out.writeInt(marks);
 }
}
```

```

 public String toString()
 {
 return(rollno+" "+name+" "+marks);
 }
 }
 class ExternalizableTest
 {
 public static void main(String s[])
 {
 try
 {
 FileOutputStream fout = new FileOutputStream("Student.txt");
 ObjectOutputStream out = new ObjectOutputStream(fout);
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 while(true)
 {
 System.out.println("Enter the name, rollno, marks");
 String name = br.readLine();
 int rollno = Integer.parseInt(br.readLine());
 int marks = Integer.parseInt(br.readLine());
 Student st = new Student(name,rollno,marks);
 out.writeObject(st);
 out.flush();
 System.out.println("do you want to insert more record(y/n)");
 String option = br.readLine();
 if(option.equals("n"))
 break;
 }
 fout.close();
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>javac ExternalizableTest.java
E:\JAVA\Programs\Core Java Program\Input-Output\Serialization>java ExternalizableTest
Enter the name, rollno, marks
abc
101
86
do you want to insert more record(y/n)
y
Enter the name, rollno, marks
xyz
102
85
do you want to insert more record(y/n)
n

```

Main method return same as in serialization

```

 Out.writeObject();
 writeObject()
 st.public void writeExternal();
 writeExternal() invoked
 with the object of student

```

### Differences between the serialization and externalization

| Serialization                                                      | Externalization                                                  |
|--------------------------------------------------------------------|------------------------------------------------------------------|
| 1. Achieved through the <b>java.io.Serializable Interface</b>      | 1. Achieved through the <b>java.io.Externalization Interface</b> |
| 2. It is the marker interface.                                     | 2 It is not the marks interface.                                 |
| 3. Read operation uses same constructor as in the write operation. | 3 The default constructor is required during the read operation. |

### Java.io.File class

An instance of file class represents to the file exists in the file system.

#### Constructor

- **public File(String FileName)**

This constructor will create the file object that will represent to the specified file in the file system.

[**Note:** -Any file object can also represents to the directory in the file system.]

- **public File(File parent, String FileName)**

This constructor will take another file instance that represents to the directory.  
This file accepts will represent to the specified file in the specified directory.

### Method of the File class



- **public boolean createNewFile( )**

This method will create a new file with the name contained by the invoking file object.

This method always creates a new file.

If file is created successfully then this method returns true otherwise false.

(Protected file, disk size full-filed)

- **public boolean mkdir( )**

This method will create the new directory with the name contained by invoking file object.

- **public boolean isFile( )**

It returns true if the invoking file object represents to the file.

- **public boolean delete( )**

It will delete file from the file system.

In case of the directory must be empty.

- **public boolean renameTo(File f)**

It will rename this file with the name of specified file.

- **public File[] listFiles( )**

This method returns the array of the objects that represents to the file and subdirectory continued by this directory.

[Note: - This method can only be invoked when this file object represents to the directory.]

- **public String[] list( )**

This method also will be invoked when the file object represents to the directory.

It returns the name of the file and subdirectory contained by this directory.

- **public boolean exists( )**

It returns true if the file name or directory name contained by this object is physically exists in the file system.

**//FileTest.java**

```
import java.io.*;
```

```
class FileTest
```

```
{
```

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

```
 {
```

```
 try
```

```
 {
```

```
 File f1 = new File("abc.txt");
```

```
 //relative path
```

```
 File f2 = new File("E:\\JAVA\\Programs\\Core Java Program\\Input-Output\\FileClass\\test\\xyz.txt");
```

```

 //absolute path
 System.out.println("is abc.txt is exist: "+f1.exists());
 System.out.println("is xyz.txt is exist: "+f2.exists());
 if(!f1.exists())
 {
 System.out.println("create abc.txt: "+f1.createNewFile());
 }
 System.out.println("creating xyz.txt: "+f2.createNewFile());
 File f3 = new File("test");
 System.out.println("creating test dir: "+f3.mkdir());
 File f4 = new File(f3,"pqr.txt");
 System.out.println("creating the pqr.txt in test dir:
"+f4.createNewFile());
 File f5 = new File("mno.txt");
 System.out.println("creating mno.txt: "+f5.createNewFile());
 File f6 = new File("jkl.txt");
 f5.renameTo(f6);
 System.out.println("file name is changed, mno.txt to jkl.txt");
 System.out.println("deleting the abc.txt: "+f1.delete());
 System.out.println("deleting the test dir: "+f3.delete());
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\FileClass>javac FileTest.java
E:\JAVA\Programs\Core Java Program\Input-Output\FileClass>java FileTest
is abc.txt is exist: true
is xyz.txt is exist: true
creating xyz.txt: false
creating test dir: false
creating the pqr.txt in test dir: false
creating mno.txt: true
file name is changed, mno.txt to jkl.txt
deleting the abc.txt: true
deleting the test dir: false

```

### //FileTest.java

```

import java.io.*;
class FileTest2
{
 public static void main(String s[])
 {
 try

```

```

 {
 File dir = new File("E:\\JAVA\\Programs\\Core Java Program\\Input-
Output\\FileClass\\test");
 File f[] = dir.listFiles();
 for(int i = 0; i<f.length; i++)
 {
 if(f[i].isFile())
 System.out.println("File--");
 else
 System.out.println("Dir--");
 System.out.println(f[i].getName());
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```

### **Java.lang.Process class**

An instance of this class represents to the running process in the operating system. Through the object of process class the output of the process can be taken in the java application and also we can provide input to the process.

Output of the process will be input of our java application and input to the process will be the output of our java application.

### **Method of Process**

- **public InputStream getInputStream( )**

This method returns the readymade InputStream object through which the output of the process can be obtained.

- **public InputStream getErrorStream( )**

This method returns the object of InputStream through which the error message of the running process can be obtained.

- **public OutputStream getOutputStream( )**

This method will returns the OutputStream object through which the input to the process can be passed.

### **Java.lang.Runtime class**

An instance of that class in java application is used for the interaction between java Runtime environment and java application.

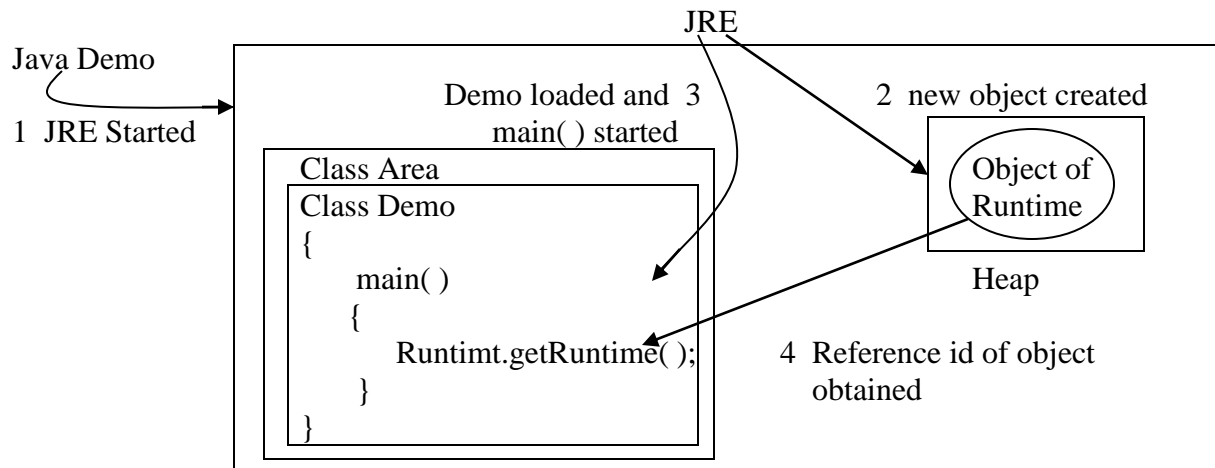
There is only instance created by the JRE of the runtime class.

Factory method of runtime class is used to provide the reference id of the runtime class object.

- **Public static Runtime getRuntime( )**

**Ex:**

```
Runtime r = Runtime.getRuntime();
```

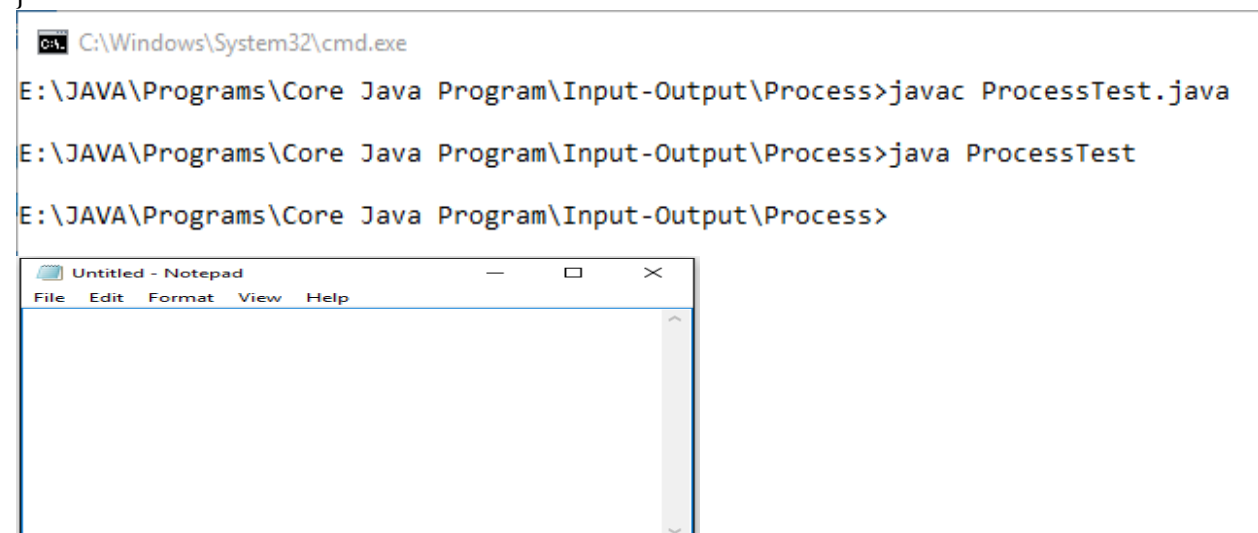


[**Note:** - `exec( )` method of `Runtime` class is use to start a new process in the Operating System.]

- **public Process exec(String exeName)**

**//ProcessTest.java**

```
import java.io.*;
class ProcessTest
{
 public static void main(String s[])throws IOException
 {
 Runtime r = Runtime.getRuntime();
 Process p = r.exec("notepad.exe");
 }
}
```



To compile another file

```
Process p = r.exec("javac Test.java");
```

[**Note:** - Getting the output of the process and also getting the error message of the process.]

**WaitFor()** → put program to hold until the previous program complete.

**Ex:**

**//Power.java**

```
import java.util.*;
class Power
{
 public static void main(String s[])
 {
 int a=2,b=3;
 System.out.println("p : - "+Math.pow(a, b));
 }
}
```

**//ProcessTest1.java**

```
import java.io.*;
class ProcessTest1
{
 public static void main(String s[])throws IOException,InterruptedException
 {
 Runtime r = Runtime.getRuntime();
 Process p1 = r.exec("javac Power.java");
 //Process p = r.exec("java Power");
 p1.waitFor();
 InputStream err = p1.getErrorStream();
 byte b[] = new byte[err.available()];
 err.read(b);
 String str = new String(b);
 System.out.print(str);
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Process>javac ProcessTest1.java
E:\JAVA\Programs\Core Java Program\Input-Output\Process>java ProcessTest1
E:\JAVA\Programs\Core Java Program\Input-Output\Process>java Power
p8.0
```

[**Note:** - Put Power.java in current directory.]

WaitFor( ) will hold the execution of the program till the specified process is not completed.

### Signature

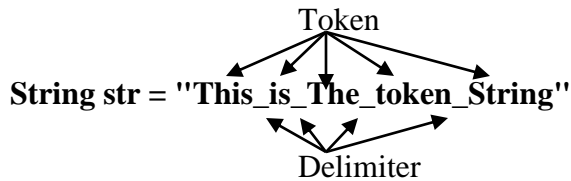
- **public void waitFor( ) throws InterruptedException**

### java.util.StringTokenizer class

This class is to prepare the tokens from any specified string.

- **public StringTokenizer(String str, String delimiter)**

Ex:



```
StringTokenizer stz = new StringTokenizer(str, "_");
```

- **public StringTokenizer(String str)**

In this constructor for the delimiter the space will be considered.

- **public StringTokenizer(String str, String delimiter, boolean b)**

If the Boolean argument is true then the delimiter will also be considered as the token.

### Method

- **public boolean hasMoreTokens( )**

If this method returns true if there is next token available.

- **public String nextToken( )**

This method returns the next token.

### //StringTokenizerTest.java

```
import java.util.*;
class StringTokenizerTest
{
 public static void main(String s[])
 {
 StringTokenizer stz = new StringTokenizer("This_is_the _token_String", "_",
true);
 while(stz.hasMoreTokens())
 {
 System.out.println(stz.nextToken());
 }
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\StringTokenizer>javac StringTokenizerTest.java
E:\JAVA\Programs\Core Java Program\Input-Output\StringTokenizer>java StringTokenizerTest
This
is
the
token
String
```

```
import java.util.*;
class StringTokenizerTest2
{
 public static void main(String s[])
 {
 StringTokenizer stz = new StringTokenizer("This_is_the _token_String", "_",
false);
 while(stz.hasMoreTokens())
 {
 System.out.println(stz.nextToken());
 }
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\StringTokenizer>javac StringTokenizerTest2.java
E:\JAVA\Programs\Core Java Program\Input-Output\StringTokenizer>java StringTokenizerTest2
This
is
the
token
String
```

## **Java.util.Scanner class**

This class is use to provide the data in the program from any source.  
This class always uses the space as delimiter.

**Ex:**

**File, device or String object**

### **Constructor**

- **public Scanner(String str)**
- **public Scanner(File f)**
- **public Scanner(InputStream isStream)**

### **Methods**

This class provides the method to read the different types of primitive data.

- **public String next( )**

It will return the String type next token.

- **public nextInt( )**

It will return the Int type nextToken.

If nextToken is not the int tupe then this method generates inputMisMatchExcaption.

- **public boolean hasNextInt( )**

It will check whether the nextToken is int type or not and returns true or false.

- **public float nextFloat( )**

It will check whether the nextToken is float or not and return true or false.

- **next\_\_\_\_\_ ( ) and hasNext\_\_\_\_\_ ( )**

These are available for all the primitive type.

- **public String nextLine( )**

This method will read the complete line till.

#### //ScannerTest.java

```
import java.util.*;
class ScannerTest
{
 public static void main(String s[])throws Exception
 {
 Scanner sc = new Scanner("ABC 101 85 A");
 while(sc.nextLine())
 {
 System.out.println(sc.next());
 }
 }
}
```

#### // ScannerTest1.java

```
import java.util.*;
class ScannerTest1
{
 public static void main(String s[])
 {
 Scanner sc = new Scanner("ABC 101 85 A");
 while(sc.hasNext())
 {
 System.out.println(sc.next());
 }
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>javac ScannerTest1.java
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>java ScannerTest1
ABC
101
85
A
```

#### // ScannerTest2.java

```
import java.io.*;
import java.util.*;
class ScannerTest2
{
 public static void main(String s[])throws FileNotFoundException
 {
 Scanner sc = new Scanner(new File("abc.txt"));
 System.out.print(" "+sc.next());
 System.out.print(" "+sc.nextInt());
 System.out.print(" "+sc.nextFloat());
 System.out.print(" "+sc.next());
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>javac ScannerTest2.java
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>java ScannerTest2
abc 101 85.5 A
```

#### // ScannerTest4.java

```
import java.io.*;
import java.util.*;
class ScannerTest4
{
 public static void main(String s[])
 {
 Scanner sc = new Scanner(System.in);
 System.out.println("Enter name, rollno, marks, and grade");
 System.out.println(sc.next());
 System.out.println(sc.nextInt());
 System.out.println(sc.nextFloat());
 System.out.println(sc.next());
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>javac ScannerTest4.java
E:\JAVA\Programs\Core Java Program\Input-Output\Scanner>java ScannerTest4
Enter name, rollno, marks, and and grade
abcc
abcc
101
101
85
85.0
A
A
```

# **NETWORKING**

Networking means the process by which the two computers can be connected to each other and communicate with each other.

Networking always is performed at the two levels.

1. Hardwar
2. Software

To make the software that can communicate with each other across the system though the network we have to first of all consider three things.

1. IP address
2. Port Number
3. Protocol

## **1. IP address**

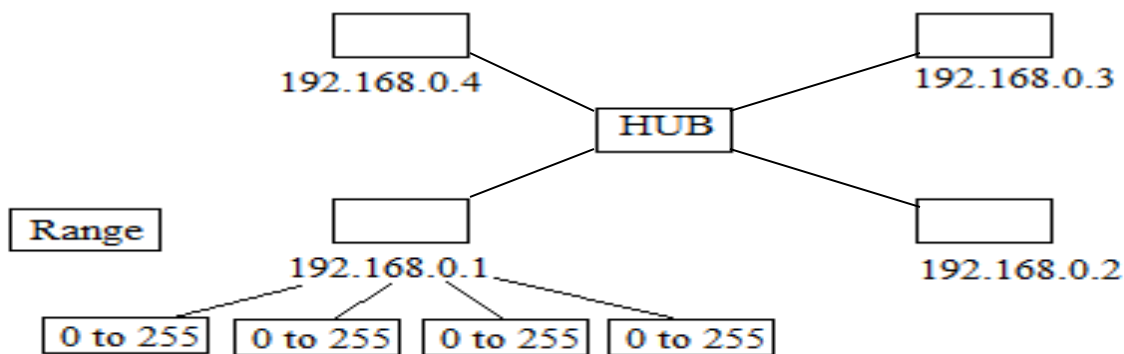
IP address is the unique identification number of a host (Computer) want to the network.

## **2. Port Number**

Port number is the application number running within the operating system an any machine.

## **3. Protocol**

Protocol is the set of rules that governs the communication.



One computer is connecting only one computer at a time.

Each and every application running in as has unique port number.

Total port no – 0 to 255

**At the network layer there are the two possible possible protocols that can be use.**

- 1. TCPIP Protocol**
- 2. UDP Protocol**

TCPIP is the connection oriented protocol and UDP is the connection less protocol.

Class in the java.net package

## Java.net

- **InitAddress**
- **Socket**
- **ServerSocket**
- **DatagramSocket**
- **URI**
- **URLConnection**

### Init Address

An instance of the InitAddress class represents to the IP Address and name of the host.

### This class provides the following methods

#### Static method

- **public static InitAddress getLocalhost( )**

This method will return the InitAddress object that contains the IPAddress and name of local host.

- **public static InitAddress getByName(String name)**

This method returns the InitAddress object that contains the IPAddress and name of specified host.

- **public static InitAddress getAllByName(String name)**

This method returns the array of object of InitAddress that contains the IPAddress and name of the entire host that has the specified name.

#### Non-Static Method

- **public String getHostName( )**
- **public String getHostAddress( )**

#### //InitTest.java

```
import java.net.*;
```

```
class InitTest
```

```
{
```

```
 public static void main(String s[])throws UnknownHostException
```

```
 {
```

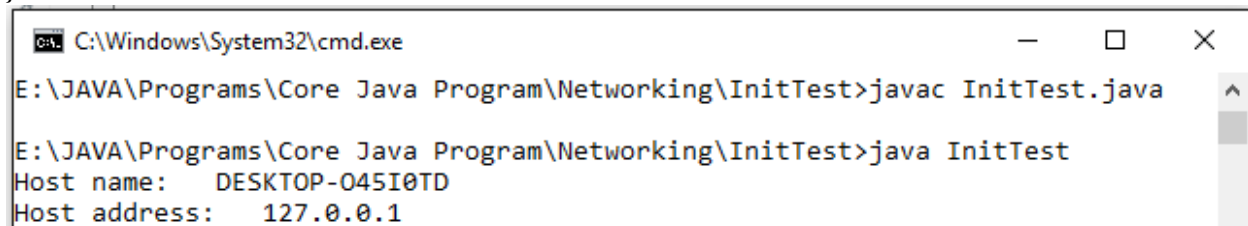
```
 InetAddress inet=InetAddress.getByName("DESKTOP-O45I0TD");
```

```
 System.out.println("Host name: "+inet.getHostName());
```

```
 System.out.println("Host address: "+inet.getHostAddress());
```

```
 }
```

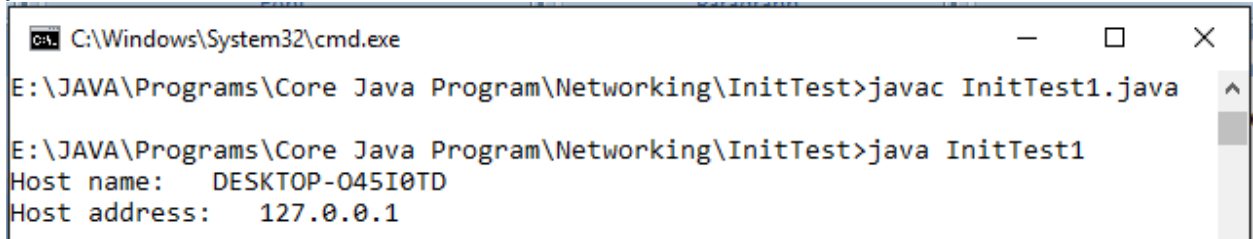
```
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Networking\InitTest>javac InitTest.java
E:\JAVA\Programs\Core Java Program\Networking\InitTest>java InitTest
Host name: DESKTOP-O45I0TD
Host address: 127.0.0.1
```

### //InitTest1.java

```
import java.net.*;
class InitTest1
{
 public static void main(String s[])throws UnknownHostException
 {
 InetAddress inet=InetAddress.getLocalHost();
 System.out.println("Host name: "+inet.getHostName());
 System.out.println("Host address: "+inet.getHostAddress());
 }
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The command prompt is open at the directory "E:\JAVA\Programs\Core Java Program\Networking\InitTest". The user has entered the command "javac InitTest1.java" and the output is "Host name: DESKTOP-045I0TD" and "Host address: 127.0.0.1". The user has then entered the command "java InitTest1" and the output is "Host name: DESKTOP-045I0TD" and "Host address: 127.0.0.1".

### Java.net.Socket class

An instance of the class represents to the TCP/IP socket.

A socket is the combination of IP Address, port number and protocol.

#### Constructor

- **public Socket(String ipAddress, int portNumber)**

Whenever the socket object is created immediately a connection request is sent to the application running on the specified host (IPAddress).

### Java.net.ServerSocket class

An instance of this class represents to the TCP/IP server.

#### Constructor

- **public ServerSocket(int portNo)**

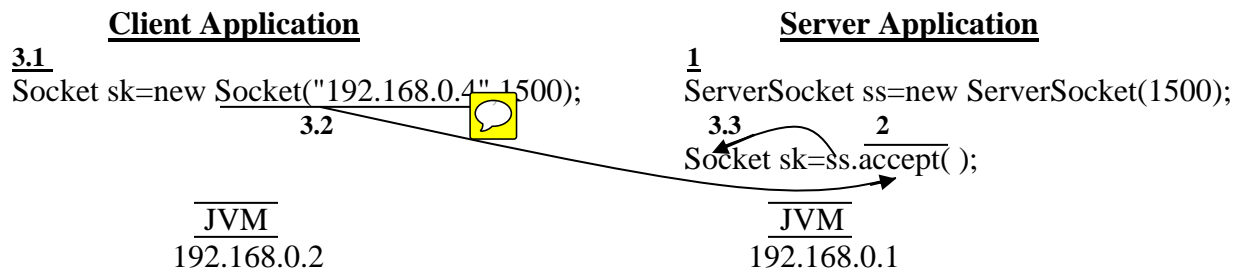
This constructor will create the ServerSocket object on to the server application.

#### Method of ServerSocket class

- **public Socket accept()**

This method is used to accept the connection request sent by the client to the ServerSocket.

Calling of this method will be terminated when the socket request is accepted.



1. ServerSocket object is created.
2. accept( ) method invoked and wait for the connection request from the client Socket.
- 3.1 In the client application Socket objet is created.
- 3.2 Connection request submitted to the sever and accepted by the accept() method.
- 3.3 Proxy Socket object created and return by the accept() method.

#### // ClientTest.java

```
import java.net.*;
class ClientTest
{
 public static void main(String s[])
 {
 try
 {
 System.out.println("Sending the connection request.....");
 Socket sk=new Socket("localhost",1500);
 System.out.println("Connection established");
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Networking\Socket>javac ClientTest.java
E:\JAVA\Programs\Core Java Program\Networking\Socket>java ClientTest
Sending the connection request.....
Connection established
```

#### // SeverTest.java

```
import java.net.*;
class ServerTest
{
 public static void main(String s[])
 {
 try
 {
 System.out.println("Server started");
 }
 }
}
```

```

 ServerSocket ss=new ServerSocket(1500);
 while(true)
 {
 System.out.println("waiting for the client request.... ");
 Socket sk=ss.accept();
 System.out.println("Client connected");
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}

```

```

C:\Windows\System32\cmd.exe - java ServerTest
E:\JAVA\Programs\Core Java Program\Networking\Socket>javac ServerTest.java
E:\JAVA\Programs\Core Java Program\Networking\Socket>java ServerTest
Server started
waiting for the client request....
Client connected
waiting for the client request....

```

## Method of the Server class

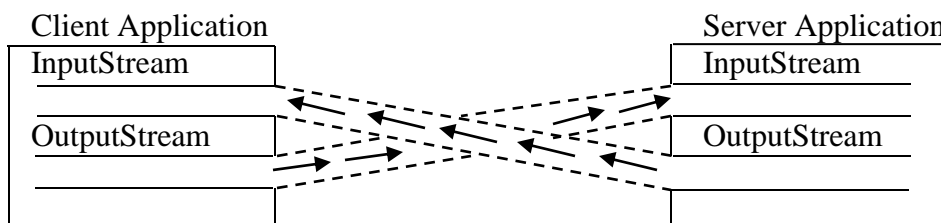
### - **public InputStream getInputStream( )**

This method will return the InputStream object that is used to receive the data from the network.

### - **Public OutputStream getOutputStream( )**

This method will return the outputStream object that is used to send the data from to the network.

As the Socket object is received on to the server then the InputStream and OutputStream gets twisted that means InputStream on to the client becomes the OuputStream on to the server and OutputStream onto the client becomes the InputStream onto the server.



### // **NetClient.java**

```

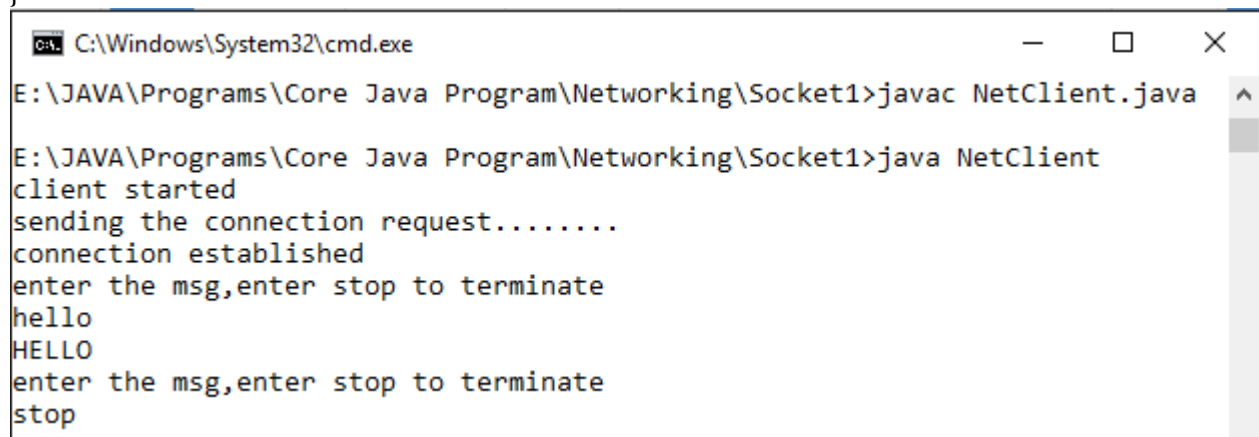
import java.net.*;
import java.io.*;
class NetClient
{
 public static void main(String s[])

```

```

 {
 try
 {
 System.out.println("client started");
 System.out.println("sending the connection request.....");
 Socket sk=new Socket("localhost",1500);
 System.out.println("connection established");
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
 DataInputStream din=new DataInputStream(sk.getInputStream());
 DataOutputStream dout=new DataOutputStream(sk.getOutputStream());
 while(true)
 {
 System.out.println("enter the msg,enter stop to terminate");
 String str=br.readLine();
 dout.writeUTF(str);
 dout.flush();
 if(str.equals("stop"))
 break;
 str=din.readUTF();
 System.out.println(str);
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user is in the directory "E:\JAVA\Programs\Core Java Program\Networking\Socket1". They have compiled the "NetClient.java" file using "javac NetClient.java". Then, they ran "java NetClient", which produced the following output:

```

client started
sending the connection request.....
connection established
enter the msg,enter stop to terminate
hello
HELLO
enter the msg,enter stop to terminate
stop

```

```

// NetServer.java
import java.net.*;
import java.io.*;
class NetServer
{
 public static void main(String s[])

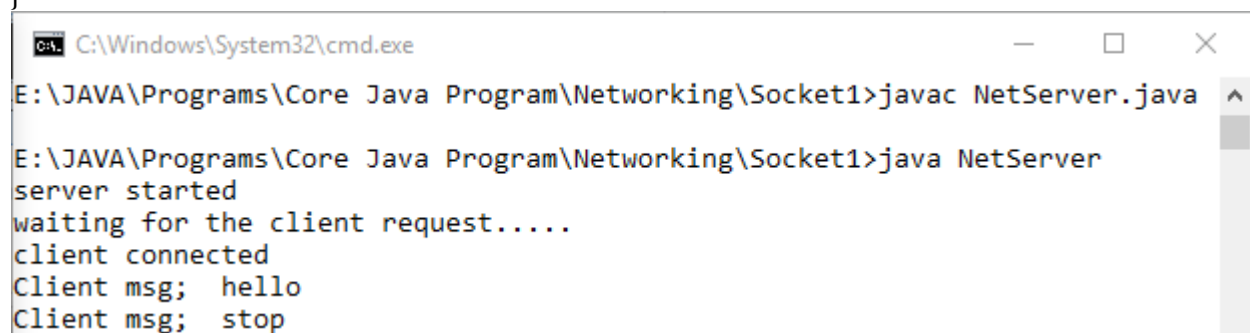
```



```

{
 try
 {
 System.out.println("server started");
 ServerSocket ss=new ServerSocket(1500);
 System.out.println("waiting for the client request.....");
 Socket sk=ss.accept();
 System.out.println("client connected");
 DataInputStream din=new DataInputStream(sk.getInputStream());
 DataOutputStream dout=new DataOutputStream(sk.getOutputStream());
 while(true)
 {
 String msg=din.readUTF();
 System.out.println("Client msg; "+msg);
 if(msg.equals("stop"))
 break;
 msg=msg.toUpperCase();
 dout.writeUTF(msg);
 dout.flush();
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}

```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The user has navigated to the directory "E:\JAVA\Programs\Core Java Program\Networking\Socket1". The commands executed are:

```

javac NetServer.java
java NetServer

```

The output of the program is displayed as follows:

```

server started
waiting for the client request.....
client connected
Client msg; hello
Client msg; stop

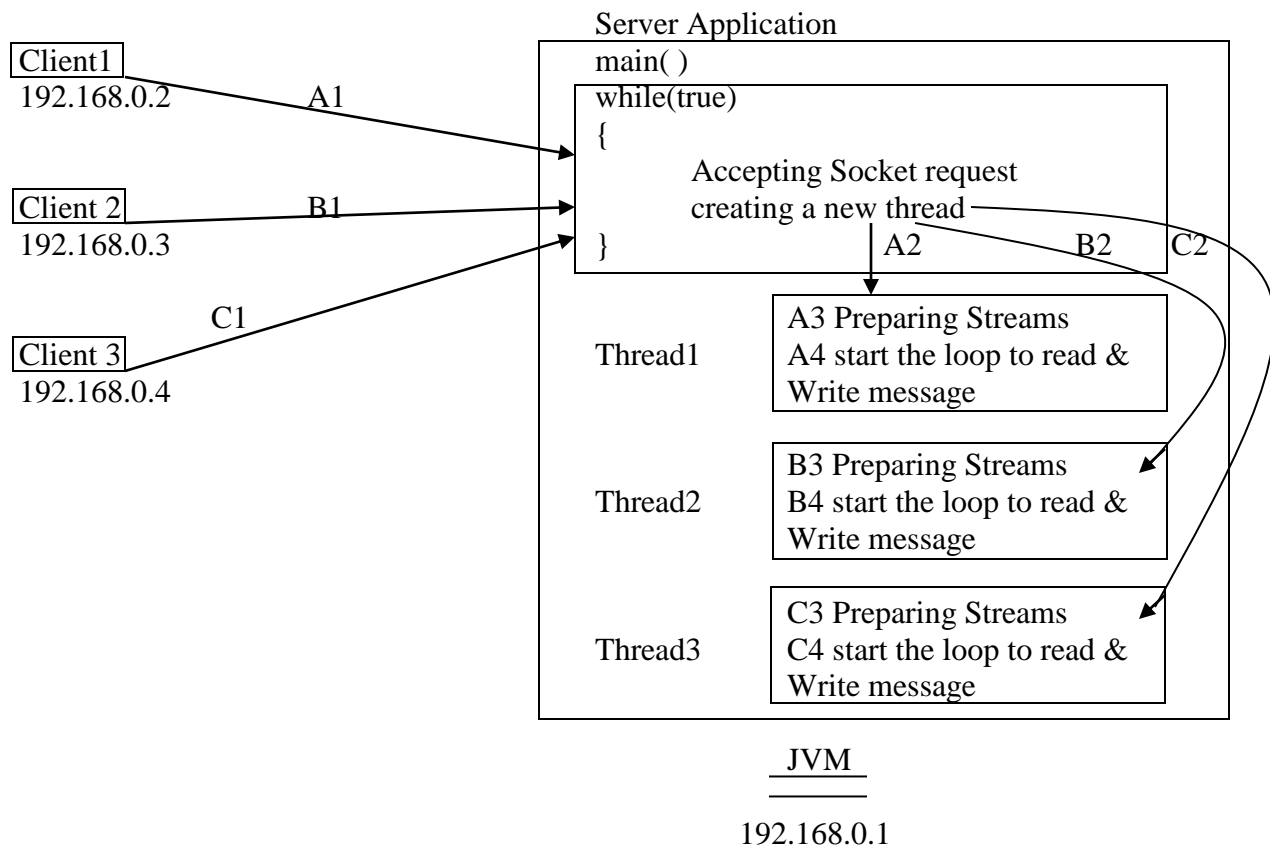
```

### **Limitation of this program**

In the previous program NetClient.java / NetServer.java there is the limitation in the server program is that sever can connect and communicate only with the single client.

We have to create the multithreaded server to connect and communicate with multiple clients.

As any socket request is accepted by the server immediately a new thread is created and that thread is responsible to prepare the InputStream and OutputStream of the client and to receive and send the message over these stream.



A1. Socket object created on the client.

A2. New thread started and the reference of socket object provided to that thread.

A3. Preparing Streams

A4. Start the loop to read and write message.

## **Broadcast Server**

Broadcast server is the extended from of the multithreaded server.

On the server side for the each client a separate thread is created that will receive the message from the associated client and will broadcast to all other client.

On to the server the connected socket object will be stored in the collection.

The client also will be the multithreaded that means on the client side the separate thread will be created for receiving the message send by the broadcast server.

### **// BroadcastServer.java**

```
import java.util.*;
import java.net.*;
import java.io.*;
class BroadcastServer
{
 public static void main(String s[])
 {
 try
 {
```

```

 System.out.println("server started");
 ServerSocket ss=new ServerSocket(1500);
 ArrayList sockets=new ArrayList();
 // collection object to store the Socket objects
 while(true)
 {
 Socket sk=ss.accept();
 System.out.println("client connected");
 sockets.add(sk);
 // adding the socket into collection object
 new ClientThread(sk,sockets);
 // new client thread started
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

class ClientThread extends Thread
{
 ArrayList al;
 Socket sk;
 ClientThread(Socket sk , ArrayList al)
 {
 this.sk=sk;
 this.al=al;
 start();
 }
 public void run()
 {
 try
 {
 DataInputStream din=new DataInputStream(sk.getInputStream());
 while(true)
 {
 String msg=din.readUTF();
 String str[]=msg.split("#");
 // System.out.println(str[0]);
 if(str[0].equals("login"))
 {
 msg=str[1]+" has loggedin";
 System.out.println(str[1]+" connected");
 }
 else if(str[0].equals("logout"))
 {
 msg=str[1]+" has loggedout";

```

```

 }
 else
 {
 msg = str[1];
 }
 broadcast(msg);

 if(str[0].equals("logout"))
 {
 al.remove(sk);
 // to remove the current socket from collection
 break;
 }
 } // while close
} // try close
catch(Exception e)
{
 e.printStackTrace();
}
} // run method close
void broadcast(String msg)
{
 try
 {
 for(Object ob:al)
 {
 Socket sk=(Socket)ob;
 DataOutputStream dout=new
DataOutputStream(sk.getOutputStream());
 dout.writeUTF(msg);
 dout.flush();
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```

#### **// ChatClient.java**

```

import java.net.*;
import java.io.*;
class ChatClient
{
 public static void main(String s[])
 {
 try
 {
 System.out.println("client started");

```

```

 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
 System.out.println("enter the user name");
 String name=br.readLine();
 Socket sk=new Socket("127.0.0.1",1500);
 System.out.println("connection established");
 new ReaderThread(sk);
 DataOutputStream dout=new DataOutputStream(sk.getOutputStream());
 dout.writeUTF("login#"+name);
 dout.flush();
 System.out.println("START TO ENTER THE MESSAGES, ENTER
'STOP' TO TERMINATE");
 while(true)
 {
 String str = br.readLine();
 if(str.equalsIgnoreCase("stop"))
 {
 dout.writeUTF("logout#"+name);

 dout.flush();
 break;
 }
 else
 {
 dout.writeUTF("message#"+name+": "+str);
 dout.flush();
 }
 } // while close
 } // try close
 catch(Exception e)
 {
 e.printStackTrace();
 }
}

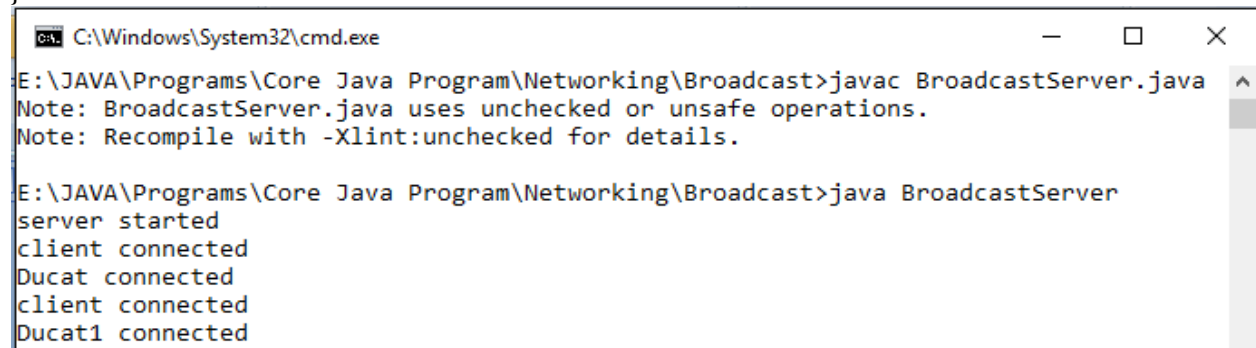
class ReaderThread extends Thread
{
 Socket sk;
 ReaderThread(Socket sk)
 {
 this.sk=sk;
 setDaemon(true);
 start();
 }
 public void run()
 {
 try
 {

```

```

 DataInputStream din = new DataInputStream(sk.getInputStream());
 while(true)
 {
 String msg=din.readUTF();
 System.out.println(msg);
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```



A screenshot of a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The window shows the following commands and output:

```

E:\JAVA\Programs\Core Java Program\Networking\Broadcast>javac BroadcastServer.java
Note: BroadcastServer.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\JAVA\Programs\Core Java Program\Networking\Broadcast>java BroadcastServer
server started
client connected
Ducat connected
client connected
Ducat1 connected

```



A screenshot of a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The window shows the following commands and output:

```

E:\JAVA\Programs\Core Java Program\Networking\Broadcast>javac ChatClient.java

E:\JAVA\Programs\Core Java Program\Networking\Broadcast>java ChatClient
client started
enter the user name
Ducat
connection established
START TO ENTER THE MESSAGES, ENTER 'STOP' TO TERMINATE
Ducat has loggedin
Ducat1 has loggedin
Hi
Ducat: Hi
Ducat1: Hello
Ducat1 has loggedout
STOP
Ducat has loggedout
E:\JAVA\Programs\Core Java Program\Networking\Broadcast>

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Networking\Broadcast>java ChatClient
client started
enter the user name
Ducat1
connection established
START TO ENTER THE MESSAGES, ENTER 'STOP' TO TERMINATE
Ducat1 has loggedin
Ducat: Hi
Hello
Ducat1: Hello
STOP
Ducat1 has loggedout
```

## **DatagramSocket and DatagramPacket class**

These classes are use to for the UDP based communication.

An instance of DatasgramSocket is used to send to receive the DatagramPacket.

### **Constructor**

- **public DatagramSocket(int portNo)**

This constructor will create the datagramSocket object and register it will the argument port number.

### **Methods**

- **public void send(DetagramPacket pocket)**
- **public void received(DatagramPacket pocket)**

These two methods are use to send or receive the DatagramPacket.

## **DatagramaPocket**

DatagramPocket instance represent to the UDP pocket.

A DatagramPacket instance contains a data in the form of byte array and can also contains the IP Address and port number of the server.

### **Constructor**

- **public Datagrampocket(byte []b, int size, InitAddress address, int portNo)**
- **public DatagramPocket(byte []b, int size)**

1<sup>st</sup> constructor will be used at the sender side.

2<sup>nd</sup> constructor will be used at the receiver side.

### **//UDPSender.java**

```
import java.net.*;
import java.io.*;
class UDPSender
{
 public static void main(String s[])
 {
 try
 {
```

```

 System.out.println("Sender Started");
 DatagramSocket sender = new DatagramSocket(3000);
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 while(true)
 {
 System.out.println("Enter the message,,ENter space to terminate");
 String str = br.readLine();
 if(str.equals(" "))
 break;
 byte b[] = str.getBytes();
 DatagramPacket packet = new DatagramPacket(b,
b.length,InitAddress.getLocalHost(),4000);
 sender.send(packet);
 System.out.println("MESSAGE SENT SUCCESSFULLY");
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```

#### **//UDPReceiver.java**

```

import java.net.*;
class UDPReceiver
{
 public static void main(String s[])
 {
 try
 {
 System.out.println("REceiver Started");
 DatagramSocket receiver = new DatagramSocket(4000);
 while(true)
 {
 byte b[] = new byte[100];
 DatagramPacket packet = new DatagramPacket(b, b.length);
 receiver.receive(packet);
 String str = new String(b);
 System.out.println("Message read"+str);
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```



```
}
}
```

### **URL and URLConnection class**

An instance of URL represents valid URL on to the internet that means an instance of URL class contains the valid URL of any web page on the internet.

- **public URL(String url)**  
This constructor will create the URL instance.

### **Method**

- **public URLConnection openConnection( )**

### **URLConnection**

An instance of URLConnection represent to the connection between the java application and the web server.

URLConnection instance provides the readymade InputStream and OutputStream through which content of web page can be loaded from the server or any content can be uploaded on to the server.

#### **//URLTest.java**

```
import java.io.*;
import java.net.*;
class URLTest
{
 public static void main(String []args)
 {
 try
 {
 System.out.println("Sending the request for web page");
 URL url=new URL("https://www.facebook.com");
 URLConnection con=url.openConnection();
 System.out.println("web page loaded");
 InputStream in=con.getInputStream();
 BufferedReader br=new BufferedReader(new InputStreamReader(in));
 File f=new File("abc.html");
 FileOutputStream fout=new FileOutputStream(f);
 String str;
 while((str=br.readLine())!=null)
 {
 System.out.println(str);
 byte b[]=str.getBytes();
 fout.write(b);
 fout.flush();
 }
 Runtime r=Runtime.getRuntime();
 String path=f.getAbsolutePath();
```

```

 System.out.println(path);
 Process p=r.exec("C:\\Program Files
(x86)\\Google\\Chrome\\Application\\chrome.exe"+path);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```

## **REFLECTION**

Reflection is the way analyzes the class at the runtime such as to identity name of the variables, name of the methods, list of the constructors and name of the parent class and list of the implemented.

There are two types of class loading in java

1. Implicit class loading
2. Explicit class loading

### **Implicit class loading**

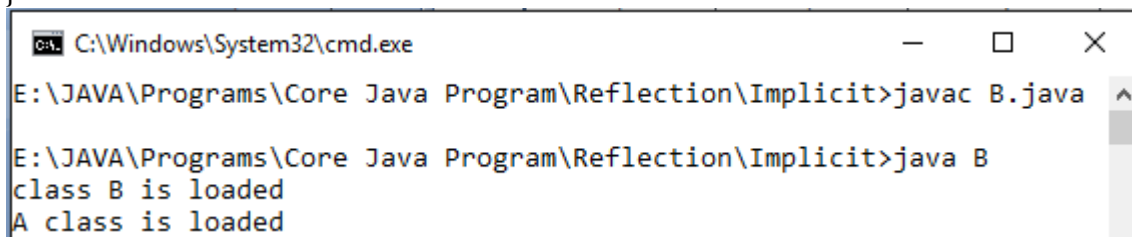
Any class can be loaded implicitly when their first object is created or first time any static member is invoked.

That means there is no need to write any each statement to load the class.

### **Example of Implicit class loading**

**//B.java**

```
class A
{
 static
 {
 System.out.println("A class is loaded");
 }
}
class B
{
 static
 {
 System.out.println("class B is loaded");
 }
 public static void main(String s [])
 {
 A ref = new A();
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Implicit>javac B.java
E:\JAVA\Programs\Core Java Program\Reflection\Implicit>java B
class B is loaded
A class is loaded
```

### **Explicit class loading**

In order to load any class explicitly the separate statement we have to written.

Explicit class loading will be used when the class name is found at the runtime.

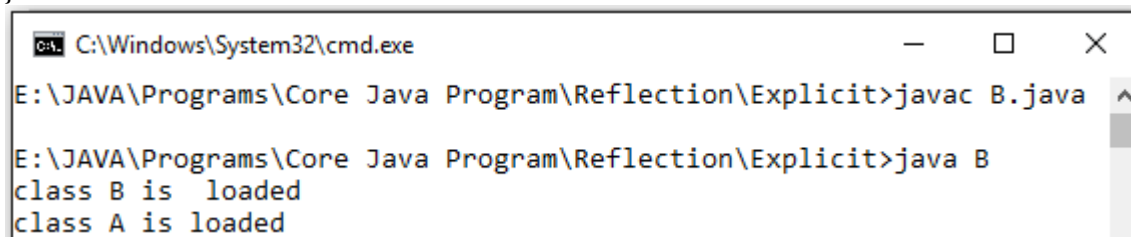
## **Java.lang.Class class**

Class class provides the static method to load any class explicitly.

### **forName()**

**forName()** method used to load any class explicitly.

```
class A
{
 static
 {
 System.out.println("class A is loaded");
 }
}
class B
{
 static
 {
 System.out.println("class B is loaded");
 }
 public static void main(String s[])
 {
 try
 {
 Class.forName("A"); //class name is the form of String
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Explicit>javac B.java
E:\JAVA\Programs\Core Java Program\Reflection\Explicit>java B
class B is loaded
class A is loaded
```

### **Advantages of Implicit class**

Advantages of implicit class loading no additional statement will be written for the implicit class loading.

### **Disadvantages of Implicit class**

If class name is not known at development time then cannot be loaded implicitly.

### Advantages of Explicit class

Advantages of explicit class if the class name is formed at the runtime in that class can also be loaded.

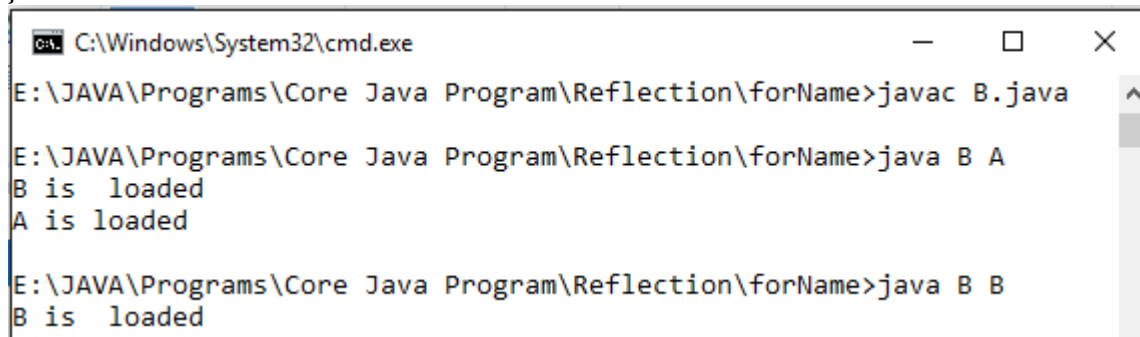
### Disadvantages of Implicit class

Explicit statements have to be written for the class loading.

[**Note:** - In the argument of `forName()` method the class name should be specified with the package name.]

[**Note:** - In the argument of `forName()` method we can also pass the String variable in which runtime value can be accepted.]

```
class A
{
 static
 {
 System.out.println("A is loaded");
 }
}
class B
{
 static
 {
 System.out.println("B is loaded");
 }
 public static void main(String s[])
 {
 try
 {
 Class.forName(s[0]);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\forName>javac B.java
E:\JAVA\Programs\Core Java Program\Reflection\forName>java B A
B is loaded
A is loaded
E:\JAVA\Programs\Core Java Program\Reflection\forName>java B B
B is loaded
```

[**Note:** - If the class file is not found then implicit loading.]

### Implicit Loading

NoClassDefFoundError

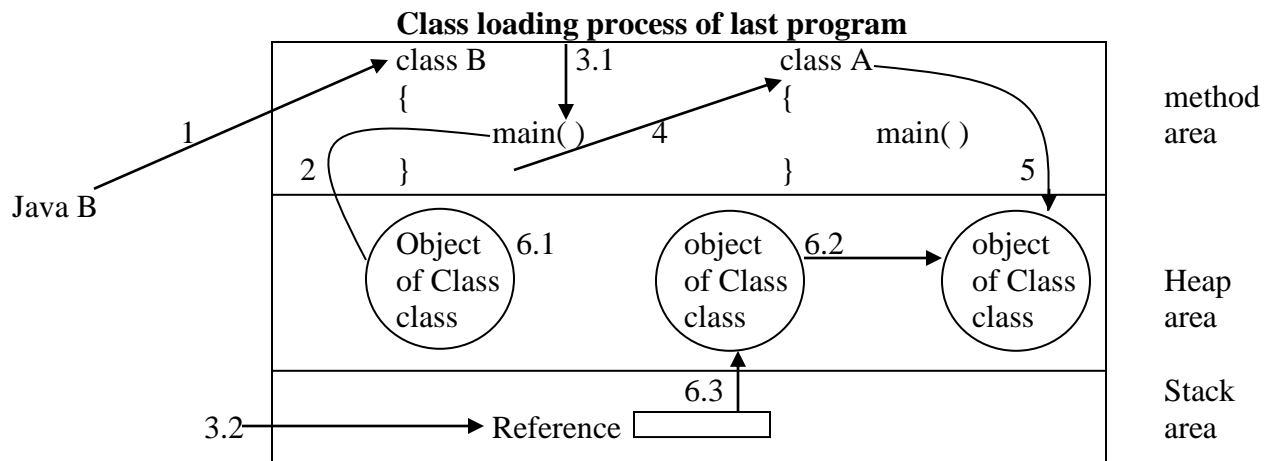
### Explicit Loading

ClassNotFoundException

### Reflected object of the loaded class

Always each and every class is reflected by the separate object of **java.lang.Class** class as any class gets loaded immediately the separate object of Class class created for that loaded class.

If at any particular time there are five classes already loaded that means the first objects of the class will be created one for the each loaded class separately?



1. Class B loaded.
2. Object of class (reflected object) created for the B class.
- 3.1 main() method invoked by the JRE.
- 3.2 form of the main() method is created.
4. A class loaded because the first object of A class is about to be created.
5. Object of Class class is created for the A class.
- 6.1 Object of A class created.
- 6.2 Reference id of Class class object stored into the A class object.
- 6.3 Reference of A class object stored into the main() method.

[**Note:** - The object of any class contains the Reference id of the Class class object of that class.]

**Question:** - How we can get the Reference id of Class class object.

**Answer:** - In case of implicit loading and explicit loading there are the different ways to get Reference id of Class class object.

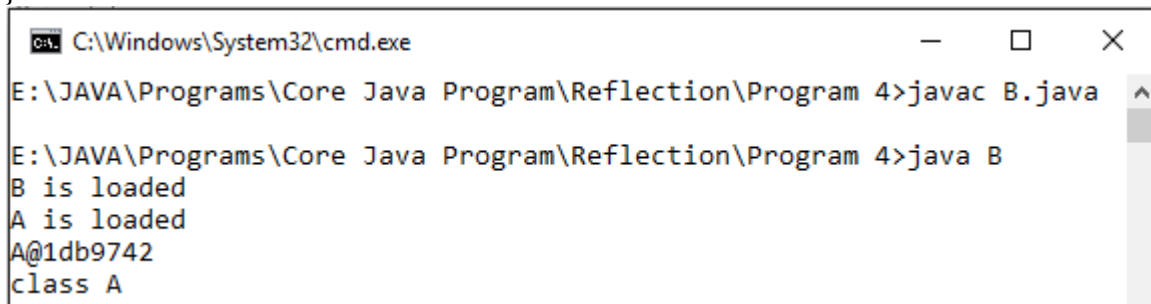
### **In case of Implicit loading**

The getClass() method Java.lang.Object class is used to get Reference id of the Class class object.

#### **Signature of getClass() method**

- **public Class getClass()**

```
class A
{
 static
 {
 System.out.println("A is loaded");
 }
}
class B
{
 static
 {
 System.out.println("B is loaded");
 }
 public static void main(String s[])
 {
 A ref = new A();
 Class c = ref.getClass();
 System.out.println(ref);
 System.out.println(c);
 }
}
```



```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Program 4>javac B.java
E:\JAVA\Programs\Core Java Program\Reflection\Program 4>java B
B is loaded
A is loaded
A@1db9742
class A
```

### **In case of Explicit class loading**

The forName() method return the reference id Class class object after loading the class.

#### **Signature of the forName() method**

- **public static Class forName(String className) throws ClassNotFoundException**

```
class A
{
 static
 {
```

```

 System.out.println("A is loaded");
 }
}
class B
{
 static
 {
 System.out.println("B is loaded");
 }
 public static void main(String s[])
 {
 try
 {
 Class c = Class.forName("A");
 //class will be loaded and reference id of class class object obtained.
 System.out.println(c);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Explicit1>javac B.java
E:\JAVA\Programs\Core Java Program\Reflection\Explicit1>java B
B is loaded
A is loaded
class A

```

### **Method of Class class**

- **public String getName( )**

This method returns the name of the class to which Class class object belongs.

**Ex:**

```

Student st = new Student();
Employee emp = new Employee();

class c1 = st.getClass();
//Reference id of object of java.lang.Class returned for the Student class.
Class c2 = emp.getClass();
//Reference id of object of java.lang.Class returned for the Employee class.

System.out.println(c1.getName()); //Student
System.out.println(c1.getName()); //Employee

```



**[Note: -** In any method we can get the name of the class when objects id is passed in the methods argument.]

```
class Animal
{
}
class Dog extends Animal
{
}
class Cat extends Animal
{
}
class Test
{
 static void forest(Animal a)
 {
 Class c = a.getClass();
 String name = c.getName();
 if(name.equals("Dog"))
 {
 Dog d = (Dog)a;
 }
 if(name.equals("Cat"))
 {
 Cat ca = (Cat)a;
 }
 }
 public static void main(String s[])
 {
 Test.forest(new Cat());
 Test.forest(new Dog());
 }
}
```

### **Statement through the method calling**

```
class Test1
{
 static void forest(Animal a)
 {
 if(a.getClass().getName().equals("Dog"))
 //to know the class name whose objects is contain by the reference variable
 {
 Dog d = (Dog)a;
 }
 if(a.getClass().getName().equals("Cat"))
 {
 Cat ca = (Cat)a;
 }
 }
}
```

```

 }
 }
 public static void main(String s[])
 {
 Test.forest(new Cat());
 Test.forest(new Dog());
 }
}

```

**[Note: -** In case of explicit class loading how we can create the object whose name founds at the runtime.]

As any class gets loaded through the `forName( )` method there class object reference id immediately return.

### **newInstance( )**

`newInstance( )` method of the `java.lang.Class` class is use to create the object of loaded class.

### **Signature**

**- public Object newInstance( )**

This method always uses the default constructor to create the object.

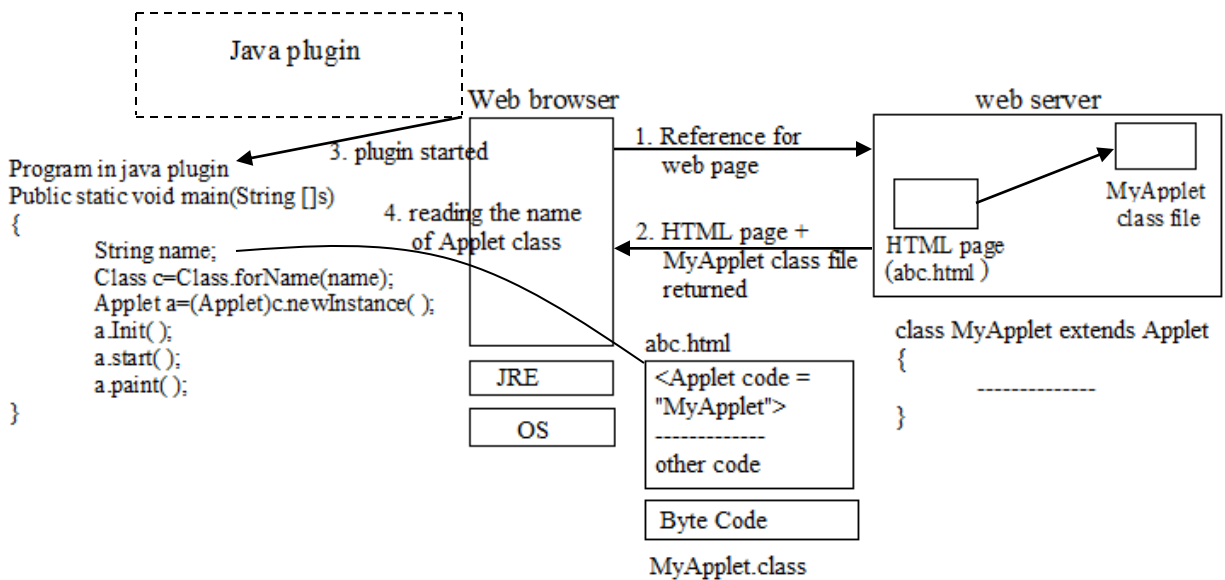
### **Ex:**

```
Class c = Class.forName("A");
```

```

class Test2
{
 public static void main(String s[])throws Exception
 {
 Class c=Class.forName(s[0]);
 Object ob=c.newInstance();
 }
}

```



[**Note:** - The object of class always the metadata about the class such as **name of the parent class**, **list of the methods**, **list of the fields**, **list of the constructor**, **list of the name of the implement interface**.]

Each and every component there is represented by the object known as the reflected object.

### Java.lang.reflect package

**Java.lang.reflect** package provides the class known as the reflected class whose instances represents to the member of class name of the reflected classes.

- **Field class**
- **Constructor class**
- **Method class**

### Method of java.lang.Class class

- **public Field[ ] getFields( )**

This method returns the array of the object of Field class each object is representing to a specific field.

This method always returns the object for public field including the inheritance fields.

- **public Field[ ] getDeclaredFields( )**

This method returns the field object for all the fields excluding the inherited fields.

- **public Field[ ] getDeclaredFields(String name)**

This method returns the field object for the specified field.

- **public Method[ ] getDeclaredMethods( )**

This method returns the array of the method objects including the private methods but excluding the inheritance methods.

- **public Method[ ] getMethods( )**

This method returns the object for the public methods including the inherited methods.

- **public Class[ ] getSuperClass( )**

This method returns the Class class object of the super class.

- **public Class[ ] getInterfaces( )**

This method returns the Class class object for all the implemented interfaces.

- **public Constructor[ ] getConstructors( )**

This method returns the constructor objects for all public constructors.

- **public Constructor[ ] getDeclaredConstructors( )**

It will return the object for all constructors.

**//MyJavaP.java**

```
import java.lang.reflect.*;
```

```
class MyJavaP
```

```
{
```

```
 public static void main(String s[])throws ClassNotFoundException
```

```
 {
```

```
 Class c = Class.forName(s[0]);
```

```
 System.out.println(s[0]+" class loaded");
```

```
 Class parent = c.getSuperclass();
```

```
 System.out.println("Super Class: "+parent.getName());
```

```
 Class intr[] = c.getInterfaces();
```

```
 System.out.println("\n List Of Implemented Interface");
```

```
 for(Class in:intr)
```

```
 {
```

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

```
 }
```

```
 Method mth[] = c.getDeclaredMethods();
```

```
 System.out.println("\n list of the methods:");
```

```
 for(Method m:mth)
```

```
 {
```

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

```
 }
```

```
 Constructor ctr[] = c.getDeclaredConstructors();
```

```
 System.out.println("\n list of constructor");
```

```
 for(Constructor ct:ctr)
```

```
 {
```

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

```
 }
```

```
 Field fd[] = c.getDeclaredFields();
```

```
 for(Field f:fd)
```

```
 {
```

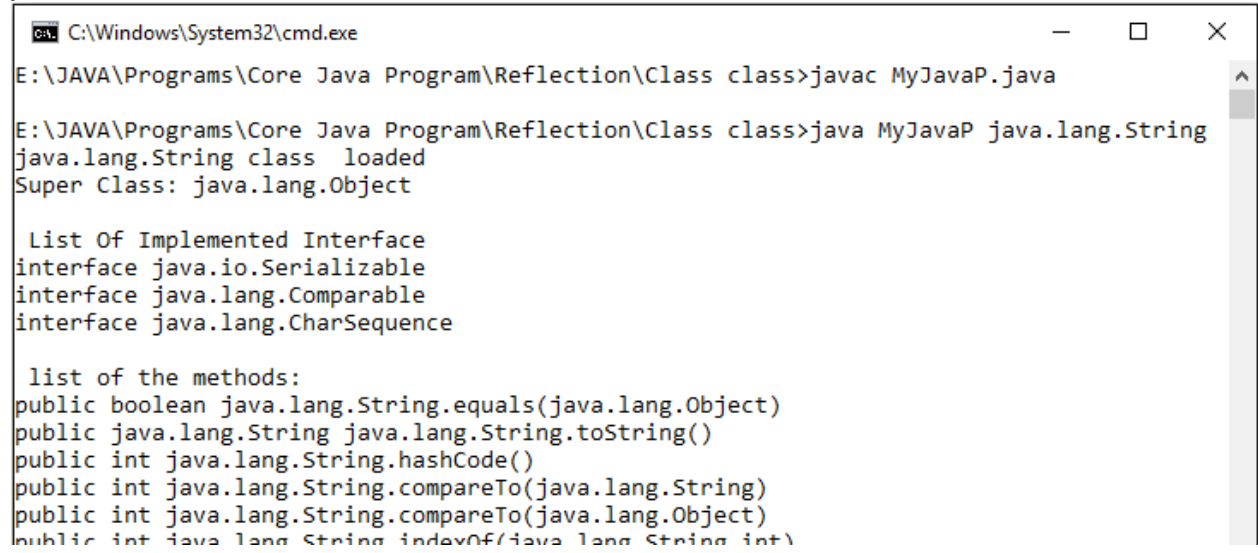
```
 System.out.println(f);
```

```
 }
```

```

 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Class class>javac MyJavaP.java

E:\JAVA\Programs\Core Java Program\Reflection\Class class>java MyJavaP java.lang.String
java.lang.String class loaded
Super Class: java.lang.Object

List Of Implemented Interface
interface java.io.Serializable
interface java.lang.Comparable
interface java.lang.CharSequence

list of the methods:
public boolean java.lang.String.equals(java.lang.Object)
public java.lang.String java.lang.String.toString()
public int java.lang.String.hashCode()
public int java.lang.String.compareTo(java.lang.String)
public int java.lang.String.compareTo(java.lang.Object)
public int java.lang.String.indexOf(java.lang.String int)

```

Through the reflection object we can also access the members of the class.

### //Test3.java

```

import java.lang.reflect.*;
class Test3
{
 int x;
 void show()
 {
 System.out.println("Value = "+x);
 }
 public static void main(String s [])throws Exception
 {
 Class c = Class.forName("Test3");
 System.out.println("Test class loaded");
 Test3 t = (Test3)c.newInstance();
 Field f = c.getDeclaredField("x");
 f.setInt(t, 100); //100 value to be stored in x
 //t is the reference id of object through which x variable will be accessed
 Method m = c.getDeclaredMethod("show");
 m.invoke(t);
 }
}

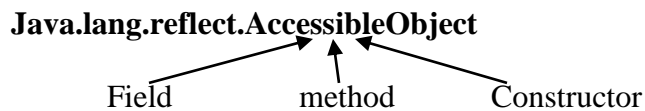
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\Class class>javac Test3.java
Note: Test3.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\JAVA\Programs\Core Java Program\Reflection\Class class>java Test3
Test class loaded
Value = 100
```

### **java.lang.reflect.AccessibleObject class**

This class is the parent class for all the reflected classes such as field, method and constructor.



AccessibleObject class provides the method named setAccessible( ) which is use to set the accessibility of the private members.

### **Signature**

- **public void setAccessible(boolean b)**

### **//AccesTest.java**

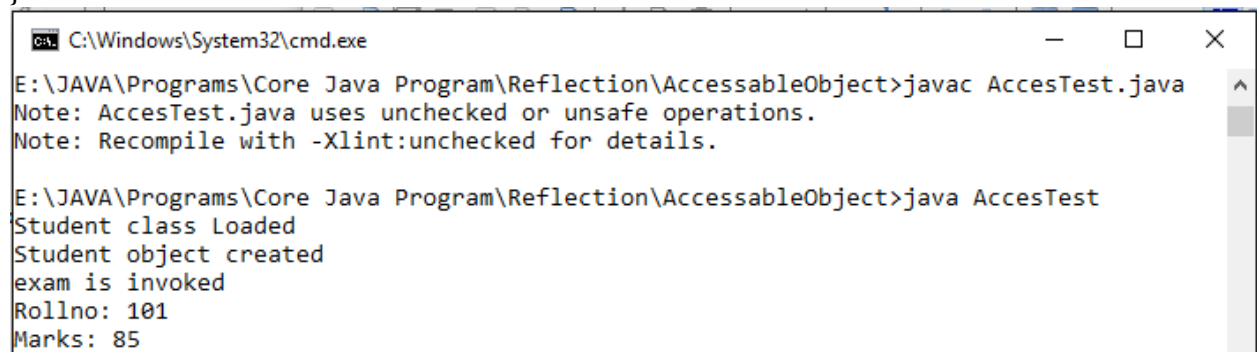
```
import java.lang.reflect.*;
class Student
{
 private int rollno,marks;
 String name;
 private Student(String name, int rollno)
 {
 this.name = name;
 this.rollno = rollno;
 System.out.println("Student object created");
 }
 private void exam(int marks)
 {
 this.marks = marks;
 System.out.println("exam is invoked");
 }
 private void show()
 {
 System.out.println("Rollno: "+rollno);
 System.out.println("Marks: "+marks);
 }
}
```

```

}
class AccesTest
{
 public static void main(String s[])
 {
 try
 {
 Class c = Class.forName("Student");
 System.out.println("Student class Loaded");
 Constructor ctr = c.getDeclaredConstructor(new Class[
]{{String.class, int.class}});
 ctr.setAccessible(true);
 Student st = (Student)ctr.newInstance(new Object[]{"ABC",101});
 Method mth1 = c.getDeclaredMethod("exam",new Class[
]{{int.class}});

 mth1.setAccessible(true);
 mth1.invoke(st, new Object[]{85});
 Method mth2 = c.getDeclaredMethod("show");
 mth2.setAccessible(true);
 mth2.invoke(st);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 }
}

```



```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Reflection\AccessableObject>javac AccesTest.java
Note: AccesTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\JAVA\Programs\Core Java Program\Reflection\AccessableObject>java AccesTest
Student class Loaded
Student object created
exam is invoked
Rollno: 101
Marks: 85

```

**[Note: - reflection can't use the real world programming it use the developer for make development tools programs.]**

# **MULTITHREADING**

Multithreading is the way to achieve the multitasking.

Multitasking is the way to load the multiple tasks in the memory at same time for the concurrent execution.

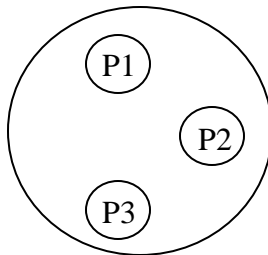
**There are two types of multithreading**

- 1. Multiprocessing**
- 2. Multithreading**

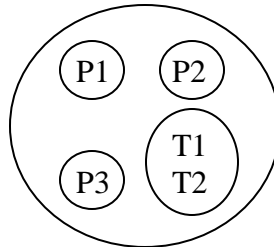
A process is the program in execution within the operating System.

A thread is the part of the process.

**Process based Multitasking**



**Thread based Multitasking**

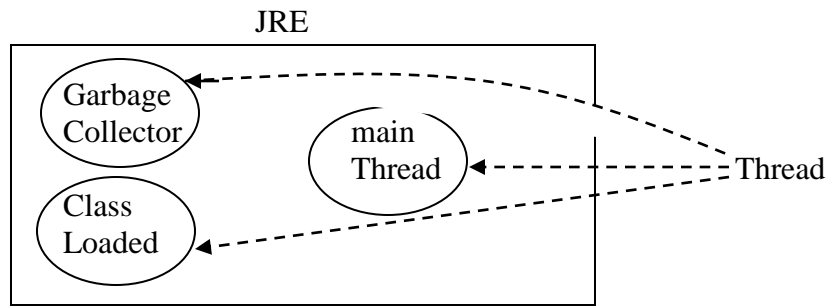


## **Difference between Multiprocessing and Multithreading**

| <b>Multiprocessing</b>                                                            | <b>Multithreading</b>                                         |
|-----------------------------------------------------------------------------------|---------------------------------------------------------------|
| <b>1</b> A process represent to the complete application within operating system. | <b>1</b> A thread is a part of the process.                   |
| <b>2</b> Creating of process is the heavy weight task.                            | <b>2</b> While creating of thread is light weighted.          |
| <b>3</b> There can be limited number of process with in any operating system.     | <b>3</b> There can be unlimited thread within any process.    |
| <b>4</b> Process can never share the data.                                        | <b>4</b> Thread can share the data.                           |
| <b>5</b> Switching between the processes is heavy weighted task.                  | <b>5</b> Switching between the thread is light weighted task. |

Java is multithreaded programming language that means program develop in java are multithreaded program that means when the JRE started it start the multiple thread such as garbage collector, class loader, main thread etc.





Main thread is responsible for the execution of java.

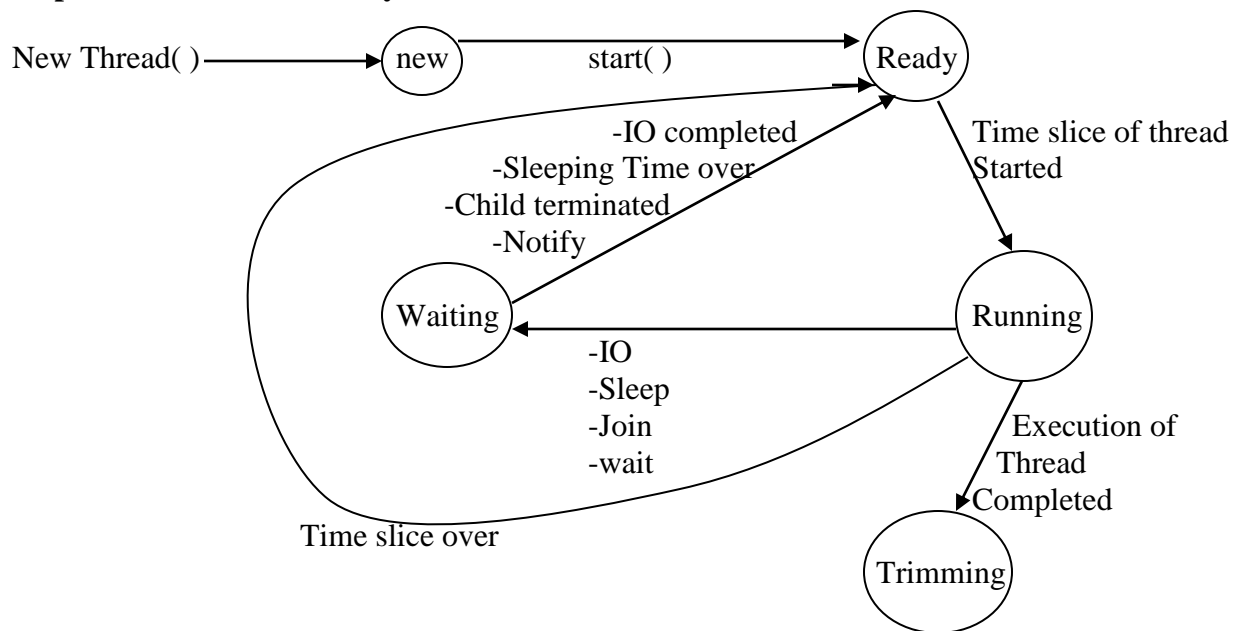
[**Note:** - Each thread has their separate stack for the execution of the method.]

### Java.lang.Thread class

An instance of this class represents to one thread.

Thread class provides the non-static and static methods to manage the state of the thread in their life cycle.

#### **Steps of thread in this life cycle**



### Method of Thread class

- **public void setName(String name)**

This method is used to set the name of the thread.

- **public String getName( )**

This method is using to returns the name of thread.

By default each thread has their default name as follow:

Thread – 0    Thread – 1    Thread – 2    as so on.

- **public void setPriority(int priority)**

This method is use to set the priority of the thread.

Priority of the thread can be within 1 to 10, one is minimum priority and 10 is the maximum priority.

**There are static constant to set the priority**

- **Thread.MAX**
- **Thread.NORMAL**
- **Thread.MIN**

**Ex:**

Thread.setPriority(Thread.MIN\_PRIORITY)

**Or**

Thread.setPriority(1)

- **public int getPriority( )**

This method returns the current priority of thread. Hierarchy of thread.

- **public static Thread currentThread( )**

This method returns the reference id of current thread.

- **public static void sleep(long millisecond)throws InterruptedException**

This method is use to sleep the thread and moved into the waiting state.

- **public void join( )throws InterruptedException**

This method is use to join one thread with other thread.

- **public void start( )**

This method is use to start the thread.

**[Note: - In the life of many thread object this method can be invoked only once.]**

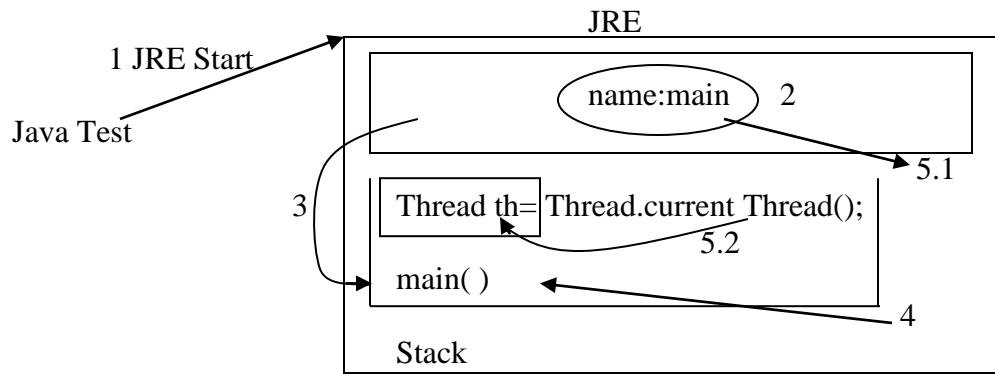
- **public boolean isAlive( )**

This method returns the Boolean value either true or false depends upon whither a thread is moved into the terminating state or not.

- **public void interrupt( )throws InterruptedException**

This method is use to interrupt the thread when a thread is in the waiting state to move the thread in ready state.

**[Note: - By using the current thread( ) method we can get the reference id of the main thread in the main method.]**



1. JRE Started
2. Object of Thread created (new Thread( )).
3. New Stack created for Thread.
4. main( ) method started.
- 5.1 Reference id Thread object obtained.
- 5.2 Reference id copied into **th**.

### Creating a program to manage the main thread

// ThreadTest.java

class ThreadTest

{

public static void main(String s[ ])

{

System.out.println("main thread started");

Thread th = Thread.currentThread();

System.out.println("Thread name: "+th.getName());

System.out.println("Thread Priority "+th.getPriority());

System.out.println("main Thread is going to sleep for 5 seconds .....");

try

{

Thread.sleep(5000);

}

catch(InterruptedException e)

{

}

System.out.println("changing the name and priority");

th.setName("My Main");

th.setPriority(Thread.MAX\_PRIORITY);

System.out.println("New Name: "+th.getName());

System.out.println("New Priority: "+th.getPriority());

System.out.println("is alive: "+th.isAlive());

System.out.println("Details of thread - [thread name:priority:group] "+th);

//toString()

}

}

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading>javac ThreadTest.java
E:\JAVA\Programs\Core Java Program\Multithreading>java ThreadTest
main thread started
Thread name: main
Thread Priority 5
main Thread is going to sleep for 5 seconds
changing the name and priority
New Name: My Main
New Priority: 10
is alive: true
Details of thread - [thread name:priority:group] Thread[My Main,10,main]
```

[Note: - By default the name of main thread is main.]

[Note: - By default the priority of any thread is the NORMAL i.e. 5.]

[Note: - In java working of any thread is totally depends upon the two things]

1. Thread object
2. Associated stack.

Some methods in thread class are static and non-static.

Through the static method only the methods running in the stack of the any thread can control that thread.

Through the non-static methods by getting the reference id of first thread the any methods running in the stack of any other thread can control to the first thread.

### Creating a new user defined thread.

#### Java.lang.Runnable Interface

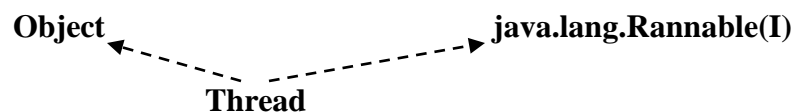
This interface is used to make the any runnable class that means we have to create a new class by implementing runnable interface.

#### Method of this Interface

- **public void run( )**

In any new thread the run( ) method always be the first method to be loaded in the new stack of the new thread.

**Java.lang.Thread** class is also the implemented class of the Runnable interface.



The run( ) method in the Thread class is empty method and object of the Thread class also be used as the implemented class of Runnable(I) but it cannot do anything.

### There are two ways to create any user defined thread.

1. By extending the Thread class
2. By Implementing the Runnable interface

## New Thread by implementing Runnable interface

//UserThreadTest.java

class MyThread implements Runnable

```
{
 public void run()
 {
 Thread th = Thread.currentThread();
 String tname = th.getName();
 System.out.println(tname+" thread started");
 for(int i = 1; i<=10; i++)
 {
 System.out.println(tname+": "+i);
 }
 }
}

class UserThreadTest
{
 public static void main(String s[])
 {
 System.out.println("main thread started");
 MyThread mth = new MyThread();
 Thread th = new Thread(mth);
 th.start(); //to start the thread , internally start() invokes the run()

 for(int i = 1; i<=10; i++)
 {
 System.out.println("main thread: "+i);
 }
 }
}
```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\Thread2>javac UserThreadTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\Thread2>java UserThreadTest
main thread started
main thread: 1
main thread: 2
main thread: 3
main thread: 4
main thread: 5
main thread: 6
main thread: 7
main thread: 8
main thread: 9
main thread: 10
Thread-0 thread started
Thread-0: 1
Thread-0: 2
Thread-0: 3
Thread-0: 4
Thread-0: 5
Thread-0: 6
Thread-0: 7
Thread-0: 8
Thread-0: 9
Thread-0: 10
```

### Constructor of Thread class

- **public Thread(Runnable r)**
- **public Thread(String name)**
- **public Thread(Runnable r, String name)**
- **public Thread( )**

[**Note:** - First or thread constructor is used when we create Thread by implementing the Runnable interface.]

### **Code of the Thread class**

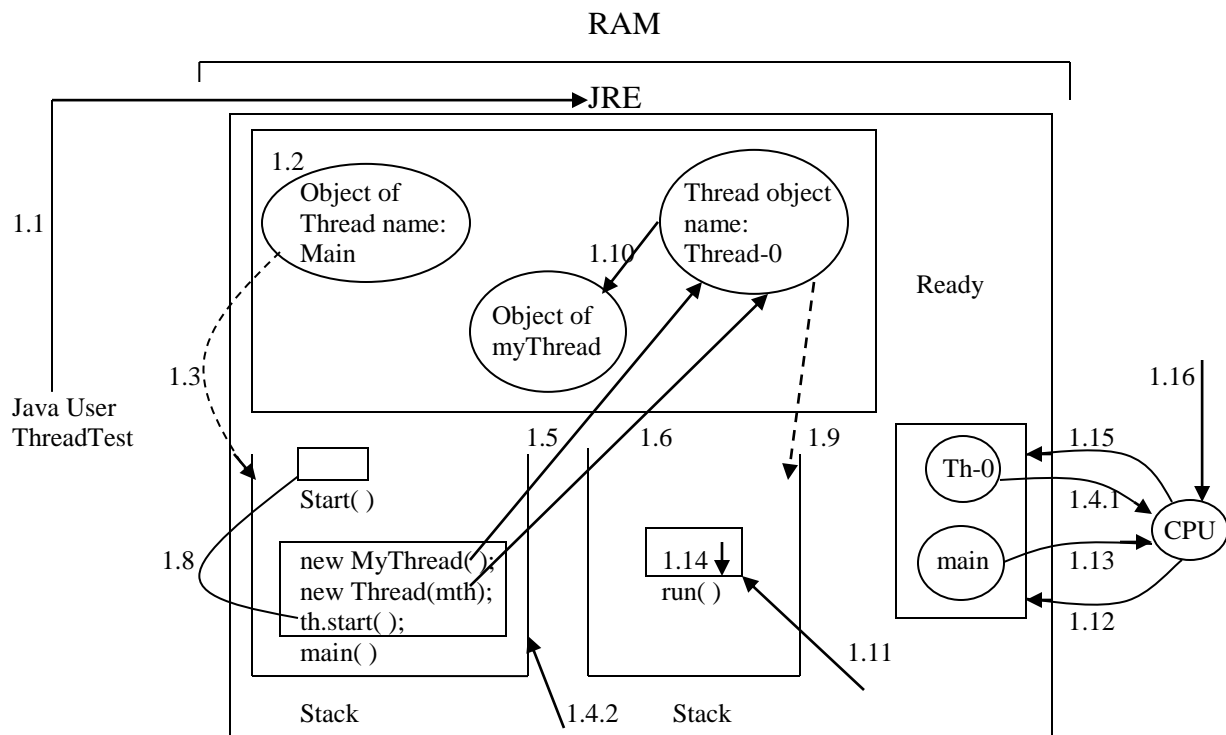
**//Thread.java**

```
class Thread implements Runnable
{
 Runnable ref;
 String name;
 public Thread()
 {
 }
 public Thread(Runnable r)
 {
 this.ref = r;
 }
 public Thread(String name)
 {
 this.name=name;
 }
}
```

```

 }
 public Thread(Runnable r, String name)
 {
 this.ref = r;
 this.name = name;
 }
 public void start()
 {
 if(ref == null)
 {
 this.run();
 }
 else
 {
 ref.run();
 }
 }
 }//run() always be loaded into the new stake other methods.
}

```



- 1.1 JRE started.
- 1.2 New Thread created.
- 1.3 New stack prepared and registered with thread.
- 1.4.1 main( ) started.
- 1.4.2 Main thread is moved in the running state to execute the main( ) method in their stack.
- 1.5 new MyThread( ) object created.

- 1.6 new Thread object created and reference of the MyThread object is passed in the argument.
- 1.7 MyThread object is pointed by Thread object.
- 1.8 start( ) method invoked with the new Thread object.
- 1.9 new static created and associated with the Thread object.
- 1.10 From the start( ) method run( ) method is invoked on the MyThread object.
- 1.11 New frame is created for the run( ) method in the new stack.
- 1.12 Time slice of main Thread is over and main thread is moved in the ready queue.
- 1.13 Thread-0 gets the CPU cycle.
- 1.14 Instructions of run( ) method executed.
- 1.15 Time slice of Thread-0 is over and thread is moved in the ready state.
- 1.16 Main thread is moved in the running state.

### **Creating a new Thread by extending the Thread class**

**//UserThreadTest2.java**

class MyThread extends Thread

```
{
 public void run()
 {
 String tname=getName();
 System.out.println(tname+"thread start");
 for(int i=1; i<=10; i++)
 {
 System.out.println(tname+" : "+i);
 }
 }
}
class UserThreadTest2
{
 public static void main(String s[])
 {
 System.out.println("main thread started ");
 MyThread mth = new MyThread();
 mth.start();
 for(int i=1; i<=10; i++)
 {
 System.out.println("main thread: "+i);
 }
 }
}
```



## Thread Interruption

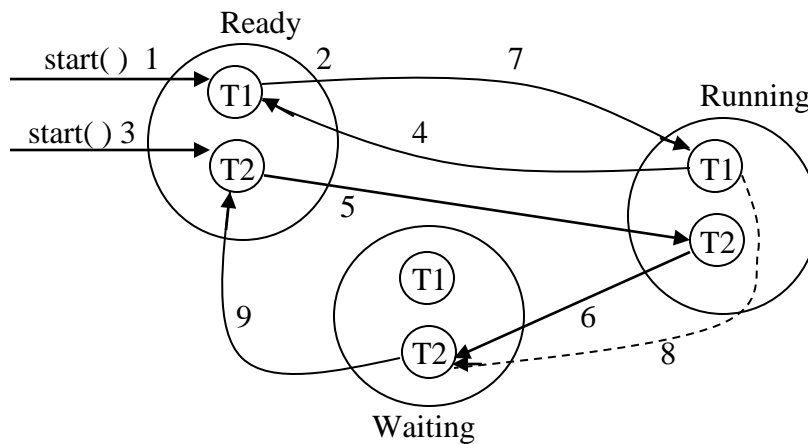
Whenever any thread is in the waiting state at that time another thread interrupt to that thread.

### The process of thread interrupts as follow

Whenever main thread is in waiting state it continuously monitor to their interrupted flag (Boolean) variable if the value of flag of Boolean variable is found as true then immediately thread generates thread interrupted exception and moved into the ready state.

#### - **public void interrupt()**

This is the method of thread class used to set the interrupted flag variable.



1. T1 thread start
2. T1 thread gets the CPU time slice.
3. T2 thread started.
4. Time slice of T1 thread is over and T1 thread is return into the ready state.
5. T2 thread is moved in the running state.
6. Sleep() method invoked at T2 thread is moved in the waiting state.
7. T1 thread gets the CPU cycle again.
8. Interrupt() method is invoked by the T1 with the reference id of the T2 thread object.
9. T2 thread generates the interrupts exception and moved in the ready state.

#### // InterruptTest.java

```
class MyThread extends Thread
{
 int time; //time=5(from command line)
 MyThread(int time)
 {
 this.time = time;
 start();
 }
 public void run()
 {
 String name = getName();
```

```

 System.out.println(name+" thread started");
 try
 {
 System.out.println("Thread is going to sleep for time sec.....");
 sleep(time*1000);
 }
 catch(InterruptedException e)
 {
 System.out.println(name+" thread gets interrupted");
 }
 System.out.println(name+" Thread is about to terminate");
 }
}
class InterruptTest
{
 public static void main(String s[])
 {
 System.out.println("main thread started");
 int x = Integer.parseInt(s[0]);
 MyThread mth = new MyThread(x);
 try
 {
 System.out.println("main is going to sleep");
 Thread.sleep(3000);
 }
 catch(Exception e)
 {
 }
 System.out.println("main is awaked interrupting to the user thread:");
 mth.interrupt();
 System.out.println("main is about to terminate");
 }
}

```

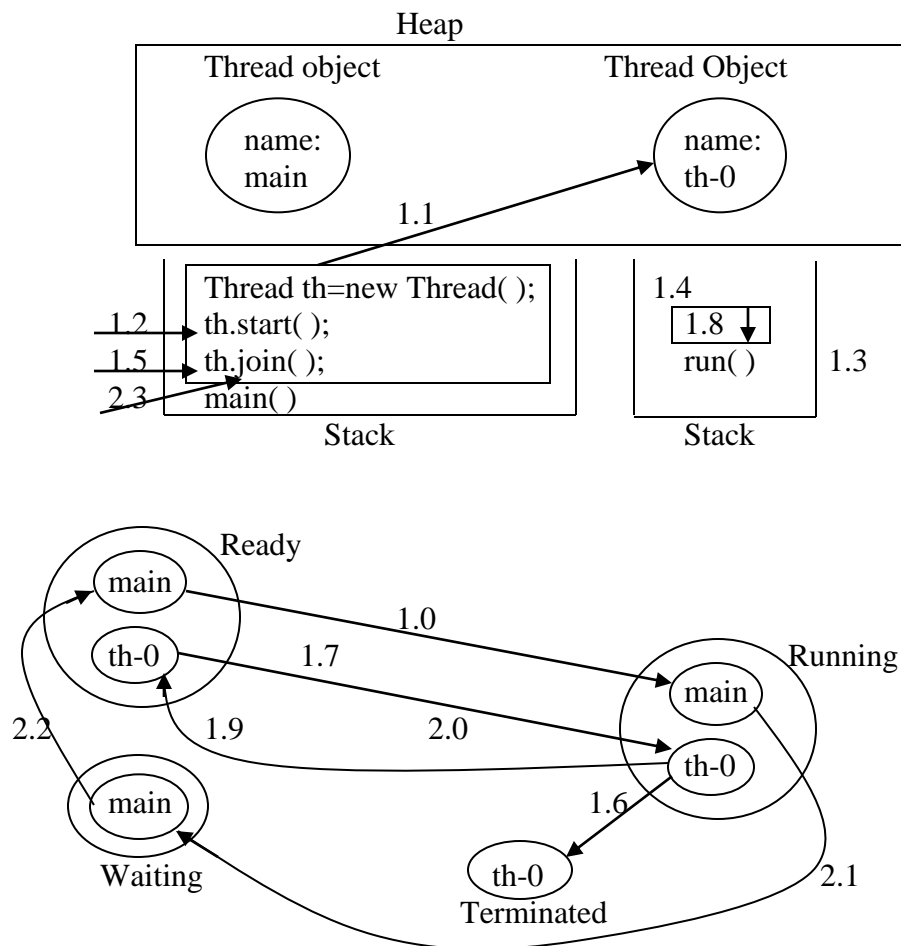
## **Thread Joining**

Thread joining is the concept that facilitates to join one thread with another thread.

In this process two threads will be first one is the joining thread and another is the join thread.

Joining thread will wait for the termination of the joint thread.

The object whose reference id is used to call the join( ) method will be the joint thread and the thread moves stacks is used to invoke the join( ) method will be the joining thread.



- 1.0 main thread is moved in the running state.
- 1.1 new Thread( ) object created from the main method.
- 1.2 start( ) method invoked from the new Thread( ) object.
- 1.3 new stack created.
- 1.4 run( ) method started from the start( ) method in the new stack.
- 1.5 join( ) method invoked by the main thread of the thread object.
- Main thread becomes the joining thread and th-0 becomes the joint thread.
- 1.6 main threads is moved in the waiting started.
- 1.7 th-0 gets the CPU cycle.
- 1.8 Exception of run( ) method started.
- 1.9 Time slice of th-0 is over.
- 2.0 th-0 thread also gets the CPU cycle.
- 2.1 th-0 thread gets terminated.
- 2.2 main thread is moved in the ready state.

```
// ThreadJoinTest.java
import java.io.*;
class InputThread extends Thread
{
 int data[];
 InputThread(String name)
```

```

 {
 super(name);
 data = new int[10];
 start();
 }
 public void run()
 {
 String tname = getName();
 System.out.println(tname+" thread started");
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter 10 values");
 try
 {
 int i = 0;
 while(i<10)
 {
 data[i] = Integer.parseInt(br.readLine());
 i++;
 }
 }
 catch(Exception e)
 {
 }
 System.out.println(tname+" is to mfinish");
 }
}
class ThreadJoinTest
{
 public static void main(String s[])
 {
 InputThread ith = new InputThread("input");
 try
 {
 ith.join();
 }
 catch(InterruptedException e)
 {
 e.printStackTrace();
 }
 int sum = 0;
 for(int i=0; i<10; i++)
 {
 sum = sum + ith.data[i];
 }
 System.out.println("Sum = "+sum);
 }
}

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\ThreaJoin>javac ThreadJoinTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\ThreaJoin>java ThreadJoinTest
input thread started
Enter 10 values
1
2
3
4
5
6
7
8
9
10
input is to finish
Sum = 55
E:\JAVA\Programs\Core Java Program\Multithreading\ThreaJoin>
```

## **Daemon Thread**

Daemon threads are the secondary threads.

The instance of the JRE never waits for the termination of the Daemon Thread.

By default all the thread are not daemon `setDaemon()` method is used to make any thread daemon.

- **`public void setDaemon(boolean b)`**

[**Note:** - We can only invoke the `setDaemon()` method before the starting of the thread.]

```
import java.io.*;
class InputThread extends Thread
{
 int data[];
 InputThread(String name)
 {
 super(name);
 data = new int[10];
 setDaemon(true);
 start();
 }
 public void run()
 {
 String tname = getName();
 System.out.println(tname+" thread started");
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("Enter 10 values");
 try
 {
 int i = 0;
```

```

 while(i<10)
 {
 data[i] = Integer.parseInt(br.readLine());
 i++;
 }
 }
 catch(IOException e)
 {
 }
 System.out.println(tname+" is to finish");
}
}
class DaemonThreadTest
{
 public static void main(String s[])
 {
 InputThread ith = new InputThread("input");
 try
 {
 Thread.sleep(5000);
 }
 catch(InterruptedException e)
 {
 e.printStackTrace();
 }
 int sum = 0;
 for(int i=0; i<10; i++)
 {
 sum = sum + ith.data[i];
 }
 System.out.println("Sum = "+sum);
 }
}

```

## **Thread Synchronization**

Synchronization means to access the resource in the mutually exclusive manner such as when one resource is accessed by the one thread then no other thread will be able to access that resource at that time.

In java there is the keyword synchronized to make the non-static method synchronized.

Any object always has a lock for all their synchronized method.

If any thread invokes the any synchronized non-static method from any object then immediately the lock of that object is given to that thread.

No other thread can synchronize method from that same object at same time and that other threads will move on the waiting state and wait for the lock.

```

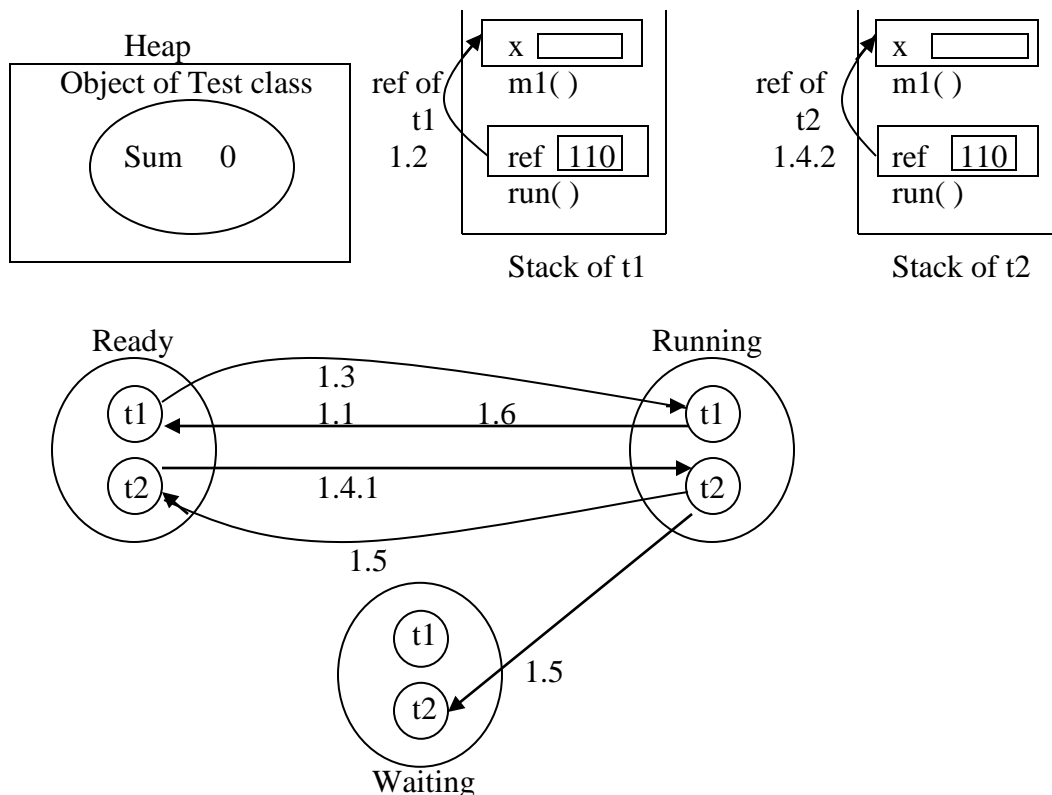
class Test
{

```

```

int sum;
synchronized void m1()
{
 int x;
 sum=0;
 for(int i=1; i<=10;++)
 {
 x=(int)(Math.random()*100);
 sum=sum+x;
 }
 System.out.println("sum "+sum);
}
}

```



- 1.1 t1 Thread is moved in the running state.
- 1.2 From the run method m1( ) method is moved and random value generation started to calculate the sum.
- 1.3 Time scale of t1 thread finished and t1 thread is moved in the running state.
- 1.4.1 t2 thread is moved in the running state.
- 1.4.2 lock is not available in the test object so m1( ) method could not be invoked.
- 1.5 t2 thread is moved in the waiting state.
- 1.6 t1 thread entered in the running state again and the execution of m1( ) method started again.

```

//SynchTest.java
class Resource
{
 int data[] = { 1,2,3,4,5,6,7,8,9,10};
 synchronized void display()
 {
 System.out.println("In the display");
 for(int i = 0; i < data.length; i++)
 {
 System.out.println((i+1)+" value: "+data[i]);
 }
 }
 synchronized void update()
 {
 for(int i = 9; i >= 0; i--)
 {
 data[i]+=10;
 System.out.println((i + 1)+" Updated values:" +data[i]);
 }
 }
}

class MyThread extends Thread
{
 Resource r;
 MyThread(Resource r, String name)
 {
 super(name);
 this.r = r;
 start();
 }
 public void run()
 {
 String tname = getName();
 if(tname.equals("first"))
 r.display();
 if(tname.equals("second"))
 r.update();
 }
}

class SynchTest
{
 public static void main(String s[])
 {
 Resource r = new Resource();
 MyThread th1 = new MyThread(r, "first");
 MyThread th2 = new MyThread(r, "second");
 }
}

```



```

}
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>javac SynchTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>java SynchTest
10 Updated values:20
9 Updated values:19
8 Updated values:18
7 Updated values:17
6 Updated values:16
5 Updated values:15
4 Updated values:14
3 Updated values:13
2 Updated values:12
1 Updated values:11
In the display
1 value: 11
2 value: 12
3 value: 13
4 value: 14
5 value: 15
6 value: 16
7 value: 17
8 value: 18
9 value: 19
10 value: 20
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>_

```

## **Synchronized Block**

It block that always takes the reference id of the object in the argument.

If the block of the argument object is free then control can be entered in the synchronized block otherwise the thread which is running the synchronized block immediately moved the waiting state and wait for the block.

**There are following two uses of synchronized block.**

1. Invoke the non-static method in the synchronized ways from outside the class.
2. To make any non-static partially synchronized.

**Firs use: →**

1. Remove the synchronized keyword for the display( ) and update( ) method.
2. Make the synchronized block in the run( ) method.

**//SynchTest1.java**

class Resource

```

{
 int data[] = {1,2,3,4,5,6,7,8,9,10};
 synchronized void display()
 {
 System.out.println("In the display");
 for(int i = 0; i < data.length; i++)
 {
 System.out.println((i+1)+" value: "+data[i]);
 }
 }
}

```

```

 }
 synchronized void update()
 {
 for(int i = 9; i >= 0; i--)
 {
 data[i]+=10;
 System.out.println((i + 1)+" Updated values:" +data[i]);
 }
 }
}
class MyThread extends Thread
{
 Resource r;
 MyThread(Resource r, String name)
 {
 super(name);
 this.r = r;
 start();
 }
 public void run()
 {
 String tname=getName();
 synchronized(r)
 {
 if(tname.equals("first"))
 r.display();
 if(tname.equals("Second"))
 r.display();
 }
 }
}
}
class SynchTest1
{
 public static void main(String s[])
 {
 Resource r = new Resource();
 MyThread th1 = new MyThread(r, "first");
 MyThread th2 = new MyThread(r, "second");
 }
}

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>javac SynchTest1.java
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>java SynchTest1
In the display
1 value: 1
2 value: 2
3 value: 3
4 value: 4
5 value: 5
6 value: 6
7 value: 7
8 value: 8
9 value: 9
10 value: 10
```

**Second use: →**

**//SynchTest2.java**

class Resource

```
{
 int data[] = {1,2,3,4,5,6,7,8,9,10};
 void display()
 {
 System.out.println("In the display");
 System.out.println("10 times loop execution started.....");
 for(int i = 0; i <=10; i++)
 {
 System.out.println("Display: "+i);
 }
 synchronized(this)
 {
 for(int i = 0; i < data.length; i++)
 {
 System.out.println((i+1)+" value: "+data[i]);
 }
 }
 }
 void update()
 {
 System.out.println("update method 10 times loop execution started..... ");
 for(int i = 1; i<=10;i++)
 {
 System.out.println("Update: "+i);
 }
 synchronized(this)
 {
 for(int i = 9; i >= 0; i--)
 {
 data[i]+=10;
 System.out.println((i + 1)+" Updated values:" +data[i]);
 }
 }
 }
}
```

```

 }
 }
}
class MyThread extends Thread
{
 Resource r;
 MyThread(Resource r, String name)
 {
 super(name);
 this.r = r;
 start();
 }
 public void run()
 {
 String tname = getName();
 if(tname.equals("first"))
 r.display();
 if(tname.equals("second"))
 r.update();
 }
}
class SynchTest2
{
 public static void main(String s[])
 {
 Resource r = new Resource();
 MyThread th1 = new MyThread(r, "first");
 MyThread th2 = new MyThread(r, "second");
 }
}

```

```
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>javac SynchTest2.java
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>java SynchTest2
update method 10 times loop execution started.....
In the display
Update: 1
Update: 2
10 times loop execution started.....
Update: 3
Display: 0
Display: 1
Update: 4
Update: 5
Display: 2
Update: 6
Display: 3
Update: 7
Display: 4
Display: 5
Update: 8
Display: 6
Update: 9
Display: 7
Update: 10
Display: 8
10 Updated values:20
```

## **Inter Thread Communication**

Synchronized keyword provides the mutual exclusivity but not in order that means if one thread entered into any synchronized method then it gets the lock for the long time so other threads have to wait for the lock for long time.

Inter Thread communication is the way through which any threads can realize the lock in between the execution of synchronize method so that the other thread can get the lock.

There are the methods in `java.lang.Object`

1. `wait()`
2. `notify()`
3. `notifyAll()`

### **wait()**

`wait()` method as thread is moved into the waiting state by the `main()` method thread just start the waiting for the notification.

### **notify()**

`notify()` method is used to notify the thread waiting in the waiting state.

After getting notified the thread remains in the waiting state and wait for the lock.

### **notifyAll()**

This method will notify all the threads waiting in the waiting state.

//InterThreadCommTest.java

```
class Resource
{
```

```

int data;
boolean flag;
synchronized void produce()
{
 try
 {
 for(int i = 1; i<=10; i++)
 {
 if(flag)
 this.wait();
 data = (int)(Math.random()*1000);
 System.out.println(data+" is generated");
 notify();
 flag = true;
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
synchronized void consume()
{
 try
 {
 for(int i=1; i<=10; i++)
 {
 if(!flag)
 this.wait();
 if(data%2==0)
 System.out.println(data+" is even");
 else
 System.out.println(data+" is odd");
 notify();
 flag = false;
 }
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}
class MyThread extends Thread
{
 Resource r;
 MyThread(String name, Resource r)

```

```

 {
 super(name);
 //this.name = name;
 this.r = r;
 start();
 }
 public void run()
 {
 String name = getName();
 if(name.equals("producer"))
 r.produce();
 if(name.equals("consumer"))
 r.consume();
 }
}
class InterThreadCommTest
{
 public static void main(String s[])
 {
 Resource r = new Resource();
 MyThread th1 = new MyThread("producer", r);
 MyThread th2 = new MyThread("consumer", r);
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>javac InterThreadCommTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>java InterThreadCommTest
160 is generated
160 is even
803 is generated
803 is odd
955 is generated
955 is odd
528 is generated
528 is even
926 is generated
926 is even
293 is generated
293 is odd
516 is generated
516 is even
81 is generated
81 is odd
722 is generated
722 is even
303 is generated
303 is odd
E:\JAVA\Programs\Core Java Program\Multithreading\SynchronizationThread>

```

## **Reentrant Locking**

This is the new concept introduced in the JDK 1.5 according to this concept more than synchronized method of any one object can be safe separately with the help of different locks.

## **Java.util.concurrent.locks.Reentrant class**

An instance of this class provides two methods

1. lock()
2. unlock()

By using these methods we can maintain different group of synchronized( ) method for one object.

### **// ReentrantTest.java**

```
import java.util.concurrent.locks.ReentrantLock;
class Resource
{
 int x,y;
 ReentrantLock r1 = new ReentrantLock();
 ReentrantLock r2 = new ReentrantLock();
 void m1()
 {
 r1.lock();
 for(int i=1; i<=10; i++)
 {
 int val = (int)(Math.random()*100);
 x+=val;
 System.out.println("value of x in m1: "+x);
 }
 System.out.println("Final value of x in m1 "+x);
 r1.unlock();
 }
 void m2()
 {
 r1.lock();
 for(int i=1; i<=10; i++)
 {
 int val = (int)(Math.random()*100);
 x-=val;
 System.out.println("value of y in m2: "+x);
 }
 System.out.println("Final value of x in m2 "+x);
 r1.unlock();
 }
 void m3()
 {
 r2.lock();
 for(int i=1; i<=10; i++)
 {
 int val = (int)(Math.random()*100);
 y+=val;
 System.out.println("value of y in m3: "+y);
 }
 }
}
```



```

 }
 System.out.println("Final value of y in m3 "+y);
 r2.unlock();
 }
 void m4()
 {
 r2.lock();
 for(int i=1; i<=10; i++)
 {
 int val = (int)(Math.random()*100);
 y-=val;
 System.out.println("value of y in m4: "+y);
 }
 System.out.println("Final value of y in m4 "+y);
 r2.unlock();
 }
}
class MyThread extends Thread
{
 Resource r;
 MyThread(Resource r, String name)
 {
 super(name);
 this.r = r;
 start();
 }
 public void run()
 {
 if(getName().equals("first"))
 r.m1();
 if(getName().equals("second"))
 r.m2();
 if(getName().equals("third"))
 r.m3();
 if(getName().equals("fourth"))
 r.m4();
 }
}
class ReentrantTest
{
 public static void main(String s[])
 {
 Resource r = new Resource();
 new MyThread(r, "first");
 new MyThread(r, "second");
 new MyThread(r, "third");
 }
}

```

```

 new MyThread(r, "fourth");
 }
}

```

```

C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\ReentrantLock>javac ReentrantTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\ReentrantLock>java ReentrantTest
value of y in m3: 91
value of x in m1: 37
value of y in m3: 113
value of x in m1: 60
value of y in m3: 125
value of y in m3: 182
value of x in m1: 105
value of y in m3: 274
value of x in m1: 141
value of y in m3: 321
value of x in m1: 179
value of y in m3: 357
value of x in m1: 273
value of y in m3: 409
value of x in m1: 280
value of y in m3: 459
value of y in m3: 478
value of x in m1: 378
Final value of y in m3 478
value of x in m1: 473
value of y in m4: 413
value of y in m4: 318
value of x in m1: 508
Final value of x in m1 508
value of y in m4: 260

```

## Thread Group

Thread group is the concept through which multiple threads can be grouped together so that any specific operation on the multiple threads can be performed in one attempt to such as interruption etc.

### **Java.lang.ThreadGroup class**

An instance of this class represents to the Thread Group.

### Constructor

- **public ThreadGroup(String groupName)**

To make the thread member of the group we have to use the one of the following constructor of thread class.

- **public Thread(ThreadGroup group, String threadName)**
- **public Thread(ThreadGroup group, Runnable r)**

### Method of the ThreadGroup

- **public int activeCount( )**

This method returns the number of threads running in this program.

- **public void interrupt( )**

This method is use to all the member threads of this group.

**//ThreadGroupTest.java**

```
class MyThread extends Thread
{
 int time;
 MyThread(ThreadGroup tg, String name, int t)
 {
 super(tg,name);
 this.time=t;
 start();
 }
 public void run()
 {
 String name=getName();
 System.out.println(name+"Thread started");
 try
 {
 System.out.println(name+"Thread is going into sleep");
 sleep(time*1000);
 }
 catch(InterruptedException e)
 {
 System.out.println(name+"Thread gets interrupted");
 }
 }
}
class ThreadGroupTest
{
 public static void main(String s[])
 {
 MyThread mth1, mth2, mth3;
 ThreadGroup group=new ThreadGroup("Test Group");
 mth1=new MyThread(group,"Lazy1",3);
 mth2=new MyThread(group,"Lazy2",5);
 mth3=new MyThread(group,"Lazy3",7);
 try
 {
 Thread.sleep(1000);
 }
 catch(Exception e)
 {
 }
 System.out.println("main is awaked and interrupting the member threads.....");
 group.interrupt();
 }
}
```

```

}
C:\Windows\System32\cmd.exe
E:\JAVA\Programs\Core Java Program\Multithreading\ThreadGroup>javac ThreadGroupTest.java
E:\JAVA\Programs\Core Java Program\Multithreading\ThreadGroup>java ThreadGroupTest
Lazy2Thread started
Lazy3Thread started
Lazy3Thread is going into sleep
Lazy1Thread started
Lazy2Thread is going into sleep
Lazy1Thread is going into sleep
main is awaked and interrupting the member threads.....
Lazy3Thread gets interrupted
Lazy2Thread gets interrupted
Lazy1Thread gets interrupted
E:\JAVA\Programs\Core Java Program\Multithreading\ThreadGroup>

```

## **Shutdown Hookup**

Shutdown hookup is the way through which a new thread is created and registered with the runtime environment so that at the ending of the program get thread automatically gets started.

Hookup thread always created normally as the other through are created but they never be started rather programmatically started rather automatically started when the all other threads get completed.

## **Java.lang.Runtime class**

An instance of runtime class is use to represent the runtime environment.

The factory method of runtime class itself returns the object of runtime.

- **Public static Runtime getRuntime( )**

## **addShutdownHook( )**

**addShutdownHook( )** method of the Runtime class is use to register the thread with the runtime.

- **public void addShutdownHook(Thread th)**

### **//HookupTest.java**

```
class ShutdownHookup extends Thread
```

```
{
 public void run()
 {
 System.out.println("Hookup thread started");
 System.out.println("cleaning task can be performed here");
 }
}
```

```
class HookupTest
```

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

```

```

 {
 System.out.println("main started");
 ShutdownHook hk=new ShutdownHook();
 Runtime r=Runtime.getRuntime();
 r.addShutdownHook(hk);
 System.out.println("main thread is about to finish");
 }
}

```

## **Thread Deadlock**

Deadlock situation where one thread is wait for lock occupied by 2<sup>nd</sup> thread and 2<sup>nd</sup> thread wait for the lock of object occupied by the 1<sup>st</sup> thread.

In that case both the thread will remain in the waiting state and wait for lock occupied by the each other.

```

class Resource
{
 int sum;
 synchronized void sumCalc()
 {
 String tname=Thread.currentThread().getName();
 for(int i=1; i<=10; i++)
 {
 int x=(int)(math.random()*100);
 sum=sum+x;
 }
 System.out.println("sum of "+name+" = "+sum);
 if(tname.equals("user1"))
 {
 System.out.println("try to get the sum of user2, may be in deadlock...");
 int otherSum=MyFactory.r2.getSum();
 System.out.println("Total sum in user1:"+(sum+oterSum));
 }
 synchronized int getSum()
 {
 return(sum);
 }
 }
}

class MyFactory
{
 static resource r1, r2;
 static
 {
 r1=new Resource();
 r2=new Resource();
 }
}

```

```

}
class MyThread extends Thread
{
 MyThread(String name);
 {
 super(name);
 start();
 }
 public void run()
 {
 if(getName().equals("user1"))
 MyFactory.r1.sumCalc();
 if(getName().equals("user2"))
 MyFatory.r2.sumCalc();
 }
}
class DeadLockTest
{
 public static void main(String s[])
 {
 new MyThread("user1");
 new MyThread("user2");
 }
}

```

### **Synchronization with static method**

For the synchronization of static method there is also a single lock available for the all class as any threads invokes to the any static synchronize( ) method lock of that class automatically giving to that thread.

```

class Test
{
 synchronized static void m1()
 {
 try
 {
 System.out.println("Thread from m1() going into sleep.");
 Thread.sleep(3000);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 System.out.println("Thread from m1() awaked");
 }
 synchronized static void m2()
 {

```

```

 try
 {
 System.out.println("Thread from m2() going into sleep");
 Thread.sleep(3000);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
 System.out.println("Thread from m2() awaked");
 }
}
class MyThread extends Thread
{
 MyThread(String name)
 {
 super(name);
 start();
 }
 public void run()
 {
 String name=getName();
 if(name.equals("first"))
 Test.m1();
 if(name.equals("second"))
 Test.m2();
 }
 public static void main(String s[])
 {
 new MyThread("first");
 new MyThread("second");
 }
}

```