

2017

Oops through JAVA

UNIT-1



JAVA

Author	: James Gosling
Vendor	: Sun Microsystems (from 2010 oracle corporation)
Type	: open source
Initial name	: Oak Languag
Project name	: Green Project
Present name	: java
First release	: 1995
Initial version	: JDK1.0
Extensions	: .java, .class, .jar
Operating systems	:Multiple Operating systems
Implementation languages	: c , c++
Current versions	: java SE8 (released in 2014)
Objectives	: to develop web applications
Moto	: WORA (Write Once Run Anywhere)

PROGRAMMING PARADIGM:

The lowest level paradigm is machine level language which directly represents the information (the contents of the program) has a sequence of digits 0 & 1

Characteristics of machine level language:

1. Computers can understand and execute it directly
2. It uses binary
3. Machine dependent
4. Time consuming

Limitations of machine level language:

1. Binary and hexadecimal code is virtually unreadable.
2. Machine language commands are few and private, making programs hard to write.
3. Machine language programs are hard to maintain and debug.

PROCEDURAL ORIENTED PROGRAMMING LANGUAGE:

Procedural oriented programming language is that which uses a common English language or vocabulary is used to write the program which describes a step by step, the procedures that should be used to write a program.

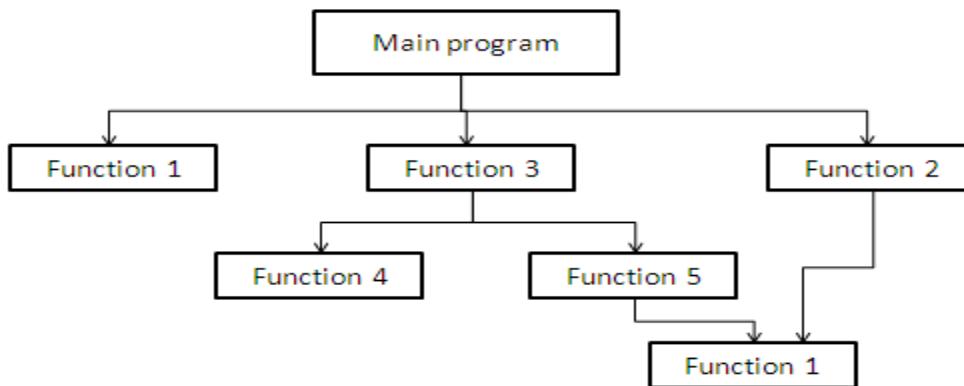
The procedural oriented programming language is made up of functions to solve the given problem.

It focuses more on functions as well as sequence of actions to be done besides the data part.

It follows the top down approach.

Characteristics of procedural oriented programming language:

1. Concentrates more on functioning of the program i.e , it has to focuses what has to happen in program.
2. Large programs are divided into smaller programs called as functions.
3. The data movement is not restricted in procedural oriented programming language (ex: global variables).
4. Functions can transfer data from one form to another .



Limitations :

1. A procedural oriented programming language concentrates on what is happening (functions) rather than what is effected (data).
2. Less reusable

3. The procedural language not usable for abstraction concept.
4. In multi functional programming many important data are taken as global so that they can be accessed by the functions which are required. The global data are more vulnerable to an advent change in program.
5. The global data can also be accessed by other functions which are not to be done.



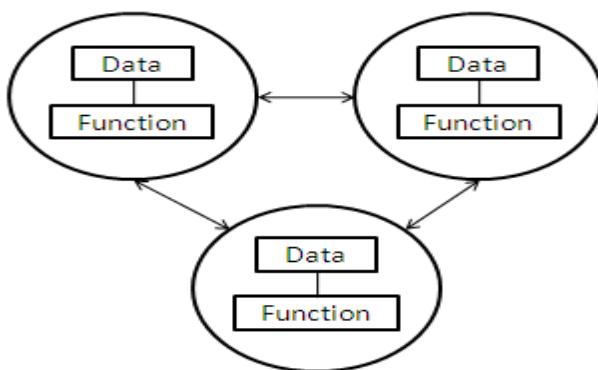
Function 1 is authorized to global data 1 & global data 2

Function 2 is authorized to global data 3

As global data1 & global data 2 are already used by function 1. So, function 2 cannot access global data 1 & global data 2 in procedural oriented programming language.

OBJECT ORIENTED PROGRAMMING LANGUAGE:

- The major motivation of object oriented approach is to overcome some of flaws of procedural oriented programming language.
- Object oriented programming treats data as a critical element in program development and does not allow it to flow freely in the system. It ties the data and its functions which operates on it closely and protects from accessing and modification by other functions.
- The object oriented programming allows a problem to decompose into smaller no of entities called objects and then builds data and functions around
- The data of an object can be accessed by the function associated with the object i.e, the function of one object can access function of other object.



- The object oriented programming is that which is based on the concepts of objects which contains the data in terms of fields which is referred as attributes and the code in the form of procedures can be referred to as methods.

Characteristics:

1. Emphasis is on data rather than procedure
2. Programs are divided into objects.
3. Data is hidden and cannot be accessed by external functions directly.

Differences between procedural oriented and object oriented

Procedural oriented	Object oriented
1. Programs are divided into smaller parts called as functions.	1. Programs are divided into smaller parts called as objects.
2. Top down approach	2. Bottom up approach
3. Importance is not given to data but focus is on functions and sequence of actions to be done.	3. Importance is given to both data and methods.
4. Data movement is free from one function to other function.	4. Communication is between objects.
5. Overloading is not possible.	5. Overloading is possible like method overloading and operator overloading.
6. We dont have any access specifiers.	6. We have access specifiers like public,private,protected...etc.
7. Most of the function using global data for sharing & which can be accessed freely by every function in the program.	7. Data cannot be move freely from function to function because of the access specifiers concept.
8. Data hiding is not possible	8. Data hiding is possible.
9. Example: PASCAL, C....	9. Example: java,c++....etc.

Advantages of oops:

1. Software complexity can be managed and it is possible to multiple instances.
2. The object oriented principle is used to protect the data.
3. We can eliminate the redundant code by using object oriented programming concept.
4. We can extend the use of classes with the concept called inheritance.
5. By using oops we can build programs from standard working modules rather than writing the code from beginning.

Applications:

1. In real time environments
2. Simulation and modeling
3. App developments
4. Gamings
5. Artificial intelligence
6. Neural networks
7. Object oriented data bases
8. Decision supporting & office automation system
9. Expert systems
10. Parallel computing...etc.

Object oriented programming concepts

An object means a real world entity such as car , bike , chair ,fan....etc. an object oriented programming is a methodology to design a program using classes and objects by providing the following features.

1. Object
2. Classes
3. Inheritance
4. Polymorphism
5. Encapsulation
6. Abstraction

1. OBJECT:

Object is an instance of a class which has a state and behaviour , which can be physically and logically present.

2. CLASSES:

Collection of objects is called as class & it is a logical entity.

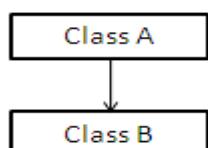
3. INHERITANCE:

When an object acquires all the properties & behaviour of the parent object that is known as inheritance.

- Inheritance is used for code reusability.
- Inheritance is also used to achieve runtime polymorphism.
- It enables the refinement & specialization of existing class.
- The class that inherits set of state & behaviour is called as sub class or child class.
- The class from which it inherits is called base class or super class.
- Types of inheritances

- i. Single inheritance
- ii. Multiple inheritance(this can be achieved in java through interfaces)
- iii. Multilevel inheritance
- iv. Hierarchical inheritance
- v. Hybrid inheritance

i. SINGLE INHERITANCE: When a class extends other class (only one class) that type of inheritance is known as single inheritance.

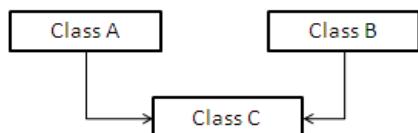


Class A is parent class or super class

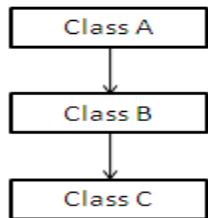
Class B is child class or sub class which can acquire the properties of class A

i.MULTIPLE INHERITANCE: It is nothing but one class extending more than one class.

Most of the object oriented programming languages like c++ , java , c# Does not support multiple inheritance (c++ supports multiple inheritance the concept of multiple inheritance achieved in java by using interfaces).

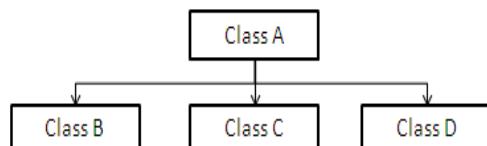


iii. MULTILEVEL INHERITANCE: In multilevel inheritance a derived class will be inheriting a parent class as well as the derived class acts as a parent class to another class.



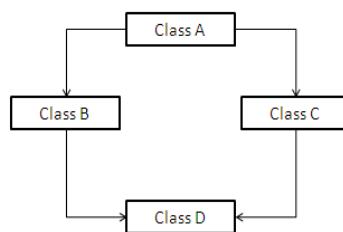
Class B is child for class A and class C is child for class B.

iv. HIERARCHIAL INHERITANCE: In hierachial inheritance one class will be inherited by more than one child class.



Class A is inherited by class B and class C.

vi. HYBRID INHERITANCE: hybrid inheritance is the combination of single and multiple inheritances.



Note: java does not support multiple & hybrid inheritances directly but supports through interfaces.

4. POLYMORPHISM:

When one task is performed in different ways then it is known as polymorphism.

- Polymorphism is that which is used to reduce the no of functions to be remember.
- Polymorphism means one method with multiple implementations for a certain class of action and this information is decided at runtime depending upon the situation (i.e, data type of the object) i.e, different methods react differently on different objects.
- Polymorphism can be of two types
 - o Static(overloading)
 - o Dynamic(overriding)

5. ENCAPSULATION:

Binding or wrapping up of code and data together into a single unit is called as encapsulation.

6. ABSTRACTION:

Hiding internal details & showing the functionality is known as abstraction.



Author	: James Gosling
Vendor	: Sun Microsystems (from 2010 oracle corporation)
Type	: open source
Initial name	: Oak Language
Project name	: Green Project
Present name	: java
First release	: 1995
Initial version	: JDK1.0
Extensions	: .java, .class, .jar
Operating systems	:Multiple Operating systems
Implementation languages	: c, c++
Current versions	: java SE8 (released in 2014)
Objectives	: to develop web applications
Moto	: WORA (Write Once Run Anywhere)

History of java:

- Java was developed by **James gosling** and his team called green team.
- James gosling and his team initiated the java language project in june 1991 who were sun Microsoft engineers.
- Originally it was designed for small embedded systems in electronic applications like setup boxes.
- Initially it was called as **Green talk** by james gosling after that named as Oak.
- In 1995 Oak was renamed as java as already a trade mark by Oak technology was in use.
- Java is an island of Indonesia where the first coffee was produced.
- The first version of java JDK 1.0 was released in January 23, 1996.

Java version history:

Versions	-	year
JDK α & β	-	1995
JDK 1.0	-	23, January 1996
JDK 1.1	-	19, February 1997
J2SE 1.2	-	8, December 1998
J2SE 1.3	-	8, may 2000
J2SE 1.4	-	6, February 2002
J2SE 1.5	-	30, September 2004
Java SE 6	-	11 December 2006
Java SE 7	-	20 july 2011
Java SE 8	-	18 march 2014

Differences between c and java

C	Java
1. In C language we use #include statements to include particular header files & it will be evaluated by the processor as the part of compilation. When preprocessor encounters the #include statement. it will pick up the specified header files & load its pointer to the memory. Therefore #include statements will perform static memory.	1. In java we will use input statements to include predefined library
2. Memory management system is not so good because in C applications developer has to take the responsibility of to create the memory & to destroy the memory.	2. Memory management has been increased where the java developers take the responsibility only to create objects but not for destroying the objects.
3. It is not possible to declare a variable at random locations in C file.	3. Variables can be declared at any locations inside the java class.
4. Code reusability is less.	4. Code reusability is more with the concept called inheritance.
5. C is a process based technology which will reduce the performance of application. Process is heavy –weight.	5. Java is a thread based technology which will improve the performance of application. Thread is a light-weight.
6. It is possible to declare global variables.	6. It is not possible to declare global variables but it is possible to declare class level variables (or) variables inside a class.

7. C is a platform dependent i.e., if we compile any C program then the C compiler may generate .exe files which is platform dependent i.e., it requires the same OS to execute.	7. Java is a platform independent If we compile any program on in any OS then java compile .class with byte code & which is not directly executable code which is also called as intermediate code to execute .class file we require JVM which will convert neutral byte code to the logical OS understandable code. In JVM,JIT(just in time) compiler takes the responsibility to convert the native component representation, therefore java is a platform independent but JVM is platform dependent.
8. In C technology the size of primitive data types may vary from system to system.	8. In java memory allocations of primitive data type is fixed.
9. Pointers are allowed in C	9. Pointers are not allowed.
10. C is comparative language.	10. Java is both comparative and interpreted.
11. In C the default scope is public.	11. In java the default scope is private.

Java features/buzz words

1. Simple
2. Object oriented
3. Platform independent
4. Architecture neutral
5. Portable
6. Robust
7. Secure
8. Dynamic
9. Interpretive
10. Multithreaded
11. High performance
12. Distributive

1. Simple:

- Java programming language is simple because
- i. It is syntactically similar to c and c++
 - i. Complex features of c and c++ are simplified and totally eliminated in java.
 - i. At compile time and execution time in java is less.

2. Object oriented:

Java is pure object oriented programming language which follows concepts of oops

- i.Object
- i.Class
- i.Inheritance
- v.Abstraction
- v.Encapsulation
- i.Polymorphism

In java the data is represented in terms of objects.

In java by using object only we can access methods and attributes

Class ex

```
{  
    Variables  
    Methods  
    Ex obj=new ex();  
}
```

3. Platform independent:

- Java is a platform independent language. A platform is a hardware or software environment in which the program runs.
- Platform independency nature can be of two types Software based and hard ware based
- Java is software base independent in nature i.e., java code can be run on multiple platforms like windows, linux , ios Etc.
- When we compile a java program on one operating system we get byte code (.class files) of which is platform independent in nature i.e this byte is generated in one operating system can be executed in any other operating system successfully.
- Java is a platform independent in nature because of availability of byte code but JVM is platform dependent in nature.

4. Portable:

Java programs compile on one operating system and can be executed in any other operating system.

5. Architecture neutral:

Properties of one processor to properties of another processor.

6. Robust:

Any technology which has the concept of exceptional handling & memory allocation is known as robust Java supports the concepts of exceptional handling and allocation & deallocation of memory at runtime.

7. Secure:

- Java programming language is more secure than other languages.
- Java provides the implicit security in JVM called as security manager.
- To provide explicit security in java application in predefine.
- To provide security for web applications java has provided web security and JAAS(Java Authentication and Authorization Security) for web application.

8. Dynamic:

Java follows dynamic memory allocation i.e., allocation of memory at runtime and the concept of dynamic loading to perform operations for which java is called as dynamic in nature.

9. Interpretive:

Java technology is both compilative and interpretive where interpreter is a part of JVM.

Byte code can be interpreted by a platform using JVM.

10. Multithreaded:

Java multi threading features makes possible of writing programmes which can do multiple group of simultaneously utilization of same resources to be executed by multiple program at same time. A thread is a light weight process or a small task in a large programme.

11. High performance:

Because of availability of JIT(Just In Time) compiler.

12. Distributive:

The protocols like HTTP & FTP are developed in java. So, internet programmers can call functions of these protocols and can get access from any remote machine.

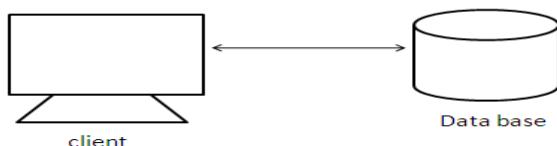
PARTS OF JAVA PROGRAMMING LANGUAGE

According to sun Microsystems the languages are categorized into following

1. J2SE: it is called java 2 standard edition which is used to develop stand alone applications

Ex: notepad, wordpad.... Etc

A standard application is which do not need client server architecture or any internet connection. This kind of application can be executed on local systems

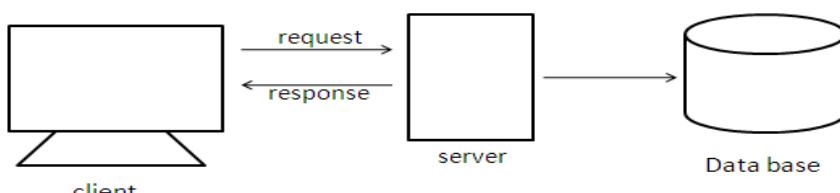


2. J2EE: It is java 2 enterprise edition which is used to develop web based applications.

Ex: railway reservations, net banking, gmail.... Etc

A web application is that which needs client server architecture and internet connection to run the application .

This can be launch by using http(hyper text transfer protocol)



3. J2ME: it is a java 2 micro edition which is used to develop application that run on mobile devices

Ex: java games

Difference between path and classpath in Java

Path

Path variable is set for providing path for all Java tools like java, javac, javap, javah, jar, appletviewer. In Java to run any program we use **java** tool and for compile Java code use **javac** tool. All these tools are available in bin folder so we set path upto bin folder.

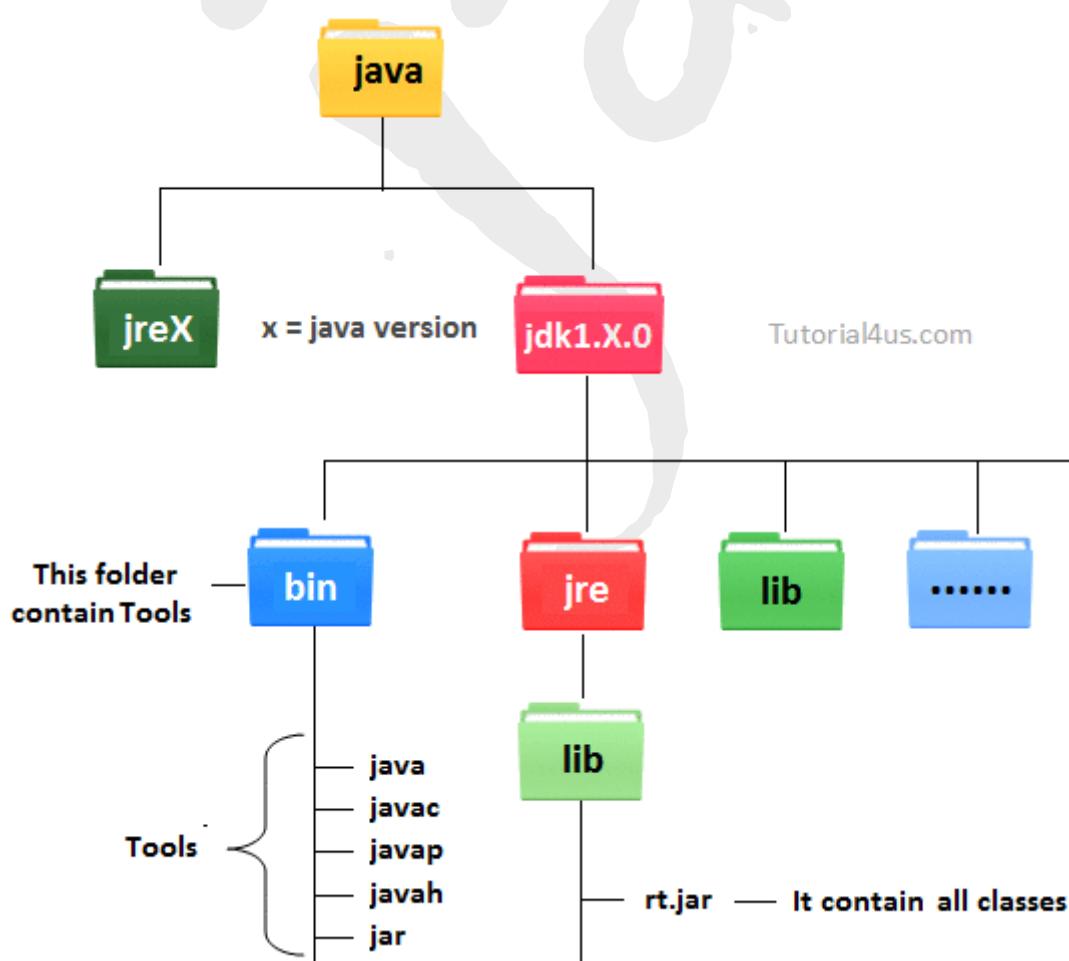
classpath

classpath variable is set for providing path of all Java classes which is used in our application. All classes are available in **lib/rt.jar** so we set classpath upto lib/rt.jar.

Difference between path and classPath

path	classpath
path variable is set for providing path for all java tools like java, javac, javap, javah, jar, appletviewer	classpath variable is set for provide path of all java classes which is used in our application.

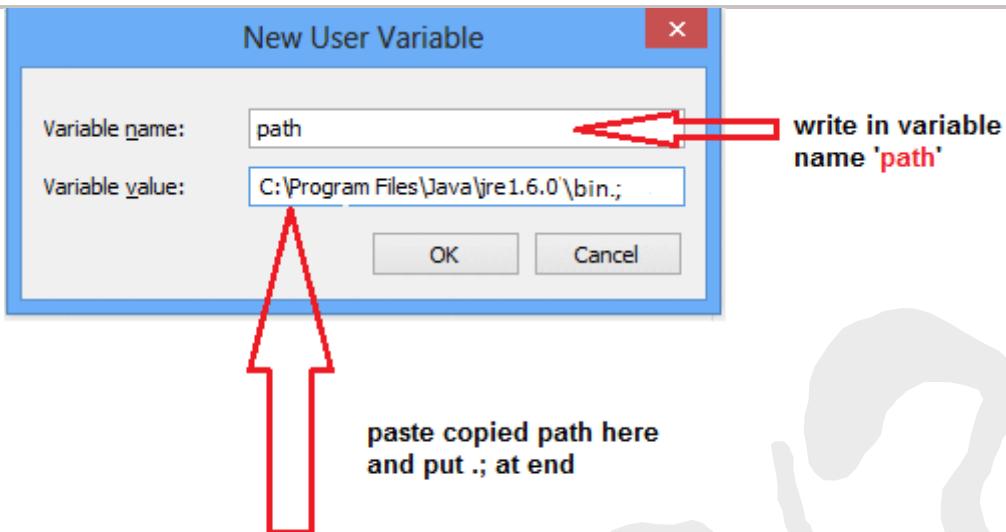
JDK Folder Hierarchy



Path variable is set for use all the tools like java, javac, javap, javah, jar, appletviewer etc.

Example

"C:\Program Files\Java\jdk1.6.0\bin"



C:\Program Files\Java\jre1.6.0\bin; ;

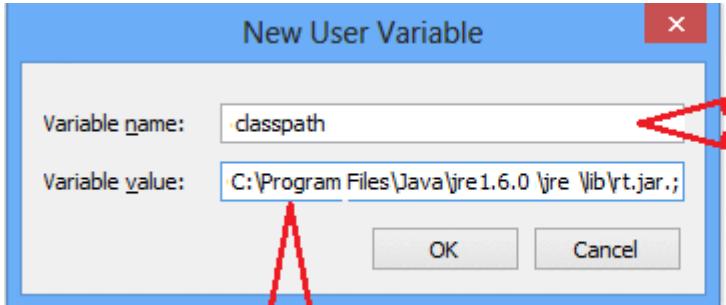
All the tools are present in bin folder so we set path upto bin folder.

Classpath variable is used to set the path for all classes which is used in our program so we set classpath upto rt.jar. in rt.jar file all the .class files are present. When we decompressed rt.jar file we get all .class files.

Example

"C:\Program Files\Java\jre1.6.0\jre\lib\rt.jar"

In above rt.jar is a jar file where all the .class files are present so we set the classpath upto rt.jar.



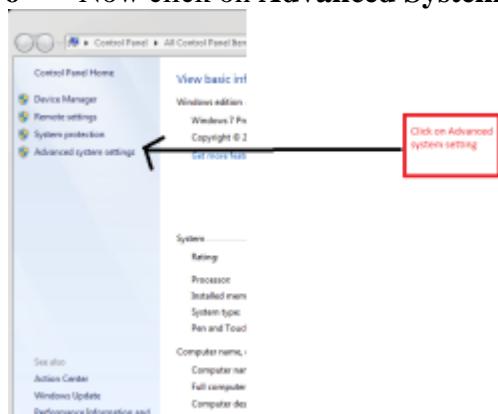
write in variable
name 'classpath'

paste copied path here
and put .; at end

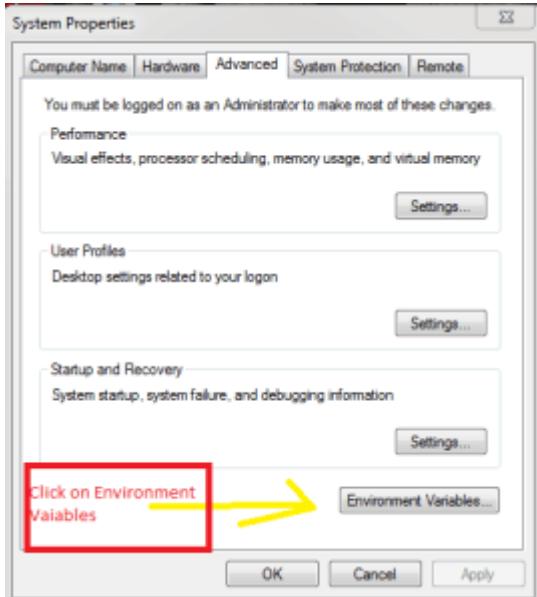
C:\Program Files\java\jre1.6.0\jre\lib\rt.jar.;

HOW TO SET PATH AND CLASSPATH VARIABLE FOR JAVA IN WINDOWS 7

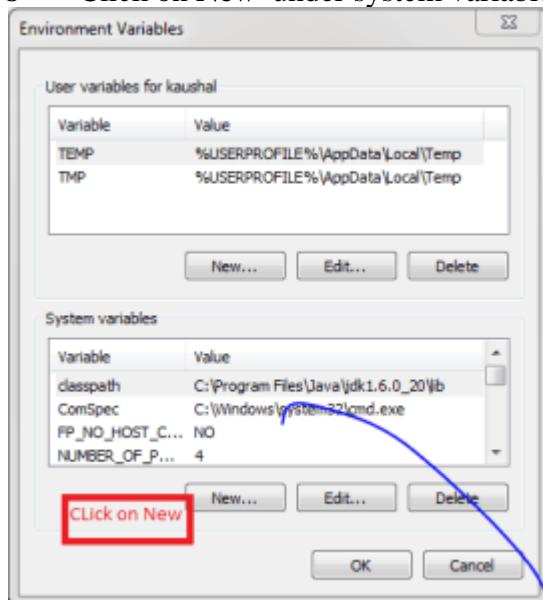
- 1- Download JDK 6 or 7
- 2- Install JDK and by Default Java installs at this place **C:\Program Files\Java**
- 3- **Java folder** contain two folder of name **jdk1.6.0_20** and **jre6**
- 4- Now we will set the classpath and path for java in our windows 7 machine
- 5- Right Click on **My Computer**
- 6- Now click on **Advanced System Setting.**



- 7- Click on **Environment Variables** on next window that appears after clicking **Advanced System Setting.**



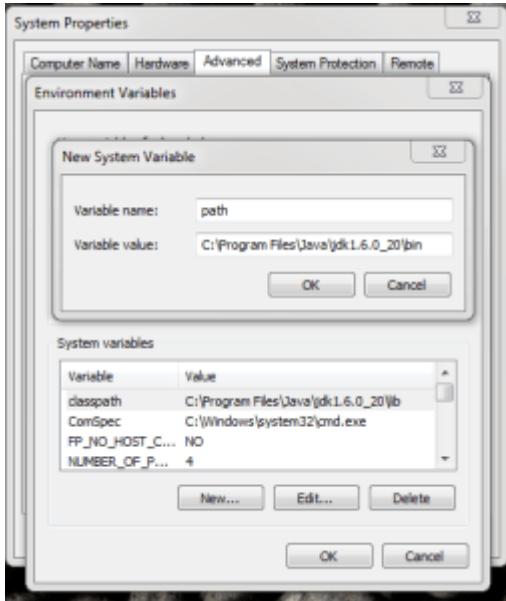
8- Click on New under system variable on Environment Variables Window



9- A new window for New System Variable will open

10- Before Variable Name write path

11- Before variable value type C:Program FilesJavajdk1.6.0_20bin

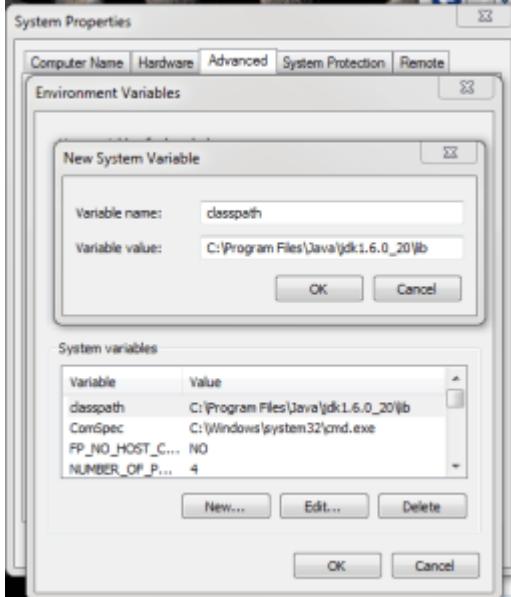


12- Click on **Ok**

13- Now to set classpath again click on **New** and again **New System Variables** will appear

14- Enter

Variable name = classpath and Variable Value = C:/Program Files/Javajdk1.6.0_20/lib



15- Now click **Ok** and again click **Ok** on **Environment Variables**

16- Finally we have set path and classpath and now start you java coding

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following –

- **Notepad** – On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.

- **Netbeans** – A Java IDE that is open-source and free which can be downloaded from <https://www.netbeans.org/index.html>.
- **Eclipse** – A Java IDE developed by the eclipse open-source community and can be downloaded from <https://www.eclipse.org>

JAVA PROGRAMMING STRUCTURE

1. Documentation section
2. Package section
3. Import section
4. Interface section
5. Class section
6. Main method section

Documentation section(or)comment section:

As part of the java application development before starting the application implementation we have to provide meta data about the application implementation which may include data like objective of implementation, author name, object name , start date etc. they provide meta data in java applications . java technology has provide separate features called comments.

These are of 3 types

- i.Single line comment
- i.Multiple line comment
- i.Document comment

- i.Single line comment: it will describes the meta data in single line. It is represented by //.
- i.Multi line comment: it is used to about meta data more than one line. It is represented as /*-----*/
- i.Document comment: it will describe the meta data in more than one page this kind of comment is used to prepare API kind of documents. To represent meta data in java applications. JDK 5.0 version has provided a feature called annotations.

Package section:

Package is a collection of classes & interfaces which will provide the following advantages

- i.Modularity
- i.Abstraction
- i.Sharability
- v.Reusability

In java applications the package declarations should be 1st statement in java file after comment section.

Import section:

The main purpose of import statement is to import the classes & interfaces from a particular class and interface from a particular package.

To use import statement in java we have to use the following syntax:

Import package.name.*;

Example: import java.io.*;

In the above statement all the specified packages are imported into java file.

Syntax: import packagename.entity_name;

Example: import java.io.BufferedReader;

In the above statement it will specify package in io

Note: in a java file atmost one package declaration statement can be included or written but more than one import statements can be written.

Interface (or) class:

A class or interface section deals with classes like abstract class or class or interfaces that can be used in the java program.

Main method class:

Class is a keyword used for developing user defined data type and every data type and every java program must start with a class and this is the class where the main method is written.

The following is the structure of a class

Class classname

{

 Data members;

 Userdefined methods;

 Static public void main(string args[])

{

 Block of statements

}

}

Example:

Class addition

{

 Int a=10;

 Static int b=20;

 Void add();

 Static void display();

 Public static void main(string args[])

```
{  
    System.out.println("hello");  
}  
}
```

Classname represents a java valid variable name and treated as classname and every classname in java is treated as user defined data type.

Whenever we define a class a memory will not be allocated for a data members & methods and this is can be used for creating objects.

Data members represents either static or instance and will be selected based on the class.

User defined methods represents either static or instance which are developed by java programmers to perform specific operations according to the user requirements and these type of methods are called as business logic methods.

Public static void main

Main each and every java program has to start its execution from main. hence this method is called as program driver.

This method is search by JVM as a starting point for the application to run.

Void since main method is not returning any value the return type of the main must be void .

Public is an access specifier in java which defines the method can be accessed by anyone in the class

String is a class

Args[] is the name of the string array which takes string values from the command prompt. The string array name can be anything .

Ex: public static void main(string brgs[])

Static keyword: since java is purely object oriented programming language. Without creating an object to a class it is not possible to access methods and members of a class.

The main is also a class it is not possible to access methods and members of a class, as main is a method inorder to access this method we have to create an object for the class where the main method is present.

Inorder to create the object, the program has to be started , inorder to start the program the execution has to begin from the main method which is an ambiguity situation, inorder to avoid the keyword called static is used i.e., as program execution has to start from main method we need to call the main method without creating object.

Since we need to call main() without using object. we should declare main() as static.

JVM calls main() by using following syntax at run time of the program

Classname.main()

Note:

1. JVM is a program written by java development team and main is a method written by programmers. Since, main() should be available to JVM . it should be declared as public .if we do not declare main() as public then it does not make itself to available to JVM , and JVM cannot execute it. When if we do not declare main as public the default access specifier is private.

Ex:

```
class A
{
    static void main(String args[])
    {
        System.out.println("hello");
    }
}
```

When we executes the above program . the JVM will gives following error

Main method not public

JVM always looks for main() with string type array as parameter otherwise JVM cannot recognize the main() inside the class and gives an exception stating that “no such method error:main”

Ex:

Class A

```
{public static void main()
{
    System.out.println("hello");
}}
```

Error: no such method error: main

System.out.println : in java print() or println() methods are used to print something onto the screen where print is used to print the statements on to the console or monitor and the control will be in the same line

Ex:

Class A

```
{public static void main(String args[])
{
    System.out.print("hello");
    System.out.print("hi");
}}
```

In println() statement the print is used to print the statements on the console or monitor and ln represents go to the new line and print the next statements.

Ex:

Class A

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

}

System.out. a method has to be called by using objects in the following way

Objectname.methodname()

To call print() we need to create an object to print stream class then it has to be called as “objectname.print()”

An alternate way to print stream class is System.out

System is a final class which is present in java.lang package which provides facilities like standard input, standard output loading of files and libraries etc.

Out is a static variable in system class and called as field in system and is of type print.stream whose access specifiers are public final

Ex:

Class A

```
{  
    public static void main(String args[]){  
        System.out.println("hello");  
        System.out.print("hi");  
    }  
}
```

A class contains 5 elements

Example:

Class A

```
{  
    Variables  
    Methods  
    Constructors  
    Instance block  
    Static block  
}
```

JAVA TOKENS

A token is a smallest individual element in a java program which have identified by the compiler.

Java supports the following number of tokens

1. Keywords
2. Identifiers
3. Variables
4. Constants
5. Operators
6. Special characters

1. keywords : keywords are the fixed meaning words or reserved words.

Java consists of 58 keywords

- Keywords for modifiers

Public
private
protected
static
abstract
final
native
volatile
synchronized
transient
strict

- keywords for flow control

if
else
switch
break
case
default
continue
while
do
for

- keywords for objects

new
instanceof
super
this

- keywords for class

import
interface
extends
implements
package
import

- keywords for primitive data type

byte
short
int
long
float

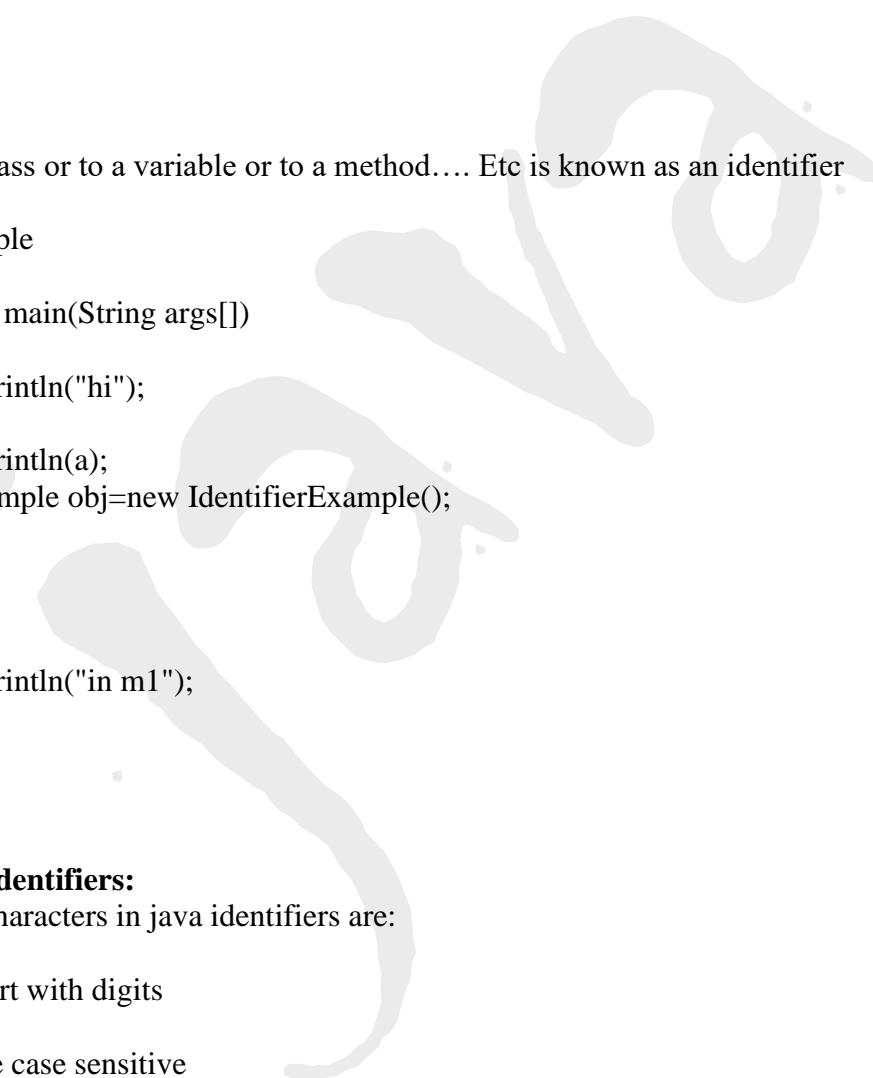
- double
- char
- Boolean
- keywords for methods
- void
- return
- keywords for reserved literals
- true
- false
- NULL
- unused keywords
- goto
- const
- 1.5 version keywords
- Assert
- enum

2. Identifiers:

A name given to a class or to a variable or to a method.... Etc is known as an identifier

Ex:

```
class IdentifierExample
{
    public static void main(String args[])
    {
        System.out.println("hi");
        int a=10;
        System.out.println(a);
        IdentifierExample obj=new IdentifierExample();
        obj.m1();
    }
    void m1()
    {
        System.out.println("in m1");
    }
}
```



hi
10
in m1

Rules to construct identifiers:

i.They only allowed characters in java identifiers are:

a-z, A-Z, 0-9, _, \$

i.Identifiers cannot start with digits

i.e., 9adi, _9adi, etc.

i.Identifiers in java are case sensitive

Ex:

```
class IdentifierExample
{
    public static void main(String args[])
    {
        System.out.println("hi");
        int var=10;
        int Var=20;
        int VAR=30;
        System.out.println(var);
        System.out.println(Var);
```

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

hi
10
20
30
```

v.In java there is no limit for the length (no of characters) of an identifiers.

Ex:

```
class IdentifierExample
{
    public static void main(String args[])
    {
        System.out.println("hi");
        int gHJGHJGGgjkFHkhkfhljsljsuwoulah=10;
        System.out.println(gHJGHJGGgjkFHkhkfhljsljsuwoulah);
    }
}

hi
10
```

v.Keywords cannot be used as an identifiers.

Ex:

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

```
IdentifierExample.java:5: error: not a statement
    int if=10;
           ^
IdentifierExample.java:5: error: ';' expected
    int ^
IdentifierExample.java:5: error: '<' expected
    int ^
IdentifierExample.java:5: error: '>' expected
    int ^
IdentifierExample.java:6: error: illegal start of expression
        System.out.println(if);
                           ^
IdentifierExample.java:6: error: ';' expected
        System.out.println(if);
```

vi. In java programming predefined classnames & predefined interface names can be used as an identifiers.
vii. Duplication of names is not possible.

3. Variables:

A variable is the name given to the place where the data is stored .

In java we have two types of variables.

1. primitive variables
2. reference variables

1. primitive variables:

primitive variables are those which are used to represent primitive values

ex: int a=100;
float a=10.5;
char a='a';

Types of primitive variables

- a. local variables
- b. instance variables
- c. static variables

a. local variable:

in java local variables are the variables which are declared inside a method or block or constructor.

- local variables are used as temporary requirement of a program.
- local variables are known as temporary variable or stack variables or automatic variables.
- local variables are called as stack variables because they are stored in stack area.
- local variables will be created while executing the block in which we declare it and ,destroy it once the control comes out of the block.
- Local variables scope is same as the block in which it is declared i.e, we cannot access local variable outside the block in which it is declared.

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        System.out.println("in main");
        LocalVariable obj=new LocalVariable();
        obj.m1();
        obj.m2();
    }
    void m1()
    {
        int a=10;
        System.out.println("a="+a);
    }
    void m2()
    {
        System.out.println("a="+a);
    }
}
```

Output:

```
LocalVariable.java:17: error: cannot find symbol
    System.out.println("a="+a);
                           ^
      symbol:  variable a
      location: class LocalVariable
1 error
```

- when dealing with local variables it is recommended to perform initialization to the local variable before it is used in the program otherwise a compilation error will be generated stating that variable is not initialized.

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        System.out.println("im main");
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        int a;
```

```
        System.out.println("a="+a);
    }
}
```

Output:

```
LocalVariable.java:12: error: variable a might not have been initialized
        System.out.println("a="+a);
                           ^
1 error
```

- There is no concept of garbage value in java.

Even though JVM provides default values for other variables but not for local variables.

- It is the responsibility of the programmer to provide default value at the declaration part

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        System.out.println("in main");
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        int a=0;
        System.out.println("a="+a);
    }
}
```

Output:

```
in main
a=0
```

- We cannot use any access specifiers for local variables.

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        public int a=0;
        private int b=0;
        protected int c=0;
        System.out.println("a="+a);
    }
}
```

output:

```
D:\z>javac LocalVariable.java
LocalVariable.java:10: error: illegal start of expression
    public int a=0;
                           ^
LocalVariable.java:11: error: illegal start of expression
    private int b=0;
                           ^
LocalVariable.java:12: error: illegal start of expression
    protected int c=0;
                           ^
3 errors
```

We cannot use static keyword for local variables

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        Static int a=0;
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>javac LocalVariable.java
LocalVariable.java:10: error: not a statement
          ^Static int a=0;
LocalVariable.java:10: error: ';' expected
          ^Static int a=0;

2 errors
```

- We can use final keyword for local variables

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        final int a=10;
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>java LocalVariable
a=10
```

- When a local variable declared as final the value is cannot be modified

Ex:

```
class LocalVariable
{
    public static void main(String args[])
    {
        LocalVariable obj=new LocalVariable();
        obj.m1();
    }
    void m1()
    {
        final int a=10;
        a=a+10;
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>javac LocalVariable.java
LocalVariable.java:11: error: cannot assign a value to final variable a
        a=a+10;
               ^
1 error
```

b. Instance variables:

- Instance variables are the variables which are declared within the class directly and outside of any methods, block or constructor.

Ex:

```
class InstaceExample
{
    int a=10;
    public static void main(String args[])
    {
        System.out.println("in main");
        InstaceExample obj=new InstaceExample();
    }
}
```

Output:

```
D:\z>java InstaceExample
in main
```

- These cannot be accessed in static area directly but has to be access only through objects

Ex:

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

Output:

```
D:\z>javac InstaceExample.java
InstaceExample.java:7: error: non-static variable a cannot be referenced from a
static context
        System.out.println(a);
               ^
1 error
```

Error: non static variable 'a' cannot be reffered from static

- Inorder to access instance member in static area we have to create object access only only through object.

Ex:

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

Output:

```
D:\z>java InstaceExample
obj.a=0
```

- For instance members it is not mandatory to provide initialization. If initializations are not done. JVM takes the responsibility of initializing the instances the instance variable with default values

Ex:

```
class InstaceExample
{
    int a;
    public static void main(String args[])
    {
        InstaceExample obj=new InstaceExample();
        System.out.println("obj.a="+obj.a);
    }
}
```

Output:

```
D:\z>java InstaceExample
obj.a=0
```

- Value of instance variable varies from object to object

i.e the no of objects created for each object a separate copy of instance variable are created.

Ex:

```
class InstaceExample
{
    int a=10;
    public static void main(String args[])
    {
        InstaceExample obj1=new InstaceExample();
        InstaceExample obj2=new InstaceExample();
        System.out.println("obj1.a"+obj1.a);
        System.out.println("obj2.a"+obj2.a);
    }
}
```

Output:

```
D:\z>java InstaceExample
obj1.a10
obj2.a10
```

- The value of instance variable in one object will not effect value of instance variable of another object .

class InstaceExample

```
{
    int a=10;
    public static void main(String args[])
    {
        InstaceExample obj1=new InstaceExample();
        InstaceExample obj2=new InstaceExample();
        System.out.println("obj1.a"+obj1.a);
        obj1.a=50;
        System.out.println("after obj1.a"+obj1.a);
        System.out.println("after obj2.a"+obj2.a);
    }
}
```

Output:

```
D:\z>java InstaceExample
obj1.a10
after obj1.a50
after obj2.a10
```

- Instance variable will be stored as the part of the object
- Instance variables scope is object scope i.e, instance variables can be created at object creation and destroyed at object destruction.

- Instance variables are also known as object level variables or attributes.

Ex:

```
class InstaceExample
{
    int a=10;
    public static void main(String args[])
    {
        InstaceExample obj=new InstaceExample();
        obj.m2();
    }
    void m2()
    {
        System.out.println("in m2");
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>java InstaceExample
in m2
a=10
```

c. Static variables:

- If a instance variable declared with static keyword then that is called as static variables
- It should be declared with in the class directly with in static modifier
- For static variables it is not mandatory to provide initialization, JVM takes the responsibility of providing default value.

Ex:

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

Output:

```
D:\z>java StaticExample
in main
a=0
```

- Static variables can be accessed directly in static area without creation of any object.

Ex:

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

Output:

```
D:\z>java StaticExample  
a=10  
obj.a=10  
classname.a=10
```

- A static variable can also be accessed with the class name where it has been declared.
- Static variables are called as class level variables or fields.
- Static variables are created at the time of class loading . hence, static variables can be accessed in instance area.
- Incase of static variable for every object there will be a single copy of static variable

Ex:

```
class StaticExample  
{  
    static int a=10;  
  
    public static void main(String args[])  
    {  
        StaticExample obj1=new StaticExample();  
  
        StaticExample obj2=new StaticExample();  
  
        System.out.println("obj1.a="+obj1.a);  
  
        System.out.println("obj2.a="+obj2.a);  
  
        obj1.a=50;  
  
        System.out.println("after obj1.a="+obj1.a);  
  
        System.out.println("after obj2.a="+obj2.a);  
    }  
}
```

Output:

```
D:\z>java StaticExample  
obj1.a=10  
obj2.a=10  
after obj1.a=50  
after obj2.a=50
```

- Static variables are created at class loading and is determined at class destruction
- Static variables can be accessed directly in static and instance area
- Instance variables can be accessed directly in instance areas but in static it should be accessed through objects .
- Static variables are class level variables and instance variables are object level variables.
- Static variables, single copy is maintained for multiple objects for instance variables each & every object has its own copy of the instance variable.

2. reference variables:

Reference variable is that which is used to refer/represent objects of a class

ex:

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

```
{  
    A a=new A();  
}  
}
```

Here a is the reference variable

4. Constants:

A value assigned to a variable is called as constants

Ex: a=10

Here a is a variable and 10 is a constant

Constants are of 4 types

1. Integer
2. Character constant
3. String constant
4. Real constants

Java supports the following types of constants

1. Numerical constants/literals
2. Character literals
3. Boolean literals

1. Numerical or integer literals:

Numerical/integer literals are those which have integral values

These are divided into three types:

- i.Decimal integral literals
- ii.Octal integral literals
- iii.Hexa decimal integral literals

i. Decimal integral literals:

These are anything with the combination of 0-9

Ex: int x=2;

ii. Octal integral literals:

Allowed anything with the combination of 0-7 with leading '0'

Ex: int a=02;

iii. Hexa decimal integral literals:

Allowed anything with the combination of 0-9,A-F,a-f with leading '0X'

Ex: int a=0X2;

```
class IntegralLiterals
```

```
{  
    public static void main(String args[]){  
        {  
            int a=10,b=010,c=0X10;  
            System.out.println("a="+a);  
            System.out.println("b="+b);  
            System.out.println("c="+c);  
        }  
    }  
}
```

```
}
```

Output:

```
D:\z>java IntegralLiterals
a=10
b=8
c=16
```

Floating point literals:

- Which consists of atleast 1 decimal value

Ex: float a=1.23f;

There are 2 types

Fractional form and exponent form

- Fractional form:

These are those which are represented with one decimal point value

Ex:

```
class IntegralLiterals
{
    public static void main(String args[])
    {
        Float d=1.2e3f;
        System.out.println("d="+d);
    }
}
```

Output:

```
D:\z>java IntegralLiterals
d=1200.0
```

- Exponent form:

These are represented in exponential form

Ex: 1.2e

- By default the decimal point represents double data type. Therefore if you want to assign the floating point values to a variable. It is responsibility of the programmer to attach or it to its suffix f or F.

Ex:

```
class FloatingPoint
{
    public static void main(String args[])
    {
        System.out.println("in main");
        float f=1.25;
        System.out.println("f="+f);
    }
}
```

Output:

```
D:\z>javac FloatingPoint.java
FloatingPoint.java:6: error: possible loss of precision
      float f=1.^25;
                           ^
required: float
found:     double
1 error
```

- When we compile above program we will get an error stating that “possible loss of precession”.
- Inorder to overcome the error attach F or f.

- In order to differentiate floating point as double we have to attach the suffix d or D.

Ex:

```
class FloatingPoint
{
    public static void main(String args[])
    {
        System.out.println("in main");
        float f=1.25f;
        System.out.println("f="+f);
    }
}
```

Output:

```
D:\z>java FloatingPoint
in main
f=1.25
```

- When dealing with floating point literals values can be specified in decimal form other than that it will not work out.

Ex:

```
class Double
{
    public static void main(String args[])
    {
        System.out.println("in main");
        double d=0*1.25;
        System.out.println("d="+d);
    }
}
```

Output:

```
D:\z>java Double
in main
d=0.0
```

- We can assign integral literals either in octal and hexadecimal form directly to floating point decimal.

Ex:

```
class Double
{
    public static void main(String args[])
    {
        System.out.println("in main");
        float a=2;
        float b=05;
        float c=0xfa;
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("c="+c);
    }
}
```

Output:

```
D:\z>java Double
in main
a=2.0
b=5.0
c=250.0
```

Boolean literals:

- These are literals which are of two values true or false.

Ex:

```
class Boolean
```

```
{  
    public static void main(String args[])  
    {  
        System.out.println("in main");  
        boolean b=true;  
        System.out.println("b="+b);  
  
    }  
}
```

Output:

```
D:\z>java Boolean  
in main  
b=true
```

- When dealing with boolaen data types the following declarations are not valid.
- We cannot supply integral values to a Boolean data types variables.

Ex:

```
class Boolean  
{  
    public static void main(String args[])  
    {  
        System.out.println("in main");  
        boolean b=0;  
        System.out.println("b="+b);  
  
    }  
}
```

Output:

```
D:\z>javac Boolean.java  
Boolean.java:6: error: incompatible types  
      boolean b=0;  
                         ^  
      required: boolean  
      found:     int  
1 error
```

- We cannot supply Boolean values interms of string which gives a compilation error.
- Boolean literals are case sensitive

Ex:

```
class Boolean  
{  
    public static void main(String args[])  
    {  
        System.out.println("in main");  
        boolean b=True;  
        System.out.println("b="+b);  
  
    }  
}
```

Output:

```
D:\z>javac Boolean.java  
Boolean.java:6: error: cannot find symbol  
      boolean b=True;  
                         ^  
      symbol:   variable True  
      location: class Boolean  
1 error
```

Ex2:

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

```
{  
    System.out.println("in main");  
    boolean b=true;  
    if(b==false)  
    {  
        System.out.println("statement1");  
    }  
    else  
    {  
        System.out.println("statement2");  
    }  
}  
}
```

Output:

```
D:\z>java Boolean  
in main  
statement2
```

Character literals

- Anything represented in single quotes(') is treated as character which is used to represents Unicode values

Ex:

```
class CharList  
{  
    public static void main(String args[])  
    {  
        char a='a';  
        System.out.println(a);  
    }  
}
```

Output:

```
D:\z>java CharList  
a
```

- If character literals are not represented in single quotes it leads to a compilation error

Ex:

```
char a=a;
```

- In character literal we can supply a maximum unicode value of 65535

Ex:

```
class CharList  
{  
    public static void main(String args[])  
    {  
        char a=65535;  
        System.out.println(a);  
    }  
}
```

Output:

```
D:\z>java CharList  
?
```

String literals:

- Any sequence of characters within double quotes is called as string literals

Ex:

```
class StringList
{
    public static void main(String args[])
    {
        String s="Rani";
        System.out.println(s);
    }
}
```

Output:

```
D:\z>java StringList
Rani
```

Escape sequence characters:

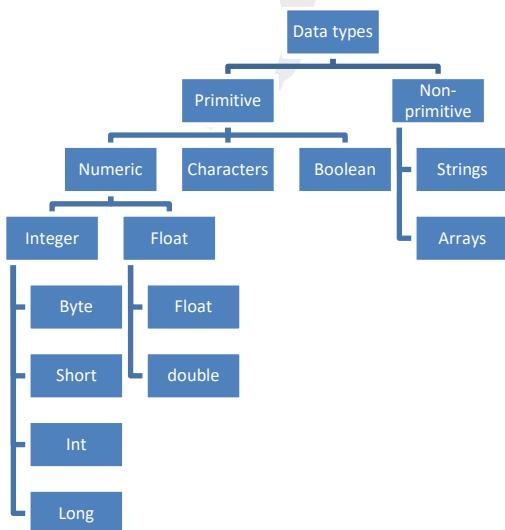
s.no	Escape sequence characters	Meaning
1.	\n	New line
2.	\t	Tab space
3.	\a	bell
4.	\'	To print single quote
5.	\”	To print double quotes

DATA TYPES IN JAVA

Java supports different data types

Data types represents the different values that has to be stored in the variable. These are different types of data types.

1. Primitive
2. Non-primitive



1. PRIMITIVE DATA TYPES:

Primitive data types are the basic data types which are called as primary data types. these data types are 3 types

- i.Numeric
- i.Char
- i.Boolean

1. NUMERIC: which are used to store the numbers these are of two types

- a. Integral
- b. Floating point

a. INTEGRAL DATA TYPE:

These are the data types which are used to represent whole numbers these are of 4 types:

Byte

Short

Int

Long

BYTE:

- In java size the size of the byte is 1 byte or 8 bits

Ex:

```
class Example
{
    public static void main(String args[])
    {
        byte a=-12;
        System.out.println(a);
    }
}
```

Output:

```
D:\z>java Example
-12
```

- Byte is used to store small range value
- Which is used to store the range of -128 to 127
- Therefore, the maximum value is 127 and minimum value is -128

Ex:

```
class Example
{
    public static void main(String args[])
    {
        byte a=127;
        byte b=-128;
        System.out.println("a="+a);
        System.out.println("b="+b);
    }
}
```

Output:

```
D:\z>java Example
a=127
b=-128
```

- The default value supplied for byte is '0',this default value is supplied by JVM for fields of the class (not for local variables)
- Byte data type is used to handle data interms of fields either from file or network.

SHORT DATA TYPE:

- It is an integral data type which is used to store the size of 2 bytes which ranges from -32768 to 32767

Ex:

```
class Example
{
    public static void main(String args[])
    {
        short a=-32767;
        short b=-32768;
        System.out.println("a="+a);
        System.out.println("b="+b);
    }
}
```

Output:

```
D:\z>java Example
a=-32767
b=-32768
```

- When we assign a value to the variable of short data type which is beyond its range. It would not work out.

Ex:

```
class Example
{
    public static void main(String args[])
    {
        short a=32768;
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>javac Example.java
Example.java:5: error: possible loss of precision
      short a=32768;
                           ^
 required: short
 found:     int
1 error
```

- When we compile the above program we get the compilation error stating that “**possible loss of precision**”

- The default value supplied for short is 0, this default value is supplied by JVM for fields of class(not for local variables)

Ex:

```
class Example
{
    short s=325;
    public static void main(String args[])
    {
        Example obj=new Example();
        System.out.println("s="+obj.s);
    }
}
```

Output:

```
D:\z>java Example  
s=325
```

INT DATA TYPE:

- The most commonly used data type in java is int. in java the size of integer is 4 bytes which stores a range of values from -2^{31} to $2^{31} - 1$

Ex:

```
class Example  
{  
    public static void main(String args[])  
    {  
        int a=-2147483648;  
        int b=2147483647;  
        System.out.println("a="+a);  
        System.out.println("b="+b);  
    }  
}
```

Output:

```
D:\z>java Example  
a=-2147483648  
b=2147483647
```

- The integer data type value ranges from -2^{31} to $2^{31} - 1$ beyond its range would not work out.

Ex:

```
class Example  
{  
    public static void main(String args[])  
    {  
        int a=2147483648;  
        System.out.println("a="+a);  
    }  
}
```

Output:

```
D:\z>javac Example.java  
Example.java:5: error: integer number too large: 2147483648  
        int a=2147483648;  
                           ^  
1 error
```

- When we compile the above program we get the compilation error stating that "**integer number is too large**"

- The default value is supplied for int is 0, this default is supplied by JVM for fields of the class(not for local variables)

Ex:

```
class Example  
{  
    int a;  
    public static void main(String args[])  
    {  
        Example obj=new Example();  
        System.out.println("a="+obj.a);  
    }  
}
```

Output:

```
D:\z>java Example  
a=0
```

LONG DATA TYPE:

- It ranges from -2^{63} to $2^{63} - 1$
- It is used to integeral values is of size 8 bytes.
- The default value supplied by JVM for long is **0L**

Ex:

```
class Example  
{  
    public static void main(String args[])  
    {  
        long a=12525;  
        System.out.println("a="+a);  
    }  
}
```

Output:

```
D:\z>java Example  
a=12525
```

- The long value ranges from -2^{63} to $2^{63} - 1$

Ex:

```
class Example  
{  
    public static void main(String args[])  
    {  
        long a=9.223372038e9;  
        System.out.println("a="+a);  
    }  
}
```

Output:

```
D:\z>javac Example.java  
Example.java:5: error: possible loss of precision  
      long a=9.223372038e9;  
  
       required: long  
       found:     double  
1 error
```

- When we compile the above program we get the compilation error stating that "**possible loss of precision**"
- The default value is supplied for long is **0L**, this default is supplied by JVM for fields of the class(not for local variables)

Ex:

```
class Example  
{  
    long a=0L;  
    public static void main(String args[])  
    {  
        Example obj=new Example();  
        System.out.println("a="+obj.a);  
    }  
}
```

Output:

```
D:\z>java Example  
a=0
```

s.no	Data type name	size	Default value
1	Byte	1 byte	0
2	Short	2 bytes	0
3	Int	4 bytes	0
4	long	8 bytes	0L

b. FLOATING POINT:

Inorder to represent the real numbers we using floating point data types.

These are of 2 types

Float

Double

FLOAT:

- The size of floating data type is 4 bytes and range is -3.4e38 to 3.4e38

Ex:

```
class Example
{
    public static void main(String args[])
    {
        float f=1.345;
        float d=-3.4e38;
        System.out.println("f="+f);
        System.out.println("d="+d);
    }
}
```

Output:

```
D:\z>javac Example.java
Example.java:5: error: possible loss of precision
      float f=1.345;
                           ^
required: float
found:   double
Example.java:6: error: possible loss of precision
      float d=-3.4e38;
                           ^
required: float
found:   double
2 errors
```

- The default value is supplied for float is 0.0f, this default is supplied by JVM for fields of the class(not for local variables)

Ex:

```
class Example
{
    float d=0.0f;
    public static void main(String args[])
    {
        Example obj=new Example();
        System.out.println("d="+obj.d);
    }
}
```

Output:

```
D:\z>java Example  
d=0.0
```

- When we compile the above program we get the compilation error stating that float value is too large.
- The maximum precision value we can supply is 7

Ex:

```
class Example  
{  
    float d=0.0f;  
    public static void main(String args[])  
    {  
        float f=1.345f;  
        Example obj=new Example();  
        System.out.println("d="+obj.d);  
    }  
}
```

Output:

```
D:\z>java Example  
d=0.0
```

DOUBLE:

- It ranges from -1.7e308 to 1.7e308

Ex:

```
class Example  
{  
    public static void main(String args[]){  
        double d=-1.7e308d;  
        System.out.println("d="+d);  
    }  
}
```

Output:

```
D:\z>java Example  
d=-1.7E308
```

- If we supply beyond the size the range of double we can get an compilation error stating that floating point is too large.

Ex:

```
class Example  
{  
    public static void main(String args[]){  
        double d=-1.7e309d;  
        System.out.println("d="+d);  
    }  
}
```

Output:

```
D:\z>javac Example.java  
Example.java:5: error: floating point number too large  
        double d=-1.7e309d;  
                           ^  
1 error
```

- The default value is supplied for double is 0.0d, this default is supplied by JVM for fields of the class(not for local variables)

Ex:

```
class Example  
{  
    double d=0.0d;
```

```
public static void main(String args[])
{
    Example obj=new Example();
    System.out.println("obj.d="+obj.d);
}
}
```

Output:

```
D:\z>java Example
obj.d=0.0
```

- The maximum precision value for double is 17

Ex:

```
class Example
{
    public static void main(String args[])
    {
        double d=1.23456789123456789d;
        System.out.println("d="+d);
    }
}
```

Output:

```
D:\z>java Example
d=1.234567891234568
```

2. CHAR:

Character data types are used to store character values, the size of character data types is 2 bytes which ranges from 0 to 65535 which is in terms of Unicode characters i.e., java can be used to store not only ASCII characters but world wide alphabet sets which are called as Unicode characters.

Ex:

```
class Example
{
    public static void main(String args[])
    {
        System.out.println("in main");
        char c='a';
        System.out.println("c="+c);
    }
}
```

Output:

```
D:\z>java Example
in main
c=a
```

- We can also supply integral values to character data type in the range from 0 to 65535

Ex:

```
class Example
{
    public static void main(String args[])
    {
        char c=65555;
        System.out.println("c="+c);
    }
}
```

Output:

```
D:\z>javac Example.java
Example.java:5: error: possible loss of precision
      char c=65555;
                           ^
required: char
found:    int
1 error
```

- The default value for character data type is '\u0000'

Ex:

```
class Example
{
    char c='\u0000';
    public static void main(String args[])
    {
        Example obj=new Example();
        System.out.println("c="+obj.c);
    }
}
```

Output:

```
D:\z>java Example
c=
```

3. BOOLEAN:

- It takes two values as true or false. This data type represents one bit of information but size is not defined
- The range of Boolean is either true or false which is case sensitive

Ex:

```
class Example
{
    public static void main(String args[])
    {
        boolean b=true;
        System.out.println("b="+b);
    }
}
```

Output:

```
D:\z>java Example
b=true
```

-
- In Boolean data type , we get an compilation error stating that cannot find symbol

Ex:

```
class Example
{
    public static void main(String args[])
    {
        boolean b=True;
        System.out.println("b="+b);
    }
}
```

Output:

```
D:\z>javac Example.java
Example.java:5: error: cannot find symbol
          boolean b=True;
                           ^
symbol:  variable True
location: class Example
1 error
```

- The default value supplied by JVM for Boolean data type is false

Ex:

```
class Example
{
    boolean b;
    public static void main(String args[])
    {
        Example obj=new Example();
        System.out.println("b="+obj.b);

    }
}
```

Output:

```
D:\z>java Example
b=false
```

- The default value supplied by JVM for string or object is NULL.

TYPE CASTING:

Type casting is to convert one data type to another data type

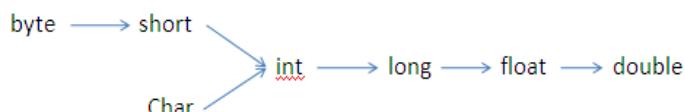
There are 2 types of type casting

1. Implicit type casting

2. Explicit type casting

1. Implicit type casting:

- It is done automatically by the compiler when the user assign smaller data type values to bigger data type variable.
- This type of type casting is also known widening or upcasting
- When we perform implicit type casting there is no loss of information.



Ex:

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

Output:

```
D:\z>java ImplicitEx
b=10
```

2. Explicit type casting:

- It has to be done by the programmers when the user requires to use a bigger data type variable to smaller data type.
- It is also known as narrowing or downcasting.
- By explicit type casting there is a loss of information.

Ex:

```
class ExplicitEx
{
    public static void main(String args[])
    {
        float a=5.2f;
        int b;
        b=(int)a;
        System.out.println("b="+b);
    }
}
```

Output:

```
D:\z>java ExplicitEx
b=5
```

- When we give beyond the size of byte then it will convert it into 2's complement.

Ex:

```
class ExplicitEx
{
    public static void main(String args[])
    {
        byte a;
        int b=130;
        a=(byte)b;
        System.out.println("a="+a);
    }
}
```

Output:

```
D:\z>java ExplicitEx
a=-126
```

Operators

Java operator is a symbol that is used to perform certain kinds of mathematical or logical calculations on data which can be unary and binary or ternary i.e taking 1 or 2 or 3 arguments respectively.

- i.Arithmetic operators
- ii.Relational operators
- iii.Logical operators
- iv. Assignment operators
- v. Increment or decrement operators
- vi. Bitwise operators
- vii. Conditional operators

viii. Instance of operators

i.Arithmetic operators:

These are the operators which are used to perform basic calculations on the given data.

The following are the list of arithmetic operators

- a) Addition
- b) Subtraction
- c) Multiplication
- d) Division
- e) Modulus

Ex: Addition

```
class ArithOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Addition of "+a+"+"+b+": "+(a+b));
    }
}
```

Output:

```
Addition of 10+20:30
```

Ex: Subtraction

```
class ArithOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Subtraction of "+a+"-"+b+": "+(a-b));
    }
}
```

Output:

```
Subtraction of 10-20:-10
```

Ex: Multiplication

```
class ArithOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Multiplication of "+a+"*"+b+": "+(a*b));
    }
}
```

Output:

```
Multiplication of 10*20:200
```

Ex: Division

```
class ArithOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Division of "+a+"/"+b+": "+(a/b));
    }
}
```

Output:

```
Division of 10/20:0
```

ex: Modulus

```
class ArithOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        System.out.println("Modulus of "+a+"%" +b+": "+(a%b));
    }
}
```

output:

```
Modulus of 10%20:10
```

- In java when we are performing arithmetic operations the resultant type is always integer type.

i.Relational operators:

The relational operators are those which are used to test or defines some kind of relationship between two entities and returns a Boolean value.

Java supports the following relational operators.

- Greater than(>)
- Less than(<)
- Greater than or equals to(>=)
- Less than or equals to(<=)
- Equals to(==)
- Not equals to(!=)

Ex: Greater than(>)

```
class RelationalOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        boolean c;
        c=a>b;
        System.out.println(a+">" +b+": "+c);
    }
}
```

```
}
```

Output:

```
10>20:false
```

Ex: Less than(<)

```
class RelationalOp
{
    public static void main(String args[])
    {
        int a=10,b=20;
        boolean c;
        c=a<b;
        System.out.println(a+"<" +b+ ":" +c);
    }
}
```

Output:

```
10<20:true
```

Ex: Greater than or equals to (>=)

```
class RelationalOp
{
    public static void main(String args[])
    {
        int a=10,b=10;
        boolean c;
        c=a>=b;
        System.out.println(a+">=" +b+ ":" +c);
    }
}
```

Output:

```
10>=10:true
```

Ex: Less than or equals to(<=)

```
class RelationalOp
{
```

```
public static void main(String args[])
{
    int a=10,b=9;
    boolean c;
    c=a<=b;
    System.out.println(a+"<="+b+": "+c);
}
```

Output:

```
10<=9:false
```

Ex: Equals to(==)

```
class RelationalOp
{
    public static void main(String args[])
    {
        int a=10,b=9;
        boolean c;
        c=a==b;
        System.out.println(a+"=="+b+": "+c);
    }
}
```

Output:

```
10==9:false
```

Ex: Not equals to(!=)

```
class RelationalOp
{
    public static void main(String args[])
    {
        int a=10,b=9;
        boolean c;
        c=a!=b;
        System.out.println(a+"!=" +b+": "+c);
    }
}
```

```
 }  
 }
```

Output:

```
10!=9:true
```

i. Logical operators:

The logical operators are those which are used to test logical relationship between two conditions which returns Boolean values

The logical operators are:

- a) Logical and(&&)
- b) Logical or(||)
- c) Logical not(!)

Ex: Logical AND(&&)

```
class LogicalOp  
{  
    public static void main(String args[])  
    {  
        int a=10,b=9;  
        boolean c;  
        c=((a<b)&&(b>a));  
        System.out.println(a+">" +b+" && "+b+">" +a+" : "+c);  
    }  
}
```

Output:

```
10>9 && 9>10 : false
```

Ex: Logical or(||)

```
class LogicalOp  
{  
    public static void main(String args[])  
    {  
        int a=10,b=9;  
        boolean c;  
        c=((a>b)|| (b<a));  
        System.out.println(a+">" +b+" || "+b+">" +a+" : "+c);  
    }  
}
```

```
}
```

Output:

```
10>9 || 9>10 : true
```

Ex: Logical not(!)

```
class LogicalOp  
{  
    public static void main(String args[])  
    {  
        int a=10,b=9;  
        boolean c;  
        c=!(a>b);  
        System.out.println("!(" + a + ">" + b + ") : " + c);  
    }  
}
```

Output:

```
!(10>9) : false
```

v. Assignment operators:

These assignment operators are used to assign the value from one entity to another entity

The following are the list of assignment operators

Equals to(=)

Ex:

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

Output:

```
b=10
```

- When assignment operators and arithmetic operators are combine then it is called as shorthand operator.

Ex:

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

```
}
```

```
}
```

Output:

```
a=30
```

- Assignment operators can also be chain

Ex:

```
class AssignmentOp
```

```
{
```

```
    public static void main(String args[])
    {
        int a=10,b;
        int c,d,e;
        c=d=e=70;
        System.out.println("c="+c);
    }
}
```

Output:

```
c =70
```

v. Increment/Decrement operators:

These are unary operators

- java supports a unary increment and decrement operator which is of two types.

a) Increment operator

b) Decrement operator

a) Increment operator: which is represented by `++`, which is incremented by value 1.

Increment operators are of two types.

- **Post increment:** first operand and then operation

Ex:

```
class PostIncOp
```

```
{
```

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

Output:

-

- **Pre increment:**

Ex:

```
class PreIncOp
```

```
{
```

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

Output:

a=11
b=11

b) Decrement operator:

This operator is represented by --, which is decremented by value 1.

These are of 2 types

- Post decrement:

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

Output:

a=9

- Pre decrement:

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

Output:

a=9
b=9

i. Bitwise operators:

The bitwise operators are those which apply on bits which perform in bit levels.

The following are list of bitwise operators

- a) Bitwise AND
- b) Bitwise OR
- c) Bitwise X-OR
- d) Bitwise compliment
- e) Bitwise left shift
- f) Bitwise right shift

a) Bitwise AND:

Bitwise AND operator copies the result based on the following table.

A	B	A& B
0	0	0
0	1	0
1	0	0
1	1	1

Ex:

```
class Bitop
```

```
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=a&b;
        System.out.println(c);
    }
}
```

Output:

12

b) Bitwise OR:

Bitwise OR copies a bit

A	B	A&B
0	0	0
0	1	1
1	0	1
1	1	1

Ex:

```
class Bitor
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=a|b;
        System.out.println(c);
    }
}
```

Output:

61

a) Bitwise X-OR:

Copies the bit if it is present in one operand but not in both.

Ex:

```
class Bitop
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=a^b;
        System.out.println(c);
    }
}
```

Output:

49

b) Bitwise compliment:

It represents as \sim . It is used for flopping of bits i.e converting 0's to 1's and 1's to 0's

Ex:

```
class Bitop
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=~(a>b);
        System.out.println(c);
    }
}
```

```
}
```

Output:

c) Bitwise left shift:

The bitwise left shift operator shifts the value by number of bits specified . it is represented as <<

Ex:

```
class Bitop
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=a<<2;
        System.out.println(c);
    }
}
```

Output:

240

d) Bitwise right shift:

The bitwise right shift operator shifts the value by number of bits specified and it is represented as >>

Ex:

```
class Bitop
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=a>>2;
        System.out.println(c);
    }
}
```

Output:

•

vii. Conditional operators:

It is also known as ternary operator this operator is used to evaluate a Boolean expression

It is represented as (? And :)

Ex:

```
class ConditionalOp
{
    public static void main(String args[])
    {
        int a=60,b=13,c;
        c=(a>b)?20:40;
        System.out.println(c);
    }
}
```

Output:

20

viii. Instance of operator:

This operator is used only for object reference variable. This operator is used to check whether the object is of a particular class or interface or not.

Ex:

```
class InstanceOp
{
    public static void main(String args[])
    {
        InstanceOp obj=new InstanceOp();
        System.out.println(obj instanceof InstanceOp);
    }
}
```

Output:

```
true
```

Instance of operator returns Boolean value.

READING DATA FROM KEYBOARD:

To read the data from the keyboard the following methods are used in java

1. Command line aruement
2. Console
3. Scanner
4. Input stream reader
5. Data input stream...etc.

1. COMMAND LINE ARGUMENT:

- The argument which are passed to the program at the time of running the program is called command line aruement.
- Which is used to customize the behaviour of the main.

Ex:

```
class CommandLineArg
{
    public static void main(String c[])
    {
        System.out.println(c[0]);
        System.out.println(c[1]);
        System.out.println(c[2]);
        System.out.println("using for");
        for(int i=0;i<3;i++)
        {
            System.out.println(c[i]);
        }
    }
}
```

Output:

```
10
20
30
using for
10
20
30
```

- The data enter through command line arguments interm of strings.

Ex:

```
class CommandLineArg
{
    public static void main(String c[])
    {
```

```
{  
    for(int i=0;i<3;i++)  
    {  
        System.out.println(c[i]);  
    }  
}
```

Output:

```
D:\z>java CommandLineArg Rani @ Sahithi  
Rani  
@  
Sahithi
```

- When values are supplied through command line arguments space will act as separator.
- Inorder to read the space in between the values we have to use double quotes(").

Ex:

```
class CommandLineArg  
{  
    public static void main(String c[])  
    {  
        System.out.println(c[0]);  
    }  
}
```

Output:

```
D:\z>java CommandLineArg "Rani @ Sahithi"  
Rani @ Sahithi
```

Ex:

```
class CommandLineArg  
{  
    public static void main(String c[])  
    {  
        System.out.println(c[0]);  
        String a[]={ "Oops", " through ", "JAVA" };  
        c=a;  
        System.out.print(c[0]);  
        System.out.print(c[1]);  
        System.out.println(c[2]);  
        System.out.println(c.length);  
    }  
}
```

Output:

```
D:\z>java CommandLineArg @  
@  
Oops through JAVA  
3
```

- In the above program c.length gives the no of arrays inc[]

2. CONSOLE:

- This is used to get the input from the console which provide method for reading text and passwords.
- The **java.io.console** class is used to read the password but will not display to the user.

Ex:

```
import java.io.*;  
class ConsoleEx  
{  
    public static void main(String args[])  
    {  
        Console c=System.Console();  
    }  
}
```

```
        System.out.println("enter data:");
        String s=c.readLine();
        System.out.println(s);
    }
}
```

Output:

- System class provides a static method console() that returns an instance of console class as follows.
Console c=System.console();
- The method of console classes are
 - 1.public string readLine
 - 2.public char[] read password

3. SCANNER:

- Which is present in **java.util.Scanner**
- In java scanner class breaks the input into tokens at delimiter i.e, white space by default.
- The java scanner class is primarily used to read a string and parse it into primitive types
- The following are the list of methods present in scanner class
 - i. **Public string next()** returns the string from scanner.
 - ii. **Public string nextLine()** it moves the string to the next line and returns string.
 - iii. **Public byte nextByte()** scans the next token as byte.
 - iv. **Public byte nextShort()** scans the next token as short.
 - v. **Public byte nextInt()** scans the next token as int.
 - vi. **Public byte nextFloat()** scans the next token as float.
 - vii. **Public byte nextDouble()** scans the next token as double.

Ex:

```
import java.util.Scanner;

class ScannerEx

{
    public static void main(String args[])
    {
        byte a;
        short b;
        int c;
        float d;
        double e;

        Scanner obj=new Scanner(System.in);

        System.out.print("enter a value:");
        a=obj.nextByte();

        System.out.println("a : "+a);

        System.out.print("enter b value:");
        b=obj.nextShort();
```

```
System.out.println("b : "+b);
System.out.print("enter c value:");
c=obj.nextInt();
System.out.println("c : "+c);
System.out.print("enter d value:");
d=obj.nextFloat();
System.out.println("d : "+d);
System.out.print("enter e value:");
e=obj.nextDouble();
System.out.println("e : "+e);
}
}
```

Output:

```
enter a value:10
a : 10
enter b value:20
b : 20
enter c value:30
c : 30
enter d value:40
d : 40.0
enter e value:50
e : 50.0
```

WRITE A JAVA PROGRAM TO PRINT ADDITION OF TWO NUMBERS

```
import java.util.*;
class Addition
{
    public static void main(String args[])
    {
        int a,b;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a & b values:");
        a=obj.nextInt();
        b=obj.nextInt();
        System.out.print("Addition of "+a+"+" +b+" : "+(a+b));
    }
}
```

Output:

```
enter a & b values:10 20
Addition of 10+20 : 30
```

WRITE A JAVA PROGRAM TO PRINT SUBTRACTION OF TWO NUMBERS

```
import java.util.*;
class Substraction
{
    public static void main(String args[])
    {
        int a,b;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a & b values:");
        a=obj.nextInt();
        b=obj.nextInt();
        System.out.print("Substraction of "+a+"-"+b+" : "+(a-b));
    }
}
```

Output:

```
enter a & b values:10
20
Substraction of 10-20 : -10
```

JAVA CONTROL STATEMENTS:

These statements are used to control the order of execution of the program based on data values and conditional logic

These are of 3 types

1. Selection statements
2. Looping/iterative statements
3. Transfer statements

Control statements

Selection statements

- Simple if
- If else
- Nested if else
- Else if
- switch

Looping statements

- For
- For each
- While
- Do while

Transfer statements

- Break
- Continue
- Return
- catch
- finally

SELECTION STATEMENTS:

Java supports the following selection statements:

1. If
2. If else
3. Nested if else
4. Else if
5. Switch

These statements are used for controlling the flow of program execution based upon the condition. These statements are also called as decision making statements (or) controlling statements.

1. if :

The if statement executes block of statements in the block. If the condition is true.

The condition must be always a Boolean expression.

Syntax:

If(condition)

```
{  
    Statement1;  
}  
Statement2;
```

If condition is true statement 1 & statement2 is executed.

If it is false statement1 is kepted off & statement2 is executed.

Ex:

```
import java.util.*;  
class IfEx  
{  
    public static void main(String args[])  
    {  
        int a;  
        Scanner obj=new Scanner(System.in);  
        System.out.print("enter values:");
```

```
a=obj.nextInt();
if(a==10)
{
    System.out.println("The given value a : 10");
}
System.out.println("program completed");
}
```

Output:

```
enter values:10
The given value a : 10
program completed
```

2. if else:

syntax:

```
If(condition)
```

```
{
    Statement1;
}
```

```
Else
```

```
{
    Statement2;
}
```

```
Statement3;
```

Write a sample java program on if else condition

```
import java.util.*;
class IfElseEx
{
    public static void main(String args[])
    {
        int a,b;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a & b values:");
        a=obj.nextInt();
        b=obj.nextInt();
        if(a>b)
        {
            System.out.println(a+">" +b);
        }
        else
        {
            System.out.println(a+"<" +b);
        }
        System.out.println("program completed");
    }
}
```

Output:

```
enter a & b values:10 20
10<20
program completed
```

Condition should always return a Boolean value

3. nested if:

syntax:

```
If(condition)
```

```
{
    If(condition)
```

```
{  
    Statement1;  
}  
Else  
{  
    Statement2;  
}  
Else  
{  
    Statement3;  
}  
}  
Statement4;
```

Write a sample java program on nested if else condition

```
import java.util.*;  
  
class NestedIfEx  
{  
    public static void main(String args[])  
    {  
        int a,b;  
        Scanner obj=new Scanner(System.in);  
        System.out.print("enter a :");  
        a=obj.nextInt();  
        if(a!=0)  
        {  
            System.out.println("in if");  
            if(a>0)  
            {  
                System.out.println("a is +ve num");  
            }  
            else  
            {  
                System.out.println("a is -ve num");  
            }  
        }  
        else  
        {  
            System.out.println("a is 0");  
        }  
        System.out.println("program completed");  
    }  
}
```

Output:

```
enter a :5  
in if  
a is +ve num  
program completed
```

4. else if:

syntax:

```
If(condition1)  
{  
    Statement1;  
}  
else If(condition2)  
{
```

```

        Statement2;
    }
Else If(condition3)
{
    Statement3;
}
.
.
.
else
{
    Statement n;
}
Statement;

```

Write a sample java program on else if condition

```

import java.util.*;
class ElseIfEx
{
    public static void main(String args[])
    {
        int a,b,c;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a,b & c values :");
        a=obj.nextInt();
        b=obj.nextInt();
        c=obj.nextInt();
        if((a>b) && (a>c))
        {
            System.out.println("the largest num is:"+a);
        }
        else if((b>a) && (b>c))
        {
            System.out.println("the largest num is:"+b);
        }
        else
        {
            System.out.println("the largest num is:"+c);
        }
        System.out.println("program completed");
    }
}

```

Output:

```

enter a,b & c values :10
20
30
the largest num is:30
program completed

```

5. switch:

A switch statements checks if a variable is equals to list of values and a corresponding statements is executed where each value is called as case.

Syntax:

```

Switch(expression)
{
    case label1:

```

```

    Statements;
    Break;
case label12:
    Statements;
    Break;
case label3:
    Statements;
    Break;
.
.
.
default:
    statements n;
    break;

```

Write a sample java program on SWITCH condition

```

class SwitchEx
{
    public static void main(String args[])
    {
        int a,b;
        char op='+';
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a & b values :");
        a=obj.nextInt();
        b=obj.nextInt();
        switch(op)
        {
            case '+':
                System.out.println("Addition of a & b is : "+(a+b));
                break;
            case '-':
                System.out.println("Substraction of a & b is : "+(a-b));
                break;
            case '*':
                System.out.println("Multiplcation of a & b is : "+(a*b));
                break;
            case '/':
                System.out.println("Division of a & b is : "+(a/b));
                break;
            case '%':
                System.out.println("Modulus of a & b is : "+(a%b));
                break;
            default:
                System.out.println("Invalid choice");
                break;
        }
    }
}

```

Output:

```

enter a & b values :10 20
Addition of a & b is : 30

```

- ❖ From 1.5 version java allowed the wrapper class & enum types.
- ❖ From 1.7 version switch will allow string as a data type.

Note:

In java default case is optional in switch.

In java , in switch case everything is optional.

In switch all the cases are optional.

In java break for default is optional.

In java breaks for cases are optional.

In java, case labels must be within the range of datatype. If not we will get an runtime error

In java case labels must be either direct constants or final constants.

- ❖ Inorder to read characters we have to use a method called charAt() of string class which takes the character value in a string from a specified position.

Example:

```
import java.util.*;
class SwitchEx
{
    public static void main(String args[])
    {
        int a,b;
        char op;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter a & b values :");
        a=obj.nextInt();
        b=obj.nextInt();
        System.out.print("enter operator value:");
        op=obj.next().charAt(0);
        switch(op)
        {
            case '+':
                System.out.println("Addition of a & b is : "+(a+b));
                break;
            case '-':
                System.out.println("Substraction of a & b is : "+(a-b));
                break;
            case '*':
                System.out.println("Multiplication of a & b is : "+(a*b));
                break;
            case '/':
                System.out.println("Division of a & b is : "+(a/b));
                break;
            case '%':
                System.out.println("Modulus of a & b is : "+(a%b));
                break;
            default:
                System.out.println("Invalid choice");
                break;
        }
    }
}
```

Output:

```
enter a & b values :10 20
enter operator value:*
Multiplication of a & b is : 200
```

LOOPING/ITERATIVE STATEMENTS:

Java provides the 3 iterative statements:

1. While
2. Do while
3. For

1. While:

It is the most fundamental looping statement. it repeats the block of statements when the condition is true.

Syntax:

```
initialization
While(condition)
{
    //Statements (or)body of the loop;
    Increment/decrement;
}
```

Write a sample java program using WHILE

```
import java.util.*;
class WhileEx
{
    public static void main(String args[])
    {
        int i=1,sum=0,n;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter number:");
        n=obj.nextInt();
        while(i<=n)
        {
            sum=sum+i;
            i++;
        }
        System.out.print("the sum of numbers is : "+sum);
    }
}
```

Output:

```
enter number:10
the sum of numbers is : 55
```

When we execute the above program when we supply n as 10 , Then we get the output as 55. But, in the above program if sum is not initialized to 0 we get an compilation error stating that “**variable might not been initialised**” because for local variables JVM will not provide default value which was responsible of user . To provide default value but if the same variable declared as instance variable JVM takes the responsibility to provide default this member has to access only through object.

```
import java.util.*;
class WhileEx
{
    int sum;
    public static void main(String args[])
    {
        int i=1,n;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter number:");
        n=obj.nextInt();
```

```
WhileEx ob=new WhileEx();
while(i<=n)
{
    ob.sum=ob.sum+i;
    i++;
}
System.out.print("the sum of numbers is : "+ob.sum);
}
```

Output:

```
enter number:10
the sum of numbers is : 55
```

2. Do while:

This loop is always executes its body atleast once because the test expression is at the bottom of the loop.
This is also known as “exit control loop”

Syntax:

```
Initialization;
do
{
    //Statements (or)body of the loop;
    Increment/decrement;
} While(condition);
```

Write a sample java program using DO WHILE

```
import java.util.*;
class DoWhileEx
{
    public static void main(String args[])
    {
        int i=1,sum=0,n;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter number:");
        n=obj.nextInt();
        do
        {
            sum=sum+i;
            i++;
        }while(i<=n);
        System.out.print("the sum of numbers is : "+sum);
    }
}
```

Output:

```
enter number:10
the sum of numbers is : 55
```

3. For:

This is also known as entry control loop.

Syntax:

```
for(initialization;condition;increment/decrement)
```

- Initialization of the control variable is done first using assignment operators this variables are also known as loop control variables.
- This value of the control variable is tested using test condition.
- When the body of the loop is executed the control is transformed back to the last statement in the loop. Now the control statement is incremented then it is again tested to check whether the condition is satisfied or not. This process continues till the value of control variable fails to satisfy the condition.

Write a sample java program using for loop

```
import java.util.*;
class ForEx
{
    public static void main(String args[])
    {
        int i=1,sum=0,n;
        Scanner obj=new Scanner(System.in);
        System.out.print("enter number:");
        n=obj.nextInt();
        for(i=1;i<=n;i++)
        {
            sum=sum+i;
        }
        System.out.print("the sum of numbers is :" +sum);
    }
}
```

Output:

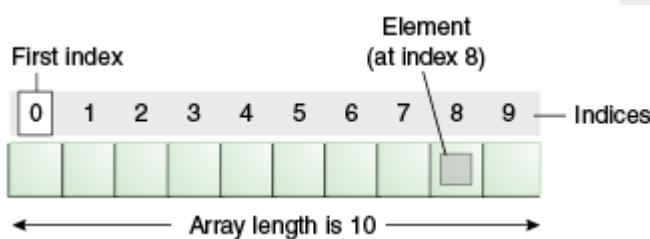
```
enter number:10
the sum of numbers is : 55
```

Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location.

Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

- o **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- o **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

- o **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime.
To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

- o Single Dimensional Array
- o Multidimensional Array

Single Dimensional Array in java

Syntax to Declare an Array in java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in java

1. arrayRefVar=new datatype[size];

Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
class Testarray
{
    public static void main(String args[])
    {
        int a[]={}; //declaration and instantiation
        a[0]=10; //initialization
        a[1]=20;
        a[2]=70;
        a[3]=40;
        a[4]=50;
        //printing array
    }
}
```

```
        for(int i=0;i<a.length;i++)//length is the property of array  
        System.out.println(a[i]);  
    }  
}
```

Output:

```
10  
20  
70  
40  
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. int a[]={33,3,4,5}//declaration, instantiation and initialization

Let's see the simple example to print this array.

```
class Testarray1  
{  
    public static void main(String args[])  
    {  
        int a[]={33,3,4,5}//declaration, instantiation and initialization  
        //printing array  
        for(int i=0;i<a.length;i++)//length is the property of array  
        System.out.println(a[i]);  
    }  
}
```

Test it Now

Output:

```
33  
3  
4  
5
```

Passing Array to method in java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

```
class Testarray2
{
    static void min(int arr[])
    {
        int min=arr[0];
        for(int i=1;i<arr.length;i++)
        {
            if(min>arr[i])
                min=arr[i];
        }
        System.out.println(min);
    }

    public static void main(String args[])
    {
        int a[]={33,3,4,5};
        min(a);//passing array to method
    }
}
```

Test it Now

Output:

3

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)

3. dataType arrayRefVar[][]; (or)

4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
```

```
arr[0][1]=2;
```

```
arr[0][2]=3;
```

```
arr[1][0]=4;
```

```
arr[1][1]=5;
```

```
arr[1][2]=6;
```

```
arr[2][0]=7;
```

```
arr[2][1]=8;
```

```
arr[2][2]=9;
```

Example of Multidimensional java array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3
{
    public static void main(String args[])
    {
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        //printing 2D array
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(arr[i][j]+" ");
            }
        }
    }
}
```

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

Test it Now

Output:

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

CONSTRUCTORS

Constructor in java is a feature which is used to construct the object. The main role of constructor in object creation is to identify the size of the object and to provide initial values to the object.

In java applications constructor will be executed automatically at the time of creating objects.

Constructor are same as class names. Constructor will not allow any modifiers like static , final ...etc.

This can be public, private,protected,default

Syntax:

[Access modifier] classname (parameter list)[throws Exception]

```
{
```

 Statements

```
}
```

In java , a constructor is a block of code which is similar to method and called as instance.

Constructors are not considered as members of the class.

A constructors allows a programmer to provide initial values for field when create the object

TYPES OF CONSTRUCTORS

Default constructor

User defined constructor

1. DEFAULT CONSTRUCTOR

In java, if programmer doesn't provide any constructor ,the compiler will create a default constructor.

Example: before compilation

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

Example: after compilation

```
class A
{
    A()
    {
        //provided by compiler
    }
    int a,b;
    void add()
    {
        System.out.println(a+b);
    }
    public static void main(String args[])
    {
        A a=new A();
    }
}
```

```
        a.add();
    }
}
```

2. USER DEFINED CONSTRUCTOR:

User defined constructors are those which are explicitly provided by the programmers which are of two types.

1. Zero argument or zero parameterized constructor

1. Zero argument constructor:

If we declare any constructor without providing any parameter then that type of constructor is called zero argument constructor.

If user provides any constructor explicitly in a particular class then compiler will not provide default constructor.

Ex:

```
class ZC
{
    ZC()
    {
        System.out.println("constructor");
    }
    public static void main(String args[])
    {
        ZC a=new ZC();
    }
}
```

Output:

```
D:\ZC.java ZC
constructor
```

PARAMETERIZED CONSTRUCTOR:

EXAMPLE:

```
class PC
{
    int a;
    int b;
    PC()
    {
        a=1000;
        b=2000;
        System.out.println("hi Zero");
        display();
    }
}
```

```

PC(int x,int y)
{
    a=x;
    b=y;
    System.out.println("hi args");
    display();
}
void display()
{
    System.out.println("a+b="+ (a+b));
}
public static void main(String args[])
{
    System.out.println("hi main");
    PC obj1=new PC();
    PC obj2=new PC(10,20);
}
}

```

Output:

```

hi main
hi Zero
a+b=3000
hi args
a+b=30

```

This

This is a java keyword which can be current class object. this keyword can be used to represent

1. To refer current class variable
2. To refer current class method
3. To refer current class constructor

1. TO REFER CURRENT CLASS VARIABLE:

- To refer current class variable by using ‘this keyword’.

Syntax : this. var_name

- In java applications when we have same set of variables has local variables and class level variables then inorder to differentiate a class level variable and local variables. then we have to use ‘this’ keyword.
- ‘this’ keyword can be used when class level variables have same meaning with other variables like local variables.

Example program:

```
class thisEx
{
    int a,b;
    void ass(int x,int y)
    {
        a=x;
        b=y;
    }
    void add()
    {
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    {
        thisEx obj1=new thisEx();
        obj1.ass(10,20);
        obj1.add();
    }
}
```

Output:

```
a+b : 30
```

Example program:

```
class thisEx1
{
    int a=10,b=20;
    void display(int x,int y)
    {
        System.out.println("x+y : "+(x+y));
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    {
        thisEx1 obj1=new thisEx1();
        obj1.display(100,200);

    }
}
```

Output:

```
x+y : 300
a+b : 30
```

- When we have local variables and instance/class variables with same preference will be given to local variables.

Example program

```
class thisEx2
{
    int a=10,b=20;
    void display(int a,int b)
    {
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    
```

```
    {
        thisEx2 obj1=new thisEx2();
        obj1.display(100,200);

    }
}
```

Output:

```
a+b : 300
```

- Inorder to differentiate class variables and other variables we have to use ‘this’ keyword.

Example:

```
class thisEx3
{
    int a=10,b=20;
    void display(int a,int b)
    {
        System.out.println("a+b : "+(this.a+this.b));
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    {
        thisEx3 obj1=new thisEx3();
        obj1.display(100,200);

    }
}
```

Output:

```
a+b : 30
a+b : 300
```

2. TO REFER CURRENT CLASS METHOD:

- ‘this’ keyword can be used to refer current class constructor

Syntax: this(parameter list);

 this() -> to call zero argument constructor

 this(10) -> to call parameterized constructor

Example program:

```
class thisConsEx
{
    int a,b;
    thisConsEx()
    {
        this(10,20);
        System.out.println("zero arguement");
    }
    thisConsEx(int x)
    {
        this();
        System.out.println("one arguement");
    }
    thisConsEx(int x,int y)
    {
        System.out.println("two arguement");
    }
}
```

```

        display(x,y);
    }
    void display(int x,int y)
    {
        a=x;
        b=y;
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    {
        thisConsEx obj1=new thisConsEx(10);

    }
}

```

Output:

```

two arguement
a+b : 30
zero arguement
one arguement

```

- When we have to call ‘this’ keyword from the constructor ‘this’ keyword must be first statement otherwise error will be occur at the compilation time.
- No cycle to be formed when ‘this’ keyword with constructor.

Example program:

```

class thisConsEx2
{
    int a,b;
    thisConsEx2()
    {
        this(10,20);
        System.out.println("zero arguement");
    }
    thisConsEx2(int x)
    {
        this();
        System.out.println("one arguement");
    }
    thisConsEx2(int x,int y)
    {
        this(10);
        System.out.println("two arguement");
    }
    public static void main(String args[])
    {
        thisConsEx2 obj1=new thisConsEx2(10);

    }
}

```

Compilation error:

```

thisConsEx2.java:14: error: recursive constructor invocation
        thisConsEx2<int x,int y>

```

- No consecutive ‘this’ to be used in the constructor(in a constructor there has to only one ‘this’ to call another constructor, otherwise compilation error will occur)

Example program:

```
class thisConsEx3
{
    int a,b;
    thisConsEx3()
    {
        this(10,20);
        System.out.println("zero arguement");
    }
    thisConsEx3(int x)
    {
        this();
        this(10,20);
        System.out.println("one arguement");
    }
    thisConsEx3(int x,int y)
    {
        System.out.println("two arguement");
        display(x,y);
    }
    void display(int x,int y)
    {
        a=x;
        b=y;
        System.out.println("a+b : "+(a+b));
    }
    public static void main(String args[])
    {
        thisConsEx3 obj1=new thisConsEx3(10);

    }
}
```

Compilation error:

```
thisConsEx3.java:12: error: call to this must be first statement in constructor
        this(10,20);
               ^
1 error
```

3. TO REFER CURRENT CLASS CONSTRUCTOR:

- To call current call methods we use ‘this’ keyword.

Example program:

```
class thisMethodEx
{
    int a,b;
    void ass()
    {
        display(10,20);
        this.display1(100,200);
    }
    void display1(int x,int y)
    {
        a=x;
```

```
b=y;
System.out.println("display1 a+b : "+(a+b));
}
void display(int x,int y)
{
    a=x;
    b=y;
    System.out.println("display2 a+b : "+(a+b));
}
public static void main(String args[])
{
    thisMethodEx obj1=new thisMethodEx();
    obj1.ass();
}
}
```

Output:

```
display2 a+b : 30
display1 a+b : 300
```

Wrapper classes in java

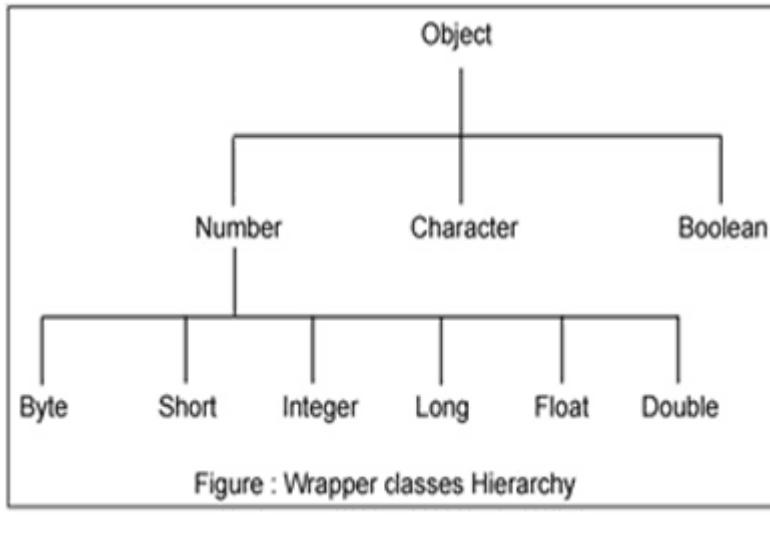
For each and every fundamental data type there exist a pre-defined class, Such predefined class is known as wrapper class. The purpose of wrapper class is to convert numeric string data into numerical or fundamental data.

Why use wrapper classes ?

We know that in java whenever we get input from user, it is in the form of string value so here we need to convert these string values in different different datatype (numerical or fundamental data), for this conversion we use wrapper classes.

Wrapper class in java provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.



Example of wrapper class

```

class WraperDemo
{
    public static void main(String[] args)
    {
        String s[] = {"10", "20"};
        System.out.println("Sum before:" + s[0] + s[1]); // 1020
        int x=Integer.parseInt(s[0]); // convert String to Integer
        int y=Integer.parseInt(s[1]); // convert String to Integer
        int z=x+y;
        System.out.println("sum after: "+z); // 30
    }
}

```

Output:

```

Sum before:1020
sum after: 30

```

Explanation: In the above example 10 and 20 are store in String array and next we convert these values in Integer using "int x=Integer.parseInt(s[0]);" and "int y=Integer.parseInt(s[1]);" statement. In "System.out.println("Sum before:" + s[0] + s[1]);" Statement normally add two string and output is 1020 because these are String numeric type not number.

```

String s = " 10.6f ";
float x = Float.parseFloat(s);
System.out.println(x); // 10.6

```

↓ ↓ ↓
 data wrapper method
 type class name

Converting String data into fundamental or numerical

We know that every command line argument of java program is available in the main() method in the form of array of object of string class on String data, one can not perform numerical operation. To perform numerical operation it is highly desirable to convert numeric String into fundamental numeric value.

Example

"10" --> numeric String --> only numeric string convert into numeric type or value.

"10x" --> alpha-numeric type --> this is not conversion.

"ABC" --> non-numeric String no conversion.

"A" --> char String no conversion.

Only 'A' is convert into ASCII value that is 65 but 'A' is not convert into numeric value because it is a String value.

Fundamental data type and corresponding wrapper classes

The following table gives fundamental data type corresponding wrapper class name and conversion method from numerical String into numerical values or fundamental value.

Fundamental Data Type	Wrapper ClassName	Conversion method from numeric string into fundamental or numeric value
Byte	Byte	public static byte parseByte(String)
Short	Short	public static short parseShort(String)
Int	Integer	public static integer parseInt(String)
Long	Long	public static long parseLong(String)
Float	Float	public static float parseFloat(String)
double	Double	public static double parseDouble(String)
Char	Character	
boolean	Boolean	public static boolean parseBoolean(String)

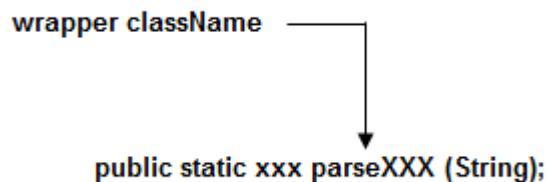
How to use wrapper class methods

All the wrapper class methods are static in nature so we need to call these method using class.methodName().

- for Integer: int x=Integer.parseInt(String);

- for float: float x=Float.parseFloat(String);
- for double: double x=Double.parseDouble(String);

Each and every wrapper class contains the following generalized method for converting numeric String into fundamental values.



Here xxx represents any fundamental data type.



Wrapper class Example: Primitive to Wrapper

```

public class WrapperExample1
{
    public static void main(String args[])
    {
        //Converting int into Integer
        int a=20;
        Integer i=Integer.valueOf(a);//converting int into Integer
        Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
        System.out.println(a+" "+i+" "+j);
    }
}
  
```

Output:

20 20 20

Wrapper class Example: Wrapper to Primitive

```

public class WrapperExample2
{
    public static void main(String args[])
    {
  
```

```

//Converting Integer to int

Integer a=new Integer(3);

int i=a.intValue();//converting Integer to int

int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);

}

}

```

Output:

3 3 3

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```

class Operation
{
    int data=50;
    void change(int data)
    {
        data=data+100;//changes will be in the local variable only
    }
    public static void main(String args[])
    {
        Operation op=new Operation();
        System.out.println("before change "+op.data);
        op.change(500);
        System.out.println("after change "+op.data);
    }
}

```

Output:

**before change 50
after change 50**

DIFFERENCES BETWEEN OBJECT, CLASS AND INSTANCE

OBJECT:

- An object is a real world entity which has state and behavior.
- The object stores its state in terms of fields and exposes its behaviour through methods.

CLASS:

A class is a template or blue print that is used to create object.

Basically a class consists of members as follows

1. Variables
2. Methods
3. Constructors
4. Instance block
5. Static block

INSTANCE:

- An instance is a unique copy of a class representing an object.
- When a new instance of class is created, the JVM will allocate memory for that class.

CLASS:

1. A class is a user defined data type. The process of binding data members and associated methods into a single unit called class.
2. A class is a blue print or template which gives a structure for creation of objects. The properties of class are called attributes (or) states.
3. The behaviour of a class is called as method.

Class structure:

```
class <classname>
{
    variable declaration;
    method definition;
}
```

In the above syntax, class is a keyword which is used for developing user defined datatypes.

1. Class name represents java valid identifiers.
2. Variable declaration represents data members of the class.
3. Method definition represents types of methods which are used to perform some business logic based on user requirement.
4. All methods of java must be defined inside the class only.
5. Each and every class in java has logical existence.

Example:

```
class Student
{
    public static void main(String args[])
    {
        int rollno=10;
        float per=95.5;
        void display()
        {
            System.out.println("main");
        }
    }
}
```

OBJECT:

In java, a class has physical existence but not logical existence therefore no memory space for data members of class is created. If we want to class data members of class. First, memory space has to be allocated for data members which can be done by creating an object with respect to class object is a real world entity which has state and behaviour

(or)

Object is a instance of a class where instance is allocation of sufficient amount of memory for data members.

(or)

Object is a runtime entity.

Object creation:

Whenever an object is create sufficient amount of memory space for data members of a class is allocated.

The creation of object is nothing but dynamic memory allocation. Inorder to creation of objects, there are 4 ways.

1. Using ‘new’ keyword
2. Using ‘clone’ keyword
3. Using ‘class’ keyword
4. Using deserialization

Types of objects:

In java there are 2 types of objects

1. Immutable objects
2. Mutable objects

1. Immutable objects:

These are objects in java which will not allow further modifications or whose state cannot be changed.

Example: string class objects & all wrapper class objects are immutable.

If try to perform with immutable objects content then immutable objects data will be participated in the operation but the result will be stored in another new object.

Example:

```
class ObjectTypesEx
{
    public static void main(String args[])
    {
        String s1=new String("ja");
        String s2=s1.concat("va");
        System.out.println("s1 : "+s1);
        System.out.println("s2 : "+s2);
    }
}
```

Output:

```
s1 : ja
s2 : java
```

2. Mutable objects:

These objects allow modification on its control. By default all java objects are mutable objects.

Example:

```
class ObjectTypesEx
{
    public static void main(String args[])
    {
        StringBuffer s1=new StringBuffer("ja");
        StringBuffer s2=s1.append("va");
        System.out.println("s1 : "+s1);
        System.out.println("s2 : "+s2);
    }
}
```

Output:

```
s1 : java
s2 : java
```

1. USING NEW KEYWORD:

The creation of object can be done by ‘new’ operator which is known as dynamic memory allocation operator.

Syntax:

<class name> <object name>=new <class name>(parameter list)

Example:

Student obj1=new student();

When JVM encounters the above statement, JVM will create an object for the respective class

Syntax2:

<class name> <object name>;

<object name>=new <class name>(parameter list)

In the above statement /syntax the first part is declaration(null value can be assigned) & in second line object is created & refers to reference variable.

Example:

```
class SumOfNos
{
    int a,b,c;
    void ass()
    {
        a=10;
        b=20;
    }
    void add()
    {
        c=a+b;
    }
}
```

```
        }
        void display()
        {
            System.out.println("a+b : "+c);
        }
    }
class ObjectEx
{
    public static void main(String args[])
    {
        SumOfNos obj1=new SumOfNos();
        obj1.ass();
        obj1.add();
        obj1.display();
        System.out.println("obj1 : "+obj1);
        SumOfNos obj2;
        obj2=new SumOfNos();
        obj2.ass();
        obj2.add();
        obj2.display();
        System.out.println("obj2 : "+obj2);
    }
}
```

Output:

```
a+b : 30
obj1 : SumOfNos@18fe7c3
a+b : 30
obj2 : SumOfNos@b8df17
```



OBJECT DESTRUCTION:

- In java technology, programs should not take the responsibility of destructing an object. Because, java has provided a automatic object destruction in the form of garbage collector.
- In java , garbage collector will monitor all the java objects which are created by the programmer & if the garbage collector identifies any object which will be destroyed automatically. If no object is in ideal mode then all the objects will be destroyed at the end of the java program.
- In C language we use ‘free’ function to reclaim the unused memory at runtime automatically.

ADVANTAGES OF GARBAGE COLLECTION:

- By using it, the memory efficiency is increased because of garbage collector removes the unused objects from heap memory.
- As garbage collection is done automatically by the garbage collector for which the programmers don't need to take extra effort.

1. Nullify the object

Syntax: <object name>=null;

2. Activate garbage collector present in system as a static method.

```
class classname
```

```
{
```

```
    public static void gc()
```

```
{
```

```
}
```

```
}
```

3. Inorder to use gc() , we have to use the following syntax system.gc();

then this method is called gc will search for objects whose reference value is nullified & then those objects are destroyed. Just before destroying the object JVM will access finalise method defined in object class inorder to clean up the resources which are associated with respect to objects.

Example: protected void finalize()

4. Finalize() which is used to clean up process will be called the number of times is dependent. The number of objects nullified, but gc() has to be called only once

Example:

```
class ObjectDestructionEx
{
    public void finalize()
    {
        System.out.println("object destroyed");
    }
    public static void main(String args[])
    {
        ObjectDestructionEx obj1=new ObjectDestructionEx();
        ObjectDestructionEx obj2=new ObjectDestructionEx();
        System.out.println("obj1 : "+obj1);
        System.out.println("obj2 : "+obj2);
        obj1=null;
        obj2=null;
        System.out.println("obj1 : "+obj1);
        System.out.println("obj2 : "+obj2);
        System.gc();
    }
}
```

Output:

```

obj1 : ObjectDestructionEx@1e63e3d
obj2 : ObjectDestructionEx@1004901
obj1 : null
obj2 : null
object destroyed

```

Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you concat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

Performance Test of String and StringBuffer

```

public class ConcatTest
{
    public static String concatWithString()
    {
        String t = "Java";
        for (int i=0; i<10000; i++)
        {
            t = t + "Tpoint";
        }
        return t;
    }
    public static String concatWithStringBuffer()
    {
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++)
        {
            sb.append("Tpoint");
        }
        return sb.toString();
    }
    public static void main(String[] args)
    {
        long startTime = System.currentTimeMillis();
        concatWithString();
        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
    }
}

```

```
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time      taken      by      Concating      with      StringBuffer:");
        "+(System.currentTimeMillis()-startTime)+"ms");
    }
}
```

Output:

```
Time taken by Concating with String: 1092ms
Time taken by Concating with StringBuffer: 0ms
```

String and StringBuffer HashCode Test

As you can see in the program given below, String returns new hashCode value when you concat string but StringBuffer returns same.

```
public class InstanceTest
{
    public static void main(String args[])
    {
        System.out.println("Hashcode test of String:");
        String str="java";
        System.out.println(str.hashCode());
        str=str+"tpoint";
        System.out.println(str.hashCode());
        System.out.println("Hashcode test of StringBuffer:");
        StringBuffer sb=new StringBuffer("java");
        System.out.println(sb.hashCode());
        sb.append("tpoint");
        System.out.println(sb.hashCode());
    }
}
```

Output:

```
Hashcode test of String:
3254818
229541438
Hashcode test of StringBuffer:
16795905
16795905
```

Difference between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is <i>synchronized</i> i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is <i>non-synchronized</i> i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is <i>less efficient</i> than StringBuilder.	StringBuilder is <i>more efficient</i> than StringBuffer.

StringBuffer Example

```
public class BufferTest
{
    public static void main(String[] args)
    {
        StringBuffer buffer=new StringBuffer("hello");
        buffer.append("java");
        System.out.println(buffer);
    }
}
```

Output:

hellojava

StringBuilder Example

```
public class BuilderTest
{
    public static void main(String[] args)
    {
        StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
}
```

Output:

hellojava

StringTokenizer in Java

1. [StringTokenizer](#)
2. [Methods of StringTokenizer](#)
3. [Example of StringTokenizer](#)

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

Constructors of StringTokenizer class

There are 3 constructors defined in the StringTokenizer class.

No.	Constructor	Description
1)	StringTokenizer(String str)	creates StringTokenizer with specified string.
2)	StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimiter.
3)	StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object
String nextToken(String delim)	returns the next token based on the delimiter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

```
import java.util.StringTokenizer;
```

```
public class Simple
```

```
{  
    public static void main(String args[])  
    {  
        StringTokenizer st = new StringTokenizer("my name is khan"," ");  
        while (st.hasMoreTokens())  
        {  
            System.out.println(st.nextToken());  
        }  
    }  
}
```

Output:

```
my  
name  
is  
khan
```

Example of nextToken(String delim) method of StringTokenizer class

```
import java.util.*;  
  
public class Test  
{  
    public static void main(String[] args)  
    {  
        StringTokenizer st = new StringTokenizer("my,name,is,adi");  
        // printing next token  
        System.out.println("Next token is : " + st.nextToken(","));  
    }  
}
```

Output:

```
Next token is : my
```

How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

Example to create Immutable class

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
public final class Employee
{
    final String pancardNumber;
    public Employee(String pancardNumber)
    {
        this.pancardNumber=pancardNumber;
    }
    public String getPancardNumber()
    {
        return pancardNumber;
    }
}
```

The above class is immutable because:

- The instance variable of the class is `final` i.e. we cannot change the value of it after creating an object.
- The class is `final` so we cannot create the subclass.
- There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

Java `toString()` method

If you want to represent any object as a string, **`toString()` method** comes into existence.

The `toString()` method returns the string representation of the object.

If you print any object, java compiler internally invokes the `toString()` method on the object. So overriding the `toString()` method, returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of Java `toString()` method

By overriding the `toString()` method of the `Object` class, we can return values of the object, so we don't need to write much code.

Understanding problem without `toString()` method

Let's see the simple code that prints reference.

```
class Student
{
    int rollno;
    String name;
    String city;
    Student(int rollno, String name, String city)
    {
        this.rollno=rollno;
        this.name=name;
        this.city=city;
    }

    public static void main(String args[])
    {
        Student s1=new Student(101,"Raj","lucknow");
        Student s2=new Student(102,"Vijay","ghaziabad");
        System.out.println(s1);//compiler writes here s1.toString()
        System.out.println(s2);//compiler writes here s2.toString()
    }
}
```

Output:

```
Student@18fe7c3
Student@b8df17
```

UNIT-2

INHERITANCE

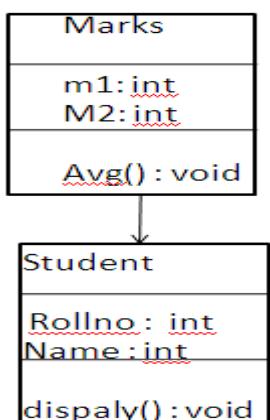
Inheritance is one of the distinct feature in one of the object oriented programming language.

- The process of getting or acquiring the properties and behaviours of one class to another class is called as inheritance.
- The class which is giving properties and behaviour to some other class is called as base class or super class or parent class.
- The class is getting the properties and behaviors from other class or base class is called as derived class or sub class or child class.
- The concept of inheritance is also called as sub classing or extendable classing or derivation or reusability.

ADVANTAGES:

When we use the concept of inheritance in java programming we have the following advantages:

1. Inheritance is used for code extensibility.
 2. Application development time is less.
 3. The execution time of application time is also less.
 4. Memory space is also less.
 5. Redundancy of the code is reduced.
- ❖ The moto of the java is achieved through inheritance.
 - ❖ Inorder to achieve inheritance in java, we have to use ‘extends’ keyword.
 - ❖ Inheritance is also called as “IS A” relationship.



- ❖ In java every class is a child class of object class. The root class for all the java class is object class.
- ❖ One derived class extends only one base class.

Sample example for inheritance:

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

```

    {
        System.out.println("in m2");
    }
}
class B extends A
{
    void m3()
    {
        System.out.println("in m3");
    }
    void m4()
    {
        System.out.println("in m4");
    }
    public static void main(String args[])
    {
        A a=new A();
        a.m1();
        a.m2();
        B b=new B();
        b.m1();
        b.m2();
        b.m3();
        b.m4();
    }
}

```

Output:

```

in m1
in m2
in m1
in m2
in m3
in m4

```

TYPES OF INHERITANCE:

Various types of inheritances in java are:

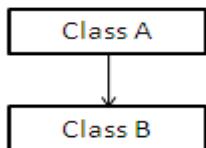
1. Single inheritance
2. Multi level inheritance
3. Hierarchical inheritance

Note:

In java multiple and hybrid is not impossible through inheritance but they were possible through interface.

1. SINGLE INHERITANCE:

The process of acquiring the properties and behaviours of one class to another class is called as single inheritance.



Here a is the parent class and b is the child class

Class A has its own properties and behaviour.

Class B also have its own properties and behaviour along with the properties and behaviours of class A.

Example program on single inheritance

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

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

    public static void main(String args[])
    {
        A obj1=new A();
        B obj2=new B();
        obj1.m1();
        obj2.m1();
        obj2.m2();
    }
}
```

Output:

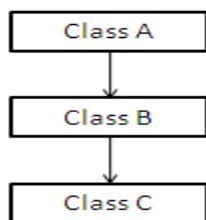
```
in m1  
in m1  
in m2
```

From the above program class A is the parent class and class B is the child class.

- Object a is created with respect to class A and B is created with respect to class B.
- Object a contains the properties & behaviours of class A(i.e, m1()).
- Object b contains the properties & behaviour of class B and also class A.

2. MULTI LEVEL INHERITANCE

- The process of acquiring the properties and behavior of a existing derived class is called multi level inheritance i.e., in multi level inheritance a derived class will be inheriting a parent class as well as the derived class acts as parent class to another class.



- Here A is the parent class and B is the child class for A and parent for class C. and parent for class B.
- i.e., class B can acquire the properties and behaviours of class A similarly class C acquires the properties & behaviours of class B as well as class A(as class B is derived class for class for class A).

Example program on multi level inheritance:

```
class A  
{  
    void m1()  
    {  
        System.out.println("A class m1");  
    }  
}  
class B extends A  
{  
    void m2()  
    {  
        System.out.println("B class m2");  
    }  
}  
class C extends B  
{  
    void m3()  
    {  
        System.out.println("C class m3");  
    }  
    public static void main(String args[])  
    {  
        A obj1=new A();  
        B obj2=new B();  
    }  
}
```

```

        C obj3=new C();
        obj1.m1();
        obj2.m1();
        obj2.m2();
        obj3.m1();
        obj3.m2();
        obj3.m3();
    }
}

```

Output:

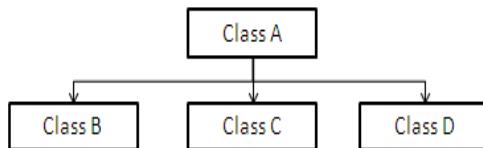
```

A class m1
A class m1
B class m2
A class m1
B class m2
C class m3

```

3. HIERARCHIAL INHERITANCE

- The process of inheriting or giving the properties and behaviour from one class to more than one class is called as hierachial inheritance.



- Here the class A is the parent class, class B & C are the child classes of class A.
- Here both classes B and C can acquire the properties and behaviours of class A.

Example program on hierachial inheritance

```

class A
{
    void m1()
    {
        System.out.println("\nA class m1");
    }
}

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

class H extends A
{
    void m3()
    {
        System.out.println("C class m3");
    }
}

public static void main(String args[])
{
    A obj1=new A();
    B obj2=new B();
}

```

```

        H obj3=new H();
        obj1.m1();
        obj2.m1();
        obj2.m2();
        obj3.m1();
        obj3.m3();
    }
}

```

Output:

```

A class m1
B class m2

A class m1
C class m3

```

SUPER

Super keyword in the java reference which is used to refer immediate parent class objects.

- Whenever we inherit the base class features (or) properties and behaviours into derived class . there can be a situation where base class features may be similar to derived class features. In this situation JVM will be in ambiguity situation to differentiate between base class and child class.
- Inorder to differentiate base class features and derived class features super keyword is used.

Example:

- The super keyword can be used in 3 ways
 1. To refer super class variable
 2. To refer super class methods
 3. To refer super class constructor

1. TO REFER SUPER CLASS VARIABLE

```

class Parent
{
    int a=10,b=20;
}

class Child extends Parent
{
    int a=100,b=200;
    void add(int a,int b)
    {
        System.out.println("Addition of a+b w.r.t method level: "+(a+b));
        System.out.println("Addition of a+b w.r.t child class : "+(this.a+this.b));
    }
}

```

```

        System.out.println("Addition of a+b w.r.t parent class : "+(super.a+super.b));
    }
public static void main(String args[])
{
    Parent obj1=new Parent();
    Child obj2=new Child();
    obj2.add(1000,2000);
}
}

```

Output:

```

Addition of a+b w.r.t method level: 3000
Addition of a+b w.r.t child class : 300
Addition of a+b w.r.t parent class : 30

```

2. TO REFER SUPER CLASS METHODS

```

class Parent
{
    int a=10,b=20;
    void m1()
    {
        System.out.println("Parent class m1");
    }
}

class SuperChildMethod extends Parent
{
    void m1()
    {

        System.out.println("Child class m1");
    }
    void m2()
    {
        super.m1();
        this.m1();
        System.out.println("Child class m2");
    }
}

public static void main(String args[])
{
    Parent obj1=new Parent();
    SuperChildMethod obj2=new SuperChildMethod();
    obj1.m1();
    obj2.m1();
    obj2.m2();
}
}

```

Output:

```

Parent class m1
Child class m1
Parent class m1
Child class m1
Child class m2

```

3. TO REFER SUPER CLASS CONSTRUCTOR

```
class Parent
{
    Parent()
    {
        System.out.println("Parent Constructor");
    }
}
class SuperChildCons extends Parent
{

    SuperChildCons()
    {
        System.out.println("Child class Constructor");
    }
    public static void main(String args[])
    {
        Parent obj1=new Parent();
        SuperChildCons obj2=new SuperChildCons();

    }
}
```

Output:

```
D:\z\aritz>java SuperCons
Parent Constructor
Parent Constructor
Child class Constructor
```

Example program

```
class Parent
{
    Parent()
    {
        System.out.println("Parent Constructor");
    }
    Parent(int x)
    {
        System.out.println("Parent with 1 argument");
    }
}
class SuperChildCons1 extends Parent
{

    SuperChildCons1()
    {
        System.out.println("Child class Constructor");
    }
    SuperChildCons1(int x)
    {
```

```
super(10);
System.out.println("Child class with 1 arguement");
}
public static void main(String args[])
{
    SuperChildCons1 obj1=new SuperChildCons1();
    SuperChildCons1 obj2=new SuperChildCons1(10);

}
```

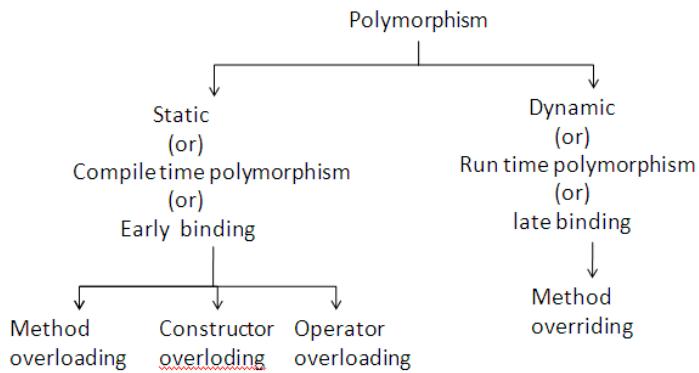
Output:

```
Parent Constructor
Child class Constructor
Parent with 1 arguement
Child class with 1 arguement
```

POLYMORPHISM

POLYMORPHISM:

The process of representing one thing in many forms is known as polymorphism



Polymorphism can be classified into 2 types

1. Static polymorphism
2. Dynamic polymorphism

1. STATIC POLYMORPHISM:

- Static polymorphism is also known as compile time polymorphism or early binding.
- Static polymorphism can be obtained by overloading.
- It can be classified into 3 types.
 - i. Method overloading
 - ii. Constructor overloading
 - iii. Operator overloading

i.METHOD OVERLOADING:

Method overloading in java is that same method name existing with multiple number of times in the same class with
different number of parameters
different order of parameters
different types of parameters

Example program on method overloading with different number of parameters

```
class MethodOverloading
{
    void m1()
    {
        System.out.println("0 arguments");
    }
    void m1(int x)
    {
        System.out.println("1 arguments");
    }
    void m1(int x,int y)
```

```
{  
    System.out.println("2 arguments");  
}  
public static void main(String args[])  
{  
    MethodOverloading obj=new MethodOverloading();  
    obj.m1();  
    obj.m1(10);  
    obj.m1(10,20);  
}  
}
```

Output:

```
0 arguments  
1 arguments  
2 arguments
```

Example program on method overloading with different types of parameters

```
class MethodOverloading1  
{  
    void m1(int x)  
    {  
        System.out.println("int type arguments");  
    }  
    void m1(float x)  
    {  
        System.out.println("float type arguments");  
    }  
    void m1(double x, int y)  
    {  
        System.out.println("double & int arguments");  
    }  
    public static void main(String args[])  
{  
    MethodOverloading1 obj=new MethodOverloading1();  
    obj.m1(10);  
    obj.m1(10.5f);  
    obj.m1(10.56,20);  
}
```

Output:

```
int type arguments  
float type arguments  
double & int arguments
```

Example program on method overloading with different order of parameters

```
class MethodOverloading2  
{  
    MethodOverloading2()  
    {  
        System.out.println("0 arguments");  
    }  
    MethodOverloading2(int x)
```

```

{
    System.out.println("1 arguments");
}
MethodOverloading2(double x, int y)
{
    System.out.println("2 arguments");
}
public static void main(String args[])
{
    MethodOverloading2 obj1=new MethodOverloading2();
    MethodOverloading2 obj2=new MethodOverloading2(10);
    MethodOverloading2 obj3=new MethodOverloading2(10,20);
}
}

```

Output:

```

0 arguments
1 arguments
2 arguments

```

- Overloading is does not effect by changing return type of method which is overloaded

ii.CONSTRUCTOR OVERLOADING:

Constructor overloading in java of a class is done with 3 ways
 different number of parameters
 different order of parameters
 different types of parameters

Example program on constructor overloading with different number of parameters

```

class ConsNo
{
    ConsNo()
    {
        System.out.println("0 arguement");
    }
    ConsNo(int x)
    {
        System.out.println("1 arguement");
    }
    ConsNo(int x,int y)
    {
        System.out.println("2 arguement");
    }
    public static void main(String args[])
    {
        ConsNo obj1=new ConsNo();
        ConsNo obj2=new ConsNo(10);
        ConsNo obj3=new ConsNo(10,20);
    }
}

```

Output:

```
0 arguement  
1 arguement  
2 arguement
```

example program on constructor overloading with different types of parameters

```
class ConsTypes  
{  
    ConsTypes(int x)  
    {  
        System.out.println("int type arguments");  
    }  
    ConsTypes(float x)  
    {  
        System.out.println("float type arguments");  
    }  
    ConsTypes(double x, int y)  
    {  
        System.out.println("double & int arguments");  
    }  
    public static void main(String args[])  
    {  
        ConsTypes obj1=new ConsTypes(10);  
        ConsTypes obj2=new ConsTypes(10.5f);  
        ConsTypes obj3=new ConsTypes(10.56,20);  
    }  
}
```

Output:

```
int type arguments  
float type arguments  
double & int arguments
```

Example program on constructor overloading with different types of parameters

```
class ConsOrder  
{  
    ConsOrder(int x,float y)  
    {  
        System.out.println("int & float arguments");  
    }  
    ConsOrder(float y, int x)  
    {  
        System.out.println("float & int arguments");  
    }  
    public static void main(String args[])  
    {  
        ConsOrder obj2=new ConsOrder(10,20.4f);  
        ConsOrder obj3=new ConsOrder(10.6f,20);  
    }  
}
```

Output:

```
int & float arguments  
float & int arguments
```

iii.OPERATOR OVERLOADING:

```
class OpOverloading
{
    public static void main(String aegs[])
    {
        int a=10,b=20;
        String s1="hello",s2="world";
        System.out.println("a+b : "+(a+b));
        System.out.println("s1+s2 : "+(s1+s2));
    }
}
```

Output:

```
a+b : 30
s1+s2 : helloworld
```

2. DYNAMIC POLYMORPHISM:

- Dynamic polymorphism is also known as “Run time polymorphism” or “late binding”.
- Dynamic polymorphism is one in which methods will bind with an object at runtime i.e., method overriding
- The below program illustrates about method overriding

```
class OverridingP
{
    void m1()
    {
        System.out.println("parent m1");
    }
}

class OverridingC extends OverridingP
{
    void m1()
    {
        System.out.println("child m1");
    }
    public static void main(String args[])
    {
        OverridingC obj1=new OverridingC();
        obj1.m1();
        OverridingP obj2=new OverridingP();
        obj2.m1();
        OverridingP obj3=new OverridingC();
        obj3.m1();
    }
}
```

Output:

```
child m1
parent m1
child m1
```

ABSTRACTION

ABSTRACTION

Abstraction in object oriented programming language is the process of hiding the implementation details from the user and providing the functionality to the user.

Example: banking ATM's where we have withdraw , transfer . . . etc options

➤ In java abstraction is achieved by using the following 2 concepts

1. Abstract classes
2. Interfaces

1. ABSTRACT CLASS:

In every program in java must be written with the concept of classes only.

In java programming language we have 2 types of classes

- i.Concrete class
- ii.Abstract class

i.CONCRETE CLASS:

A concrete class which consists of concrete or incremented methods only i.e., A concrete class contains fully defined methods only, which can be accessed only by using objects.

Example program:

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

Output:

```
m1  
m2
```

i.ABSTRACT CLASS:

An abstract class is that which contains defined & undefined methods , these undefined methods of class are called as abstract methods

➤ These undefined methods to make as abstract methods. We have to prefix with abstract keyword.

Syntax:

```
Abstract return type<method name>(parameter list)
```

Example:

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

DIFFERENTIATE BETWEEN CONCRETE METHODS AND ABSTRACT METHODS

CONCRETE METHOD	ABSTRACT METHOD
1. Concrete() is a normal java method which will have method declaration and method implementation 2. Concrete() method provides less sharability. 3. These are written in normal class Example: class Concrete { void m1() { System.out.println("m1"); } void m2() { System.out.println("m2"); } public static void main(String args[]) { Concrete obj=new Concrete(); obj.m1(); obj.m2(); } }	1. Abstract() is a java method which will have method declaration. 2. Abstract () provides more sharability. 3. Abstract () has to be written in abstract class only. Example : abstract class A { abstract void m1(); }

ABSTRACT CLASS:

A class which is declared as abstract class and which contains atleast one method as an abstract .

- To specify a class as abstract we have to use the abstract modifier
- The abstract classes are used to provide more sharability.
- Abstract classes are used not to provide instantiation for particular class.
- The abstract class contains both implemented(concrete) and unimplemented(abstract) methods, these unimplemented or abstract cmethods implementation has to be provided in the classes which extends the abstract class.

DIFFERENTIATE BETWEEN CONCRETE CLASSES AND ABSTRACT CLASSES

CONCRETE CLASS	ABSTRACT CLASS
1. Concrete class consists of zero or more number of concrete methods only.	1. Abstract class contains zero or more number of concrete and abstract methods.
2. In concrete class instantiation is possible	2. In abstract class instantiation is not possible.
3. It is not mandatory to implement or to override the methods of a concrete class in a sub class.	3. It is mandatory to implement or to override the methods of a abstract class

1. A abstract class is that which consists of zero or more abstract methods and zero or more concrete methods

Example:

```
abstract class AParent
{
    abstract void m1();
    void m2()
    {
        System.out.println("Parent m2");
    }
}
class AbstractClass1 extends AParent
{
    void m1()
    {
        System.out.println("child m1");
    }
    public static void main(String args[])
    {
        AbstractClass1 obj=new AbstractClass1();
        obj.m1();
        obj.m2();
    }
}
```

Output:

```
child m1
Parent m2
```

2. It is not mandatory to extend the abstract class

Example:

```
abstract class AParent
{
    abstract void m1();
    void m2()
    {
        System.out.println("Parent m2");
    }
}
```

```

class AbstractClass2
{
    void m1()
    {
        System.out.println("child m1");
    }
    public static void main(String args[])
    {
        AbstractClass2 obj=new AbstractClass2();
        obj.m1();
        obj.m2();
    }
}

```

Compilation error:

```

AbstractClass2.java:19: error: cannot find symbol
    obj.m2();
           ^
      symbol:   method m2()
      location: variable obj of type AbstractClass2
1 error

```

- When an abstract class is made as parent class it is mandatory to provide the implementation to abstract methods of parent class otherwise it raises compilation error stating that “ abstract method of parent class is not overriden in child class”

Example:

```

abstract class AParent
{
    abstract void m1();
    void m2()
    {
        System.out.println("Parent m2");
    }
}
class AbstractClass3 extends AParent
{
    public static void main(String args[])
    {
        AbstractClass3 obj=new AbstractClass3();
        obj.m2();
    }
}

```

Compilation error:

```

AbstractClass3.java:9: error: AbstractClass3 is not abstract and does not override
    abstract method m1() in AParent
class AbstractClass3 extends AParent
           ^
1 error

```

- It is not mandatory that a abstract class should have abstract method .

Example:

```

abstract class AParent
{
    void m2()
    {

```

```

        System.out.println("Parent m2");
    }
}
class AbstractClass4 extends AParent
{
    public static void main(String args[])
    {
        AbstractClass4 obj=new AbstractClass4();
        obj.m2();
    }
}

```

Output :

```
Parent m2
```

5. An abstract class can have only abstract methods also. When extended by a child class, for all the abstract methods of abstract classes implementation has to be provided in the child class. If not we get compilation error stating that

Example:

```

abstract class AParent
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

class AbstractClass5 extends AParent
{
    void m2()
    {
        System.out.println("m2");
    }

    void m3()
    {
        System.out.println("m3");
    }

    public static void main(String args[])
    {
        AbstractClass5 obj=new AbstractClass5();
        obj.m2();
        obj.m3();
    }
}

```

Compilation error:

```
AbstractClass5.java:7: error: AbstractClass5 is not abstract and does not override
abstract method m1() in AParent
class AbstractClass5 extends AParent
^
1 error
```

Example2:

```

abstract class AParent
{
    abstract void m1();
    abstract void m2();
```

```
abstract void m3();  
}  
class AbstractClass5 extends AParent  
{  
    void m1()  
    {  
        System.out.println("m1");  
    }  
    void m2()  
    {  
        System.out.println("m2");  
    }  
    void m3()  
    {  
        System.out.println("m3");  
    }  
    public static void main(String args[])  
    {  
        AbstractClass5 obj=new AbstractClass5();  
        obj.m1();  
        obj.m2();  
        obj.m3();  
    }  
}
```

Output:

```
m1  
m2  
m3
```

6. An abstract class can also be an empty class

Example:

```
abstract class AParent  
{  
}  
class AbstractClass6 extends AParent  
{  
    void m1()  
    {  
        System.out.println("m1");  
    }  
    public static void main(String args[])  
    {  
        AbstractClass6 obj=new AbstractClass6();  
        obj.m1();  
    }  
}
```

Output:

```
m1
```

7. For abstract class we can have more than one child i.e, with abstract class concept we can implements more than one child class.

Example:

```
abstract class AParent
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}
class AbstractClassB extends AParent
{
    void m1()
    {
        System.out.println("m1");
    }
    void m2()
    {
        System.out.println("m2");
    }
    void m3()
    {
        System.out.println("m3");
    }
}
class AbstractClass7 extends AParent
{
    void m1()
    {
        System.out.println("m1");
    }
    void m2()
    {
        System.out.println("m2");
    }
    void m3()
    {
        System.out.println("m3");
    }
}
public static void main(String args[])
{
    AbstractClass7 obj1=new AbstractClass7();
    obj1.m1();
    obj1.m2();
    obj1.m3();
    AbstractClassB obj2=new AbstractClassB();
    obj2.m1();
    obj2.m2();
    obj2.m3();
}
```

Output:

```
m1
m2
m3
m1
m2
m3
```

8. An abstract class can extend another abstract class and can provide implementation for all the abstract methods of its parent class which can be accessed in its child class with the object of child class.

Example:

```
abstract class AParent
{
    abstract void m1();
    abstract void m2();
    abstract void m3();
}

abstract class AbstractClassB extends AParent
{
    void m1()
    {
        System.out.println("m1");
    }

    void m2()
    {
        System.out.println("m2");
    }

    void m3()
    {
        System.out.println("m3");
    }
}

class AbstractClass8 extends AbstractClassB
{
    void m1()
    {
        System.out.println("m1");
    }

    void m2()
    {
        System.out.println("m2");
    }

    void m3()
    {
        System.out.println("m3");
    }
}

public static void main(String args[])
{
    AbstractClass8 obj1=new AbstractClass8();
    obj1.m1();
    obj1.m2();
    obj1.m3();
}
```

Output:

```
m1
m2
m3
```

9. An abstract class can extend another abstract class and these abstract class can implement the abstract class method of parent class, which intern can have abstract method for which implementation can be provided abstract in another class.

Example:

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

abstract class AbstractClassB extends AParent
{
    abstract void m2();
    void m1()
    {
        System.out.println("m1");
    }
}

class AbstractClass9 extends AbstractClassB
{
    void m2()
    {
        System.out.println("m2");
    }

    public static void main(String args[])
    {
        AbstractClass9 obj1=new AbstractClass9();
        obj1.m1();
        obj1.m2();
    }
}
```

Output:

```
m1
m2
```

10. An abstract class can extend another class and both abstract classes can be implemented in child class of abstract class(child abstract class of parent abstract class).

Example:

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

abstract class AbstractClassB extends AParent
{
    abstract void m2();
}

class AbstractClass10 extends AbstractClassB
{
    void m1()
    {
        System.out.println("m1");
    }

    void m2()
```

```
{  
    System.out.println("m2");  
}  
public static void main(String args[])  
{  
    AbstractClass10 obj1=new AbstractClass10();  
    obj1.m1();  
    obj1.m2();  
}  
}
```

Output:

```
m1  
m2
```

11. An abstract class can extend abstract class and the child abstract class can consists of zero methods which is extended by another class to provide implementation for parent abstract class.

Example:

```
abstract class AParent  
{  
    abstract void m1();  
}  
abstract class AbstractClassB extends AParent  
{  
}  
class AbstractClass11 extends AbstractClassB  
{  
    void m1()  
    {  
        System.out.println("m1");  
    }  
    public static void main(String args[])  
    {  
        AbstractClass11 obj1=new AbstractClass11();  
        obj1.m1();  
    }  
}
```

Output:

```
m1
```

12. An abstract class cannot be instantiated which leads to compilation error.

Example:

```
abstract class AParent  
{  
    abstract void m1();  
}  
class AbstractClass12  
{  
    void m1()  
    {  
        System.out.println("m1");  
    }  
    public static void main(String args[])  
    {
```

```

AbstractClass12 obj1=new AbstractClass12();
obj1.m1();
AParent obj2=new AParent();
}
}
Compilation error:
AbstractClass12.java:15: error: AParent is abstract; cannot be instantiated
    AParent obj2=new AParent();
1 error

```

13. When an abstract class has a child class. We can create object of child class which can be referred by parent class reference variable.

Example:

```

abstract class AParent
{
    abstract void m1();
}

class AbstractClass13 extends AParent
{
    void m1()
    {
        System.out.println("m1");
    }

    public static void main(String args[])
    {
        AbstractClass13 obj1=new AbstractClass13();
        obj1.m1();
        AParent obj2=new AbstractClass13();
        obj2.m1();
    }
}

```

Output:

```

m1
m1

```

14. An abstract class can have main(), as method is a concrete() and you can also create objects of another classes of another classes , other than abstract classes in this class.

Example:

```

class AParent
{
    void m1()
    {
        System.out.println("m1");
    }
}

abstract class AbstractClass14
{
    public static void main(String args[])
    {
        AParent obj1=new AParent();
        obj1.m1();
    }
}

```

Output:

m1

15. Abstract class can extend a abstract class and also a normal class.

Example:

```
class AParent
{
    void m1()
    {
        System.out.println("m1");
    }
}
abstract class AbstractClass15 extends AParent
{
    public static void main(String args[])
    {
        AParent obj1=new AParent();
        obj1.m1();
    }
}
```

Output:

m1

16. An abstract class can have static methods.

Example:

```
class AParent
{
    void m1()
    {
        System.out.println("normal class m1");
    }
}
abstract class AbstractClass16 extends AParent
{
    static void m2()
    {
        System.out.println("abstract class m2");
    }
    public static void main(String args[])
    {
        AParent obj1=new AParent();
        obj1.m1();
        m2();
    }
}
```

Output:

**normal class m1
abstract class m2**

17. An abstract class can also have constructors because

- i. An abstract class can have
- ii. An abstract class which consists of normal methods called as concrete methods

Example:

```
abstract class AParent

{
    AParent()
    {
        System.out.println("abstract class constructor");
    }
}

class AbstractClass17 extends AParent

{
    AbstractClass17()
    {
        System.out.println("normal class constructor");
    }

    public static void main(String args[])
    {
        AbstractClass17 obj1=new AbstractClass17();
    }
}
```

Output:

```
abstract class constructor
normal class constructor
```

18. An abstract class can also have normal variables.

Example:

```
abstract class AParent

{
    int i;
    AParent()
    {
        System.out.println("abstract class constructor");
    }
}

class AbstractClass18 extends AParent

{
    AbstractClass18()
    {
        System.out.println("normal class constructor");
    }
}
```

```
public static void main(String args[])
{
    AbstractClass18 obj1=new AbstractClass18();
    System.out.println("abstract class variable : "+obj1.i);
}
```

Output:

```
abstract class constructor
normal class constructor
abstract class variable : 0
```

19. Abstract class constructor is called by child class constructor at the time of child class objects creation by calling super();

Example:

```
abstract class AParent
{
    int i;
    AParent()
    {
        System.out.println("abstract class constructor");
    }
}
class AbstractClass19 extends AParent
{
    AbstractClass19()
    {
        super();
        System.out.println("normal class constructor");
    }
}
public static void main(String args[])
{
    AbstractClass19 obj1=new AbstractClass19();
    System.out.println("abstract class variable : "+obj1.i);
}
```

Output:

```
abstract class constructor
normal class constructor
abstract class variable : 0
```

20. An abstract class can have both abstract and static methods.

Example:

```
abstract class AParent
{
    abstract void m1();
    static void m2()
    {
        System.out.println("abstract class m2");
    }
}
class AbstractClass20 extends AParent
{
    void m1()
    {
```

```

        System.out.println("abstract class m1");
    }
public static void main(String args[])
{
    AbstractClass20 obj1=new AbstractClass20();
    obj1.m1();
    m2();
}
}

```

Output:

```

abstract class m1
abstract class m2

```

21. Both super & this keywords can be used in abstract class.

Example:

```

abstract class AParent
{
    AParent()
    {
        System.out.println("A abstract class with 0 arguements");
    }
}

class AbstractClassB extends AParent
{
    AbstractClassB()
    {
        this(10);
        System.out.println("B abstract class with 0 arguements");
    }

    AbstractClassB(int x)
    {
        super();
        System.out.println("B abstract class with 1 arguements");
    }
}

class AbstractClass21 extends AbstractClassB
{
    AbstractClass21()
    {
        System.out.println("child class constructor with 0");
    }
}

public static void main(String args[])
{
    AbstractClass21 obj1=new AbstractClass21();
}
}

```

Output:

```

A abstract class with 0 arguements
B abstract class with 1 arguements
B abstract class with 0 arguements
child class constructor with 0

```

22. Abstract methods can be with any number of arguments and with any return type.

Example:

```
abstract class AParent
{
    abstract void m1(int x);
}
class AbstractClass22 extends AParent
{
    void m1(int x)
    {
        System.out.println("1 arguement : "+x);
    }
    public static void main(String args[])
    {
        AbstractClass22 obj1=new AbstractClass22();
        obj1.m1(10);
    }
}
```

Output:

```
1 arguement : 10
```

Java Inner Class

Java inner class or nested class is a class i.e. declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class
```

```
{
```

```
//code
```

```
class Java_Inner_class
```

```
{
```

```
//code
```

```
}
```

```
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

- 1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
- 2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization:** It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

1. Non-static nested class(inner class)

- o a)Member inner class
 - o b)Anonymous inner class
 - o c)Local inner class
2. Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer
{
    //code
    class Inner
    {
        //code
    }
}
```

Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1
{
    private int data=30;
    class Inner
    {
        void msg()
        {
            System.out.println("data is "+data);
        }
    }
    public static void main(String args[])
    {
```

```
TestMemberOuter1 obj=new TestMemberOuter1();
TestMemberOuter1.Inner in=obj.new Inner();
in.msg();
}
}
```

Output:

```
data is 30
```

Internal working of Java member inner class

The java compiler creates two class files in case of inner class. The class file name of inner class is "Outer\$Inner". If you want to instantiate inner class, you must have to create the instance of outer class. In such case, instance of inner class is created inside the instance of outer class.

Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

```
abstract class Person
{
    abstract void eat();
}

class TestAnonymousInner
{
    public static void main(String args[])
    {
        Person p=new Person()
        {
            void eat()
            {
                System.out.println("nice fruits");
            }
        };
        p.eat();
    }
}
```

Output:

nice fruits

Internal working of given code

```
Person p=new Person()  
{  
    void eat()  
    {  
        System.out.println("nice fruits");  
    }  
};
```

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Person type.

Java anonymous inner class example using interface

```
interface Eatable  
{  
    void eat();  
}  
class TestAnonymousInner1  
{  
    public static void main(String args[])  
    {  
        Eatable e=new Eatable()  
        {  
            public void eat()  
            {  
                System.out.println("nice fruits");  
            }  
        };  
        e.eat();  
    }  
}
```

Output:

nice fruits

Internal working of given code

It performs two main tasks behind this code:

```
Eatable e=new Eatable()  
{  
    public void eat()  
    {  
        System.out.println("nice fruits");  
    }  
};
```

1. A class is created but its name is decided by the compiler which implements the Eatable interface and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Eatable type.

Java Local inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Java local inner class example

```
public class localInner1
{
    private int data=30;//instance variable
    void display()
    {
        class Local
        {
            void msg()
            {
                System.out.println(data);
            }
        }
        Local l=new Local();
        l.msg();
    }
    public static void main(String args[])
    {
        localInner1 obj=new localInner1();
        obj.display();
    }
}
```

Output:

30

Java static nested class

A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Java static nested class example with instance method

```
class TestOuter1
{
    static int data=30;
    static class Inner
    {
        void msg()
        {
            System.out.println("data is "+data);
        }
    }
}
```

```
public static void main(String args[])
{
    TestOuter1.Inner obj=new TestOuter1.Inner();
    obj.msg();
}
}
```

Output:

```
data is 30
```

Java Nested Interface

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitly.

Syntax of nested interface which is declared within the interface

```
interface interface_name
{
    ...
    interface nested_interface_name
    {
        ...
    }
}
```

Syntax of nested interface which is declared within the class

```
class class_name
{
    ...
    interface nested_interface_name
    {
        ...
    }
}
```

```
    }  
}
```

Example of nested interface which is declared within the interface

In this example, we are going to learn how to declare the nested interface and how we can access it.
interface Showable

```
{  
  
    void show();  
  
    interface Message  
    {  
  
        void msg();  
    }  
}  
  
class TestNestedInterface1 implements Showable.Message  
{  
  
    public void msg()  
    {  
  
        System.out.println("Hello nested interface");  
    }  
  
    public static void main(String args[])  
    {  
  
        Showable.Message message=new TestNestedInterface1();//upcasting here  
        message.msg();  
    }  
}
```

Output:

```
Hello nested interface
```

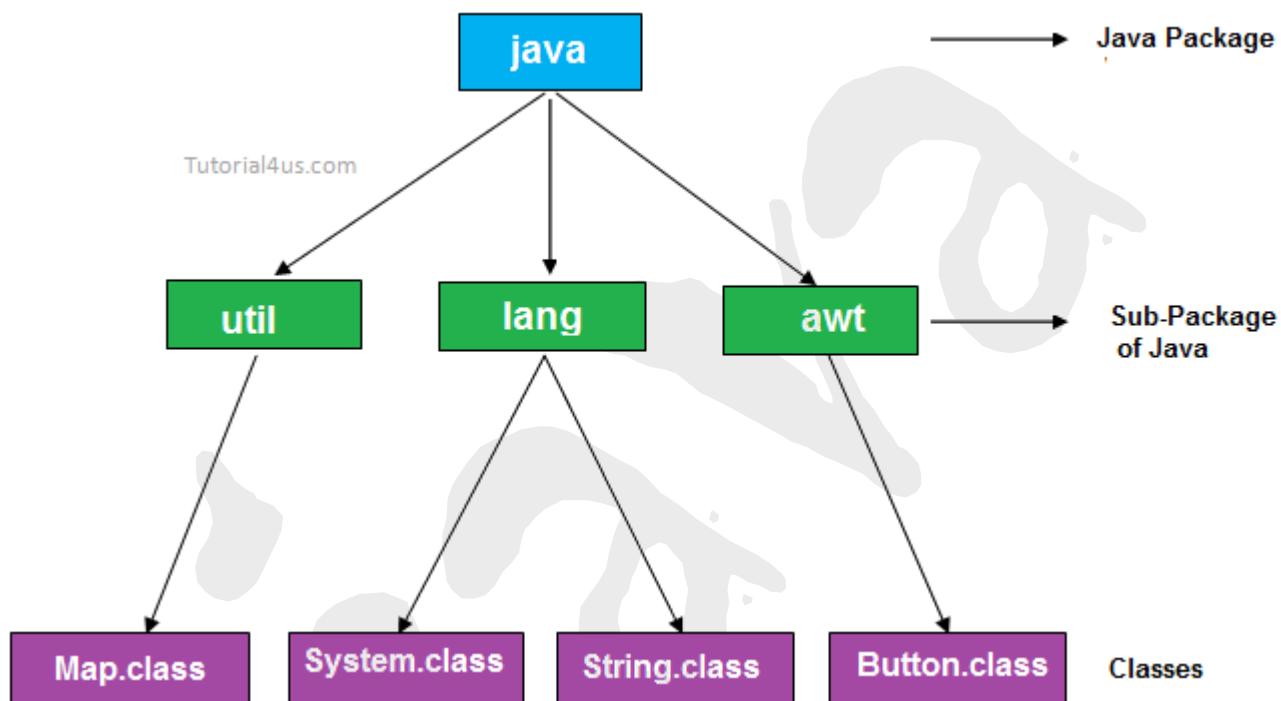


Package in Java

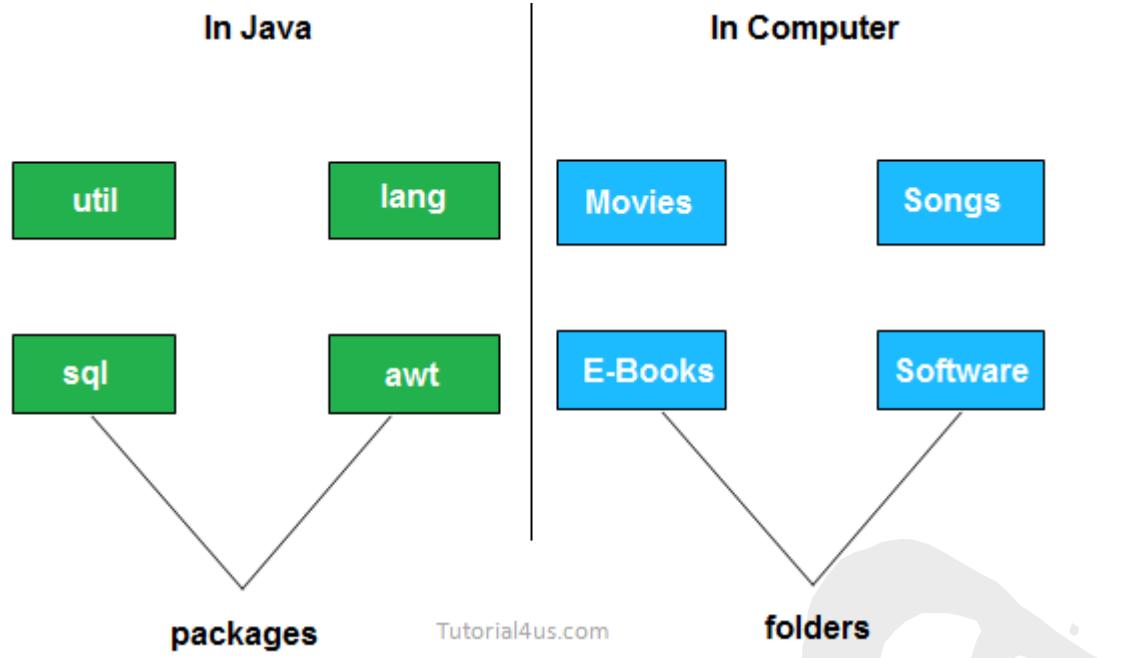
A package is a collection of similar types of classes, interfaces and sub-packages.

Purpose of package

The purpose of package concept is to provide common classes and interfaces for any program separately. In other words if we want to develop any class or interface which is common for most of the java programs than such common classes and interfaces must be place in a package.



Packages in Java are the way to organize files when a project has many modules. Same like we organized our files in Computer. For example we store all movies in one folder and songs in other folder, here also we store same type of files in a particular package for example in awt package have all classes and interfaces for design GUI components.



Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Type of package

Package are classified into two type which are given below.

1. Predefined or built-in package
2. User defined package

Predefined or built-in package

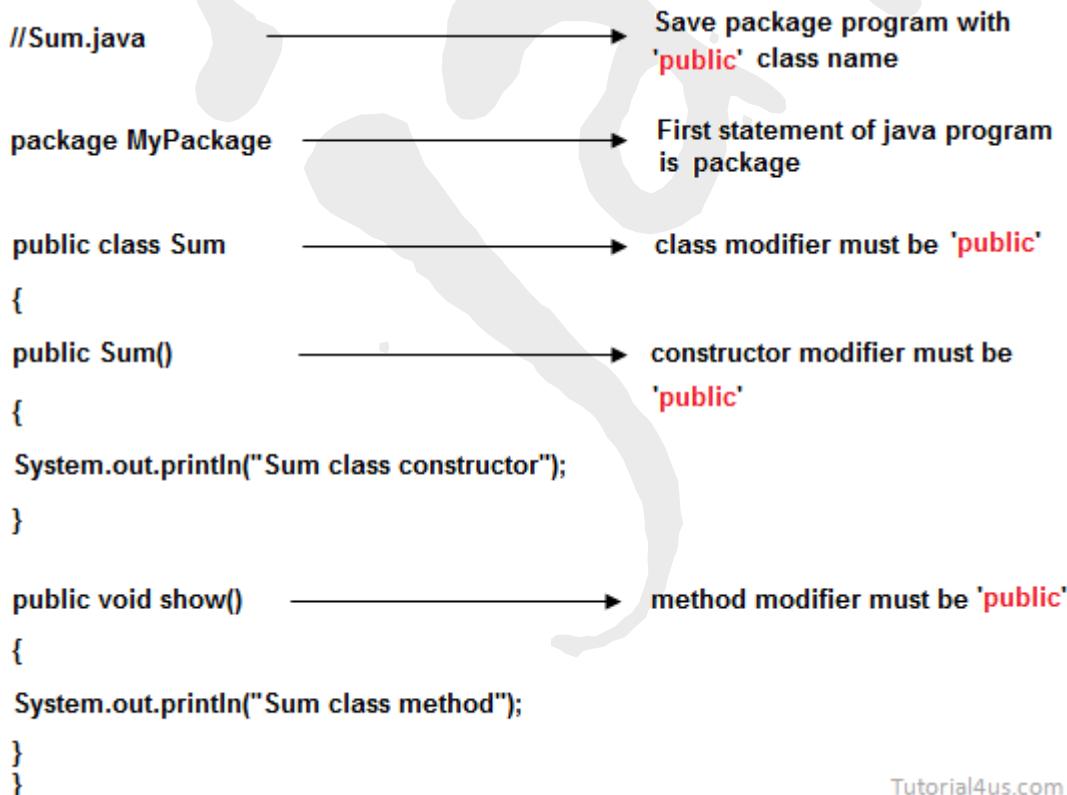
These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

User defined package

If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface have no constructor.)
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)
- Every package program should be save either with public class name or public Interface name



Compile package programs

For compilation of package program first we save program with public className.java and it compile using below syntax:

Syntax

```
javac -d . className.java
```

Syntax

```
javac -d path className.java
```

Explanations: In above syntax "**-d**" is a specific tool which is tell to java compiler create a separate folder for the given package in given path. When we give specific path then it create a new folder at that location and when we use . (dot) then it crate a folder at current working directory.

Note: Any package program can be compile but can not be execute or run. These program can be executed through user defined program which are importing package program.

Example of package program

Package program which is save with A.java and compile by javac -d . A.java

Example

```
package mypack;  
public class A  
{  
    public void show()  
    {  
        System.out.println("Sum method");  
    }  
}
```

Import above class in below program using import packageName.className

Example

```
import mypack.A;  
public class Hello  
{  
    public static void main(String arg[])  
    {  
        A a=new A();  
        a.show();  
        System.out.println("show() class A");  
    }  
}
```

```
}
```

Explanations: In the above program first we create Package program which is save with A.java and compiled by "javac -d . A.java". Again we import class "A" in class Hello using "import mypack.A;" statement.

Difference between Inheritance and package

Inheritance concept always used to reuse the feature within the program between class to class, interface to interface and interface to class but not accessing the feature across the program. Package concept is to reuse the feature both within the program and across the programs between class to class, interface to interface and interface to class.

Difference between package keyword and import keyword

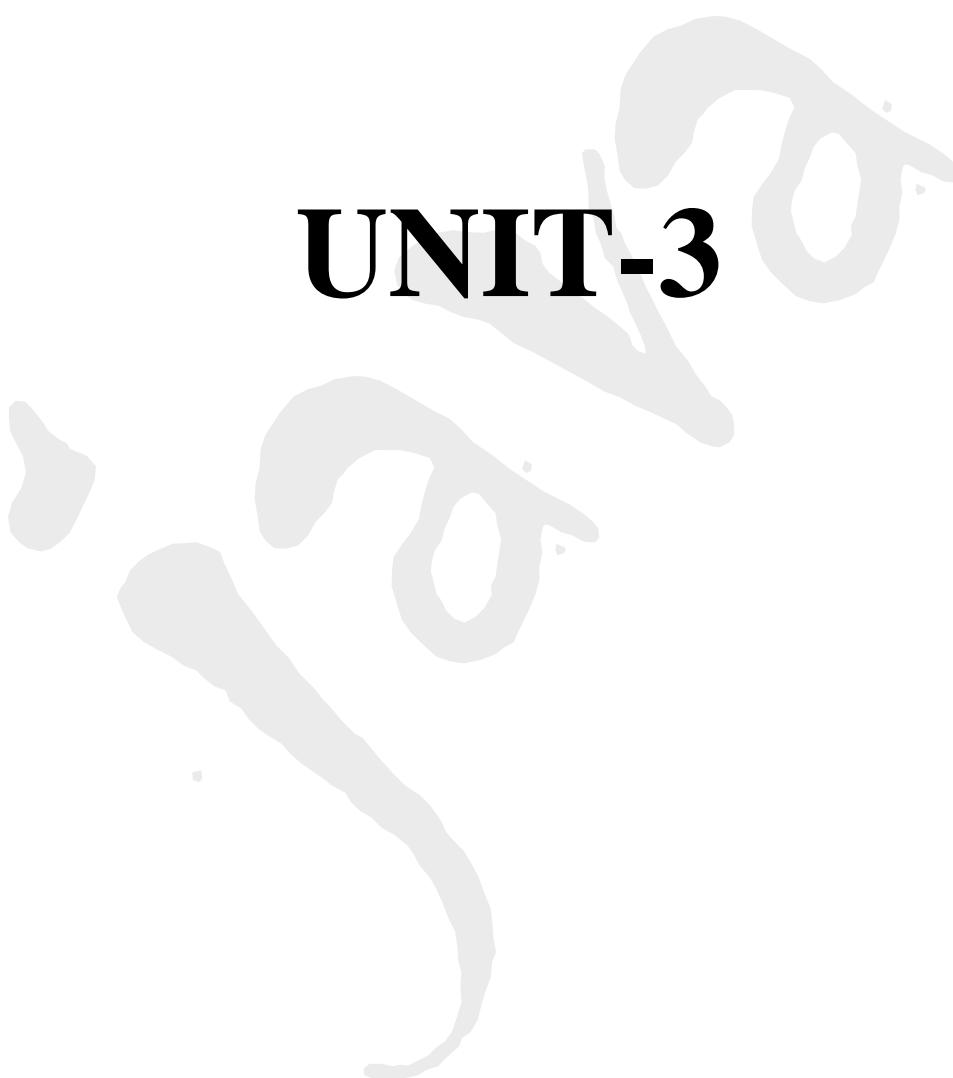
Package keyword is always used for creating the undefined package and placing common classes and interfaces.

import is a keyword which is used for referring or using the classes and interfaces of a specific package.

Access Specifiers

- private: accessible only in the class
- no modifier: so-called “package” access — accessible only in the same package
- protected: accessible (inherited) by subclasses, and accessible by code in same package
- public: accessible anywhere the class is accessible, and inherited by subclasses

Access By	private	package	protected	public
the class itself	yes	yes	yes	yes
a subclass in same package	no	yes	yes	yes
non-subclass in same package	no	yes	yes	yes
a subclass in other package	no	no	yes	yes
non-subclass in other package	no	no	no	yes



UNIT-3

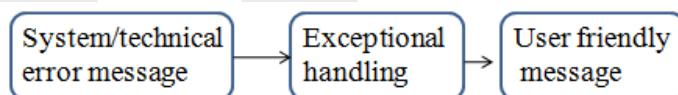
EXCEPTIONAL HANDLING

- An exception is that unwanted or an unexpected event that disturbs normal flow of a program is called as “exception”.

Eg: file not found, arithmetic exception, stack overflow, null pointer, number format exception etc.
- The main objective of exceptional handling is to make a graceful termination of a program.
- Exceptional handling doesn’t mean repairing an exception but to provide an alternative way to continue with the rest of the program normally.
- When we write any program in any programming language, we get the following errors in types of situations.
 - Compile time
 - Run time
- Compile time error occurs when program is syntactically incorrect, like semicolon missing, PLP- possible loss of precision etc.
- Run time error are those which are generally listed or resulted when an user enters invalid input at the time of running a program.
- The languages like C, C++ are treated as weak programming languages because these languages don’t have the capacity to handle the exceptions which are occurs during program execution
- If we develop any project/application by using these kind of language & if an error occurs they don’t have the capability to convert this error message to a user friendly message. As these error messages are understandable only to program developers.
- User friendly messages are those which are easily understandable by non-voice user.
- If an exception occurs in a programming language, the program terminated abnormally & by default a system or a technical error message are displayed.

DEFINITION OF EXCEPTIONAL HANDLING:

The process of converting a system or technical message to a user friendly message is called an exceptional handling.



INTERNAL FLOW OF GENERATING ERROR MESSAGE:

- Each and every operation/action in java must be performed with respective to a object
- Whenever an exception occurs in a java program ‘JVM’ by default creates an exception object.
- When an error occurs in a method, the method creates an object & handles it off to the runtime system[JRE] this object is called as exception object which contains information about error including its type & state of the program i.e., when the error occurred.
[JRE = JVM + java API]
- Creating a exception object & handing to the runtime system is called an throwing a exception.
- The following exception format information is present in the exception object created.

Name: Description

At place

Example program:

```
class CSEExp
```

```

{
    void m1()
    {
        System.out.println("iam in m1");

        m2();
    }

    void m2()
    {
        System.out.println("iam in m2");

        System.out.println(10/0);
    }

    public static void main(String args[])
    {
        System.out.println("before calling");

        CSEExp obj=new CSEExp();

        obj.m1();

        System.out.println("after calling");
    }
}

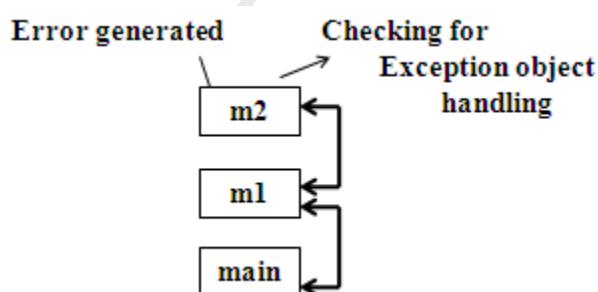
```

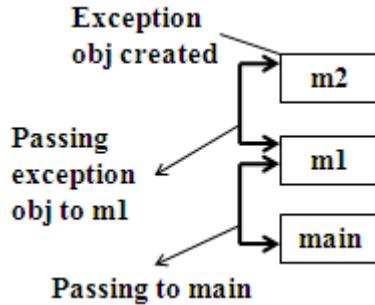
Output:

```

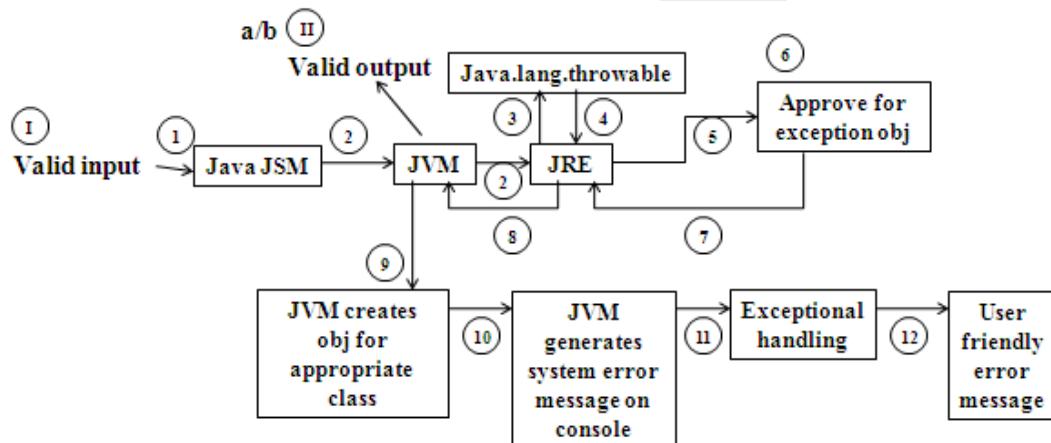
before calling
iam in m1
iam in m2
Exception in thread "main" java.lang.ArithmetricException: / by zero
    at CSEExp.m2(CSEExp.java:11)
    at CSEExp.m1(CSEExp.java:6)
    at CSEExp.main(CSEExp.java:17)

```





6. After a method throws an exception, the runtime system attempts to find something to handle it.
7. The set of possible something to handle the exception is the ordered list of methods that had been called to get to the method where the error had occurred. These list of methods is known as call stack
8. The runtime system searches call stack for a method that contains a block of code that can handle the exception. This block of code is called as Exception handler.
9. The search begins with the method in which the error had occurred and proceeds to call stack in the reverse order in which these methods are called.
10. When an appropriate handler is found, the runtime system passes the exception to the handler. If the type of the exception object thrown matches the type that can be handled by the handler.
11. Now the exception handler chosen is said to catch the exception. If the runtime system exhaustively searches all the methods in the call stack without appropriate exception handler the runtime system terminates abnormally.
12. The flow of exception creation of object is as follows.



I: User entered input values

II: JVM process the program with the valid input values to give the valid output.

Step 1: User entered invalid input values.

Step 2: JVM cannot process the program with invalid input values then contacts the JRE.

Step 3: JRE contacts the java.lang.Throwable for finding what type of exception has occurred in java program.

Step 4: java.lang.Throwable decides the type of exception which is occurred in the program & gives to JRE.

Step 5: JRE contacts java exception API.

Step 6: Exception API contains different kinds of exceptions.

Step 7: In java exception API gives appropriate exception for subclass to JRE.

Step 8: JRE gives a subclass exception to JVM.

Step 9: JVM creates an object for appropriate exception for subclass.

Step 10: JVM generates system error message.

Step 11: By using exception handling mechanisms we convert the system generated error message to user friendly error message.

PROGRAMMATICALLY DEFINATION OF EXCEPTION:

Exception is an object occurs at runtime which describes the system generated error which consists of name of exception, description, place where exception has created.

TYPES OF EXCEPTION:

Exceptions are categorized into 2 types.

- 1) Predefined or built-in exceptions.
- 2) User defined exceptions.

PREDEFINED EXCEPTIONS:

They are built-in exceptions which are developed by sun micro systems and supplied as a part of JDK to solve a universal problem.

Ex: division by zero, class not found, file not found, array index out of bounds.

USER DEFINED EXCEPTIONS:

These are the exceptions which are developed by java program and supplied as a part of the project that they are dealing with.

Ex: Invalid salary, invalid account no etc.

PREDEFINED EXCEPTIONS/BUILT-IN EXCEPTIONS(TYPES)

These are of 2 kinds.

- i. Asynchronous
- ii. Synchronous

ASYNCHRONOUS EXCEPTION:

These are the exceptions which are dealing with hardware problems & external problems.

Ex: power failure, memory problem.

Currently java program does not deals with asynchronous exception. In java a predefined super class for handling all the asynchronous exception is `java.lang.Error`.

SYNCHRONOUS EXCEPTION:

These are the exceptions are those which are always programmatic runtime errors and classified into 2.

- a. Checked exceptions
- b. Unchecked exceptions

CHECKED EXCEPTION:

These are the exceptions are those which are checked by the compiler for smooth execution of a program at runtime are called as checked exceptions and these are subclass of java.lang.Exception class.

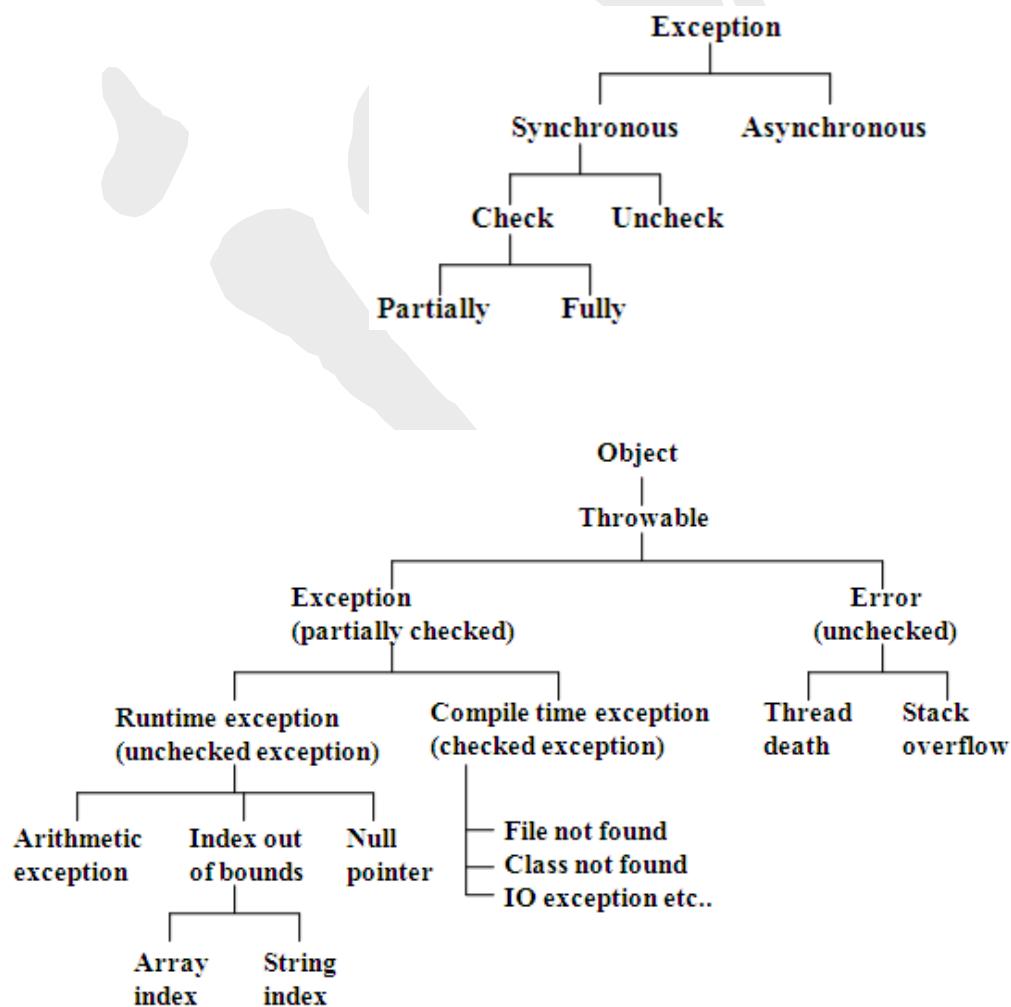
Ex: IO Exception, file not found exception, class not found exception, interrupted exception.

UNCHECKED EXCEPTION:

These are those exceptions which are not checked by the compiler and these are subclass of java.lang.runtime exceptions.

Ex: array index out of bounds, arithmetic exception, and number format exception.

Runtime exception is also a subclass of java.lang.Exception.



For all the exception classes the super class is **Exception**.

For all the exception & error is **Throwable**.

For universally the super class is **Object**.

Error is non-recoverable

Exception is recoverable by mechanism of exception handling

Checked exceptions are classified into 2.

- i. Fully checked exception
- ii. Partially checked exception

FULLY CHECKED EXCEPTION:

An exception class is said to be fully checked exception if its child classes are completely checked.

Ex: IO exception; java.lang.IOException.

PARTIALLY CHECKED EXCEPTION:

A class is said to be partially checked exception if its child classes are both combination of checked and unchecked exceptions.

Ex: java.lang.Exception.

Unchecked exceptions are java.lang.Runtime and Error

EXCEPTIONAL HANDLING MECHANISMS:

In order to handle the exceptions and for a graceful execution of a program we have 5 keywords.

1. Try
2. Catch
3. Finally
4. Throw
5. Throws

UNCHECKED EXCEPTION:

sno	Exception	Description
1.	Arithmetic exception	Arithmetic error such as divisible by zero
2.	Array index out of bounds exception	Accessing element out of array index
3.	Class cast exception	Invalid casting
4.	Illegal argument exception	Illegal arguments used to invoke a method
5.	Illegal thread state exception	Requested operation not compatible with current state
6.	Negative array size exception	It is due to array created with negative sign
7.	Null pointer exception	Invalid use of a null reference

8.	Number format exception	Invalid conversion of a string to numeric format
9.	String index out of bounds exception	Attempt to index out of bound to a string

UNCHECKED EXCEPTION:

Sno	Exception	Description
1	Class not found exception	Encountered when a class is not found
2	No such field exception	Occurs when a requested field does not exist
3	No such method exception	Occurs when a request method does not exist.
4	Illegal access exception	Occurs when an access to a class is defined
5	Instantaneous exception	Attempt to a create an object for an interface/an abstract class

Syntax:

```

try
{
    Code to run(risky code)
}

catch(exception_name reference_name)
{
    Code to run if an exception is raised
}

```

If we don't have try catch and if an exception is raised then the program goes for abnormal termination. In java lang we are handling exceptions by try catch blocks where try consists of risky code where an exception may be raised. Catch block consists of exception handling code. Catch block code is an alternative code for the exception code written in try. If an exception is raised in try block, the catch block code is executed or else and goes for normal termination of the program, and if exception is not raised and catch block is not executed and goes for normal termination.

Example program for try-catch:

```

class TryCatchEx
{
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=8/0;
            System.out.println("after exception");
        }
    }
}

```

```

        }

    catch(ArithmeticException e)
    {
        System.out.println("i'm in catch");
        e.printStackTrace();
    }

    System.out.println("program completed");
}

}

```

Output:

```

before exception
i'm in try
i'm in catch
java.lang.ArithmetricException: / by zero
    at TryCatchEx.main(TryCatchEx.java:9)
program completed

```

Message format printing:

Note 1: It is not possible to write only try. A try should be always associated with catch or finally.

Example:

```

class TryCatchEx1
{
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=8/0;
            System.out.println("after exception");
        }
        System.out.println("program completed");
    }
}

```

Ouput:

```
D:\z\exp>javac TryCatchEx1.java
TryCatchEx1.java:6: error: 'try' without 'catch', 'finally' or resource declarations
        try
        ^
1 error
```

Note 2: In between try and catch we should not write any executable statements.

Example:

```
class TryCatchEx2
{
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=5/0;
        }

        System.out.println("iam in between try and catch");
        catch(ArithmaticException e)
        {
            System.out.println("i'm in catch");
            System.out.println("divide by zero");
        }

        System.out.println("program completed");
    }
}
```

Output:

```
D:\z\exp>javac TryCatchEx2.java
TryCatchEx2.java:6: error: 'try' without 'catch', 'finally' or resource declarations
        try
        ^
TryCatchEx2.java:12: error: 'catch' without 'try'
                catch(ArithmetricException e)
                ^
TryCatchEx2.java:12: error: ')' expected
                catch(ArithmetricException e)
                ^
TryCatchEx2.java:12: error: not a statement
                catch(ArithmetricException e)
                ^
TryCatchEx2.java:12: error: ';' expected
                catch(ArithmetricException e)
                ^
5 errors
```

Note 3: Any statements which are written after exception occurring code in try block, those statements are not executed.

Example:

```
class TryCatchEx3
{
    public static void main(String args[])
    {
        System.out.println("before exception");
        try
        {
            System.out.println("i'm in try");
            int a=5/0;
            System.out.println("after exception");
        }
        catch(ArithmetricException e)
        {
            System.out.println("i'm in catch");
            System.out.println("divide by zero");
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before exception
i'm in try
i'm in catch
divide by zero
program completed
```

Note 4: In catch block we can write exception class name directly

Example:

```
class TryCatchEx4
{
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=5/0;
            System.out.println("after exception");
        }
        catch(Exception e)
        {
            System.out.println("i'm in catch");
            System.out.println("divide by zero");
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before exception
i'm in try
i'm in catch
divide by zero
program completed
```

Note 5: If there are n numbers of exceptions in try block & if we have n number of catch blocks for the n number of exceptions that are going to be raised in try. Then only the catch block of first exception raised in try block is going to be executed irrespective of the order of catch blocks.(same as switch case).

Example:

```
class TryCatchEx5
```

```

{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=5/0;
            System.out.println(s.length());
        }

        catch(NullPointerException ne)
        {
            System.out.println("i'm in catch1");
            System.out.println("null value");
        }

        catch(Exception e)
        {
            System.out.println("i'm in catch2");
            System.out.println("divide by zero");
        }

        System.out.println("program completed");
    }
}

```

Output:

```

before exception
i'm in try
i'm in catch2
divide by zero
program completed

```

Note 6: If we write the super class exception class first, in catch and then subclass of exception class next then we get compilation error.

Example:

```

class TryCatchEx6
{

```

```

static String s;

public static void main(String args[])
{
    System.out.println("before exception");

    try
    {
        System.out.println("i'm in try");

        int a=5/0;

        System.out.println(s.length());
    }

    catch(Exception ne)
    {
        System.out.println("i'm in catch1");

        System.out.println("null value");
    }

    catch(ArithmeticException e)
    {
        System.out.println("i'm in catch2");

        System.out.println("divide by zero");
    }

    System.out.println("program completed");
}

```

Output:

```

D:\z\exp>javac TryCatchEx6.java
TryCatchEx6.java:18: error: exception ArithmeticException has already been caught
        catch(ArithmeticException e)
                  ^
1 error

```

Note 7: In try catch first priority should be given to try and then for catch.

Example:

```

class TryCatchEx7
{
    public static void main(String args[])
    {

```

```
System.out.println("before exception");

catch(ArithmetricException e)

{

    System.out.println("i'm in catch2");

    System.out.println("divide by zero");

}

System.out.println("program completed");

try

{

    System.out.println("i'm in try");

    int a=5/0;

}

}

}
```

Output:

```
D:\z\exp>javac TryCatchEx7.java
TryCatchEx7.java:6: error: 'catch' without 'try'
        catch(ArithmetricException e)
               ^
TryCatchEx7.java:6: error: ')' expected
        catch(ArithmetricException e)
               ^
TryCatchEx7.java:6: error: not a statement
        catch(ArithmetricException e)
               ^
TryCatchEx7.java:6: error: ';' expected
        catch(ArithmetricException e)
               ^
TryCatchEx7.java:12: error: 'try' without 'catch', 'finally' or resource declarations
        try
               ^
5 errors
```

Exception message printing in java:

In java, in order to print exception message we have 3 approaches.

- 1) printStackTrace()
- 2) printing content from the object
- 3) getMessage

Example:

```
class ExpMsgPrintingEx
{
    public static void main(String args[])
    {
        System.out.println("before exception");

        try
        {
            System.out.println("i'm in try");
            int a=8/0;
        }
        catch(Exception e)
        {
            System.out.println("i'm in catch");
            System.out.println("divide by zero");
            System.out.println("*****");
            e.printStackTrace();
            System.out.println("*****");
            System.out.println(e);
            System.out.println("*****");
            System.out.println(e.toString());
            System.out.println("*****");
            System.out.println(e.getMessage());
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before exception
i'm in try
i'm in catch
divide by zero
*****
java.lang.ArithmmeticException: / by zero
    at ExpMsgPrintingEx.main(ExpMsgPrintingEx.java:9)
*****
java.lang.ArithmmeticException: / by zero
*****
java.lang.ArithmmeticException: / by zero
*****
/ by zero
program completed
```

- The three approaches for printing the exception message are defined in Throwable class

- 1) **printStackTrace():**

This method prints exception information in the following format.

Name of the exception: description
StackTrace();

- 2) **toString():**

This prints exception information in the following format

Name: description

- 3) **getMessage():**

It prints exception in the following format

Description

Possibilities of try-catch:

1. Single try catch:

Syntax:

```
Statement1;
try
{
    Statement2;
}
catch()
{
    Statement3;
}
Statement4;
```

Case 1: If exception is at statement1, program goes for abnormal termination.

Example:

```
class SingleTryCatch1
{
    public static void main(String args[])
    {
        System.out.println(10/0);
        System.out.println("before try");
    }
}
```

```
try
{
    System.out.println(10/0);
}
catch(Exception e)
{
    System.out.println("iam in actch");
}
System.out.println("program completed");
}
```

Output:

```
D:\z\exp>java SingleTryCatch1
Exception in thread "main" java.lang.ArithmetcException: / by zero
        at SingleTryCatch1.main(SingleTryCatch1.java:5)
```

Case 2: If exception is at statement2 and catch is matched. 1, 3, 4, normal termination

Example:

```
class SingleTryCatch2
{
    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println(10/0);
        }
        catch(Exception e)
        {
            System.out.println("divide by zero exception");
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before try
divide by zero exception
program completed
```

Case 3: If exception is at statement2 and catch is not matched. 1, normal termination

Example:

```
class SingleTryCatch3
{
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println(10/0);
        }
        catch(NullPointerException e)
        {
            System.out.println("null pointer exception");
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before try
Exception in thread "main" java.lang.ArithmaticException: / by zero
        at SingleTryCatch3.main(SingleTryCatch3.java:8)
```

Case 4: If exception is at statement4 then 1, 2, abnormal termination

Example:

```
class SingleTryCatch4
{
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println("iam in try");
        }
    }
}
```

```

        }
        catch(Exception e)
        {
            System.out.println("divide by zero exception");
        }
        System.out.println(10/0);
        System.out.println("program completed");
    }
}

```

Output:

```

before try
iam in try
Exception in thread "main" java.lang.ArithmetricException: / by zero
    at SingleTryCatch4.main(SingleTryCatch4.java:14)

```

2. Multiple try-catch:

Syntax:

```

Statement1;
try
{
    Statement2;
}
catch()
{
    Statement3;
}
try
{
    Statement4;
}
catch()
{
    Statement5;
}

Statement6;

```

Case 1: If exception is raised at statement1 then it will be abnormal termination

Example:

```

class MultipleTryCatch1
{

```

```

static String s;

public static void main(String args[])
{
    System.out.println(10/0);

    try
    {
        System.out.println(10/0);
    }

    catch(ArithmetricException e)
    {
        System.out.println("divide by zero exception");
    }

    try
    {
        System.out.println(s.length());
    }

    catch(NullPointerException ne)
    {
        System.out.println("null pointer exception");
    }

    System.out.println("program completed");
}

```

Output:

```

Exception in thread "main" java.lang.ArithmetricException: / by zero
at MultipleTryCatch1.main(MultipleTryCatch1.java:6)

```

Case 2: If exception is raised at statement2 corresponding catch is found immediately after try. Then 1, 3, 4, 6, normal termination

Example:

```

class MultipleTryCatch2
{
    static String s;

    public static void main(String args[])
    {

```

```

System.out.println("before try");

try
{
    System.out.println("iam in try1");
    System.out.println(10/0);
}

catch(ArithmeticException e)
{
    System.out.println("divide by zero exception");
}

try
{
    System.out.println("iam in try2");
}

catch(NullPointerException ne)
{
    System.out.println("null pointer exception");
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try1
divide by zero exception
iam in try2
program completed

```

Case 3: If exception is raised at statement2 and corresponding catch is not after try, but it is with second try then statement1 and normal termination

Example:

```

class MultipleTryCatch3
{
    static String s;

    public static void main(String args[])
    {

```

```

System.out.println("before try");

try
{
    System.out.println("iam in try1");
    System.out.println(10/0);
}

catch(NullPointerException ne)
{
    System.out.println("null pointer exception");
}

try
{
    System.out.println("iam in try2");
}

catch(ArithmeticException e)
{
    System.out.println("divide by zero exception");
}

System.out.println("program completed");
}

}

```

Output:

```

before try
iam in try1
Exception in thread "main" java.lang.ArithmetiException: / by zero
        at MultipleTryCatch3.main(MultipleTryCatch3.java:10)

```

Case 4: If exception is raised at statement3 then 1, 2, 4, 6 and normal termination

Example:

```

class MultipleTryCatch4

{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");
        try

```

```
{  
    System.out.println("iam in try1");  
}  
catch(NullPointerException ne)  
{  
    System.out.println(10/0);  
}  
try  
{  
    System.out.println("iam in try2");  
}  
catch(Exception e)  
{  
    System.out.println("iam in catch2");  
}  
System.out.println("program completed");  
}  
}
```

Output:

```
before try  
iam in try1  
iam in try2  
program completed
```

Case 5: If exception is raised at statement4 and corresponding catch is found immediately after try then 1, 2, 5, 6 and normal termination

Example:

```
class MultipleTryCatch5  
{  
    static String s;  
    public static void main(String args[])  
    {  
        System.out.println("before try");  
        try  
        {  
            System.out.println("iam in try1");  
        }
```

```

    }

    catch(ArithmeticException ae)
    {
        System.out.println("divide by zero");
    }

    try
    {
        System.out.println("iam in try2");

        System.out.println(s.length());
    }

    catch(NullPointerException ne)
    {
        System.out.println("null pointer exception");
    }

    System.out.println("program completed");
}

}

```

Output:

```

before try
iam in try1
iam in try2
null pointer exception
program completed

```

Case 6: If exception is raised at statement5 the statements 1, 2, 4, 6 and normal termination

Example:

```

class MultipleTryCatch6
{
    static String s;

    public static void main(String args[])
    {

        System.out.println("before try");

        try
        {
            System.out.println("iam in try1");
        }

```

```

        catch(ArithmeticException ae)
        {
            System.out.println("divide by zero");
        }
    try
    {
        System.out.println("iam in try2");
    }
    catch(NullPointerException ne)
    {
        System.out.println(s.length());
        System.out.println("null pointer exception");
    }
    System.out.println("program completed");
}

```

Output:

```
before try
iam in try1
iam in try2
program completed
```

Case 7: If exception is raised at statement4 and corresponding catch is above try the statements 1, 2 and abnormal termination

Example:

```

class MultipleTryCatch7
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try1");
        }
        catch(ArithmeticException ae)

```

```

    {
        System.out.println("divide by zero");
    }

    try
    {
        System.out.println("iam in try2");
        System.out.println(10/0);
    }

    catch(NullPointerException ne)
    {
        System.out.println("null pointer exception");
    }

    System.out.println("program completed");
}

}

```

Output:

```

before try
iam in try1
iam in try2
Exception in thread "main" java.lang.ArithmetricException: / by zero
    at MultipleTryCatch7.main(MultipleTryCatch7.java:18)

```

Case 8: If exception is raised at statement6 then statements 1, 2, 4 and abnormal termination

Example:

```

class MultipleTryCatch8
{
    static String s;

    public static void main(String args[])
    {

        System.out.println("before try");

        try
        {
            System.out.println("iam in try1");

        }

        catch(ArithmetricException ae)
        {

```

```

        System.out.println("divide by zero");

    }

    try

    {

        System.out.println("iam in try2");

    }

    catch(NullPointerException ne)

    {

        System.out.println("null pointer exception");

    }

    System.out.println(10/0);

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try1
iam in try2
Exception in thread "main" java.lang.ArithmetricException: / by zero
        at MultipleTryCatch8.main(MultipleTryCatch8.java:23)

```

Case 9: If exception is raised at statement2 and corresponding catch is found after try and exception is raised at statement4 corresponding catch is not found after try then statements 1, 2 and abnormal termination

Example:

```

class MultipleTryCatch9

{

    static String s;

    public static void main(String args[])

    {

        System.out.println("before try");

        try

        {

            System.out.println("iam in try1");

            System.out.println(10/0);

        }

        catch(ArithmetricException ae)

```

```

        {
            System.out.println("divide by zero");
        }
    try
    {
        System.out.println("iam in try2");
        System.out.println(s.length());
    }
    catch(ArrayIndexOutOfBoundsException ne)
    {
        System.out.println("array out of bounds exception");
    }
    System.out.println("program completed");
}

```

Output:

```

before try
iam in try1
divide by zero
iam in try2
Exception in thread "main" java.lang.NullPointerException
        at MultipleTryCatch9.main(MultipleTryCatch9.java:19)

```

Case 10: If exception is raised at statement2 and corresponding catch is not found and exception is raised at statement4 and corresponding catch is found then statements 1 and abnormal termination.

Example:

```

class MultipleTryCatch10
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println("iam in try1");
            System.out.println(10/0);
        }

```

```

        catch(ArrayIndexOutOfBoundsException ne)
        {
            System.out.println("array out of bounds exception");
        }
        try
        {
            System.out.println("iam in try2");
            System.out.println(s.length());
        }
        catch(NullPointerException ne)
        {
            System.out.println("null pointer exception");
        }
        System.out.println("program completed");
    }
}

```

Output:

```

before try
iam in try1
Exception in thread "main" java.lang.ArithmetricException: / by zero
        at MultipleTryCatch10.main(MultipleTryCatch10.java:10)

```

Case 11: If exception is raised at statement2 and corresponding catch is found immediately and exception raised at statement4 and corresponding catch is found the statements 1, 3, 5, 6 and normal termination

Example:

```

class MultipleTryCatch11
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println("iam in try1");
            System.out.println(10/0);
        }

```

```
        catch(ArithmeticException ae)
        {
            System.out.println("divide by zero exception");
        }
    try
    {
        System.out.println("iam in try2");
        System.out.println(s.length());
    }
    catch(NullPointerException ne)
    {
        System.out.println("null pointer exception");
    }
    System.out.println("program completed");
}
}
```

Output:

```
before try
iam in try1
divide by zero exception
iam in try2
null pointer exception
program completed
```

3. Try with multiple catches:

Syntax:

```
Statement1;  
try  
{  
    Statement2;  
}  
catch()  
{  
    Statement3;  
}  
catch()  
{  
    Statement4;  
}  
Statement5;
```

Case1: If exception is raised at statement1 then abnormal termination

Example:

```
class TryMultipleCatch1  
{  
    static String s;  
    public static void main(String args[])  
    {  
        System.out.println("before try");  
        System.out.println(10/0);  
        try  
        {  
            System.out.println("iam in try");  
        }  
        catch(ArithmaticException ae)  
        {  
            System.out.println("divide by zero exception");  
        }  
        catch(NullPointerException ne)  
        {  
            System.out.println("null pointer exception");  
        }  
        System.out.println("program completed");  
    }
```

```
}
```

Output:

```
before try
Exception in thread "main" java.lang.ArithmetricException: / by zero
        at TryMultipleCatch1.main(TryMultipleCatch1.java:7)
```

Case2: If exception is raised at statement2 corresponding catch is found immediately occur after try then 1, 3, 5 and normal termination

Example:

```
class TryMultipleCatch2
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println("iam in try");
            System.out.println(10/0);
        }
        catch(ArithmetricException ae)
        {
            System.out.println("divide by zero exception");
        }
        catch(NullPointerException ne)
        {
            System.out.println("null pointer exception");
        }
        System.out.println("program completed");
    }
}
```

Output:

```
before try
iam in try
divide by zero exception
program completed
```

Case3: If exception is raised at statement2 and corresponding catch is not found immediately but it is present in the program then statements 1, 4, 5 and normal termination

Example:

```
class TryMultipleCatch3
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");
            System.out.println(s.length());
        }

        catch(ArithmaticException ae)
        {
            System.out.println("divide by zero exception");
        }

        catch(NullPointerException ne)
        {
            System.out.println("null pointer exception");
        }

        System.out.println("program completed");
    }
}
```

Output:

```
before try
iam in try
null pointer exception
program completed
```

4. Nested try catch:

a. Try catch within try block

Syntax:

```
Statement1;
try
{
```

```
Statement2;
try
{
    Statement3;
}
catch()
{
    Statement4;
}
}
catch()
{
    Statement5;
}
Statement6;
```

Case1: If exception is raised at statement1 then program goes for abnormal termination

Example:

```
class TryCatchInTry1
{
    static String s;
    public static void main(String args[])
    {
        System.out.println("before try");
        System.out.println(10/0);
        try
        {
            System.out.println("iam in try1");
            try
            {
                System.out.println("iam in try in try");
                System.out.println(s.length());
            }
            catch(ArithmeticException ae)
            {
                System.out.println("divide by zero exception");
            }
        }
        catch(NullPointerException ne)
        {
    }}
```

```

        System.out.println("null pointer exception");

    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
Exception in thread "main" java.lang.ArithmetricException: / by zero
at TryCatchInTry1.main(TryCatchInTry1.java:7)

```

Case2: If exception is raised at statement2 and catch is found inside the try then statement1 and abnormal termination

Example:

```

class TryCatchInTry2

{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");
            System.out.println(10/0);

            try
            {
                System.out.println("iam in try in try");
                System.out.println(s.length());
            }
            catch(ArithmetricException ae)
            {
                System.out.println("divide by zero exception");
            }
        }
        catch(NullPointerException ne)
        {
}

```

```

        System.out.println("null pointer exception");

    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try
Exception in thread "main" java.lang.ArithmetricException: / by zero
at TryCatchInTry2.main(TryCatchInTry2.java:10)

```

Case3: If exception is raised at statement2 and catch is found inside the try and also at outside of try then statements 1, 5, 6 and normal termination

Example:

```

class TryCatchInTry3

{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            System.out.println(10/0);

            try
            {
                System.out.println("iam in try in try");

                System.out.println(s.length());
            }
            catch(ArithmetricException ae)
            {
                System.out.println("divide by zero exception1");
            }
        }
        catch(ArithmetricException be)
        {

```

```

        System.out.println("divide by zero exception2");

    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try
divide by zero exception2
program completed

```

Case4: If exception is raised at statement3 corresponding catch is found immediately after try (inside try) then statements 1, 2, 4, 6 and normal termination

Example:

```

class TryCatchInTry4

{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            try
            {
                System.out.println("iam in try in try");

                System.out.println(10/0);
            }
            catch(ArithmeticException ae)
            {
                System.out.println("divide by zero exception");
            }
        }
        catch(NullPointerException be)
        {
            System.out.println("null pointer exception");
        }
    }
}

```

```

    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try
iam in try in try
divide by zero exception
program completed

```

Case5: If exception is raised at statement4 then statements 1, 2, 3, 6 and normal termination

Example:

```

class TryCatchInTry5

{
    static String s;

    public static void main(String args[])
    {

        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            try
            {
                System.out.println("iam in try in try");

                }

                catch(ArithmaticException ae)
                {
                    System.out.println(10/0);

                    System.out.println("divide by zero exception");
                }
            }

            catch(NullPointerException be)
            {
                System.out.println("null pointer exception");
            }
        }
    }
}

```

```
        System.out.println("program completed");

    }

}
```

Output:

```
before try
iam in try
iam in try in try
program completed
```

Case6: If exception is raised at statement5 then statements 1, 2, 3, 6 and normal termination

Example:

```
class TryCatchInTry6

{

    static String s;

    public static void main(String args[])
    {

        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            try
            {
                System.out.println("iam in try in try");

                }
                catch(ArithmetricException ae)
                {
                    System.out.println("divide by zero exception");
                }
            }

        catch(NullPointerException be)
        {
            System.out.println(10/0);
            System.out.println("null pointer exception");
        }

        System.out.println("program completed");
    }
}
```

```
    }  
}
```

Output:

```
before try  
iam in try  
iam in try in try  
program completed
```

Case7: If exception is raised at statement6 then statements 1, 2, 3 and abnormal termination

Example:

```
class TryCatchInTry7  
{  
    static String s;  
    public static void main(String args[])  
    {  
        System.out.println("before try");  
        try  
        {  
            System.out.println("iam in try");  
            try  
            {  
                System.out.println("iam in try in try");  
            }  
            catch(ArithmaticException ae)  
            {  
                System.out.println("divide by zero exception");  
            }  
        }  
        catch(NullPointerException be)  
        {  
            System.out.println("null pointer exception");  
        }  
        System.out.println(10/0);  
        System.out.println("program completed");  
    }
```

```
}
```

Output:

```
before try
iam in try
iam in try in try
Exception in thread "main" java.lang.ArithmetricException: / by zero
    at TryCatchInTry7.main(TryCatchInTry7.java:23)
```

b. Try catch within catch block

Syntax:

```
Statement1;
try
{
    Statement2;
}
catch()
{
    Statement3;
    try
    {
        Statement4;
    }
    catch()
    {
        Statement5;
    }
}
Statement6;
```

Case1: If exception is raised at statement 1 then program goes for abnormal termination

Example:

```
class TryCatchInCatch1
{
    static String s;
    public static void main(String args[])
}
```

```

{
    System.out.println("before try");
    System.out.println(10/0);
    try
    {
        System.out.println("iam in try");
    }
    catch(ArithmaticException ae)
    {
        System.out.println("iam in catch");
        try
        {
            System.out.println("iam in try in catch block");
        }
        catch(NullPointerException be)
        {
            System.out.println("null pointer exception");
        }
    }
    System.out.println("program completed");
}

```

Output:

```

before try
Exception in thread "main" java.lang.ArithmaticException: / by zero
at TryCatchInCatch1.main(TryCatchInCatch1.java:7)

```

Case2: If exception is raised at statement2 corresponding catch is not found then statement1 and abnormal termination

Example:

```

class TryCatchInCatch2
{
    static String s;
    public static void main(String args[])
    {

```

```

System.out.println("before try");

try
{
    System.out.println("iam in try");
    System.out.println(s.length());
}

catch(ArithmetricException ae)
{
    System.out.println("iam in catch");

    try
    {
        System.out.println("iam in try in catch block");
    }

    catch(NullPointerException be)
    {
        System.out.println("null pointer exception");
    }
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try
Exception in thread "main" java.lang.NullPointerException
        at TryCatchInCatch2.main(TryCatchInCatch2.java:10)

```

Case3: If exception is raised at statement2 corresponding catch is found and there is no exception is raised in inside of catch block then statements 1, 3, 4, 6 and normal termination

Example:

```

class TryCatchInCatch3
{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

```

```

try
{
    System.out.println("iam in try");
    System.out.println(10/0);
}

catch(ArithmeticException ae)
{
    System.out.println("iam in catch");

    try
    {
        System.out.println("iam in try in catch block");
    }

    catch(NullPointerException be)
    {
        System.out.println("null pointer exception");
    }
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try
iam in catch
iam in try in catch block
program completed

```

Case4: If exception is raised at statement3 and corresponding catch is inside the catch block then statements 1, 2, 6 and normal termination

Example:

```

class TryCatchInCatch4
{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

```

```

try
{
    System.out.println("iam in try");

}
catch(ArithmaticException ae)
{
    System.out.println("iam in catch");

    System.out.println(s.length());

    try
    {
        System.out.println("iam in try in catch block");
    }
    catch(NullPointerException be)
    {
        System.out.println("null pointer exception");
    }
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try
program completed

```

Case5: If exception is raised at statement2 and corresponding catch is found and if exception is raised at statement3 and corresponding catch is found inside the catch block, then statement1 and abnormal termination

Example:

```

class TryCatchInCatch4

{
    static String s;

    public static void main(String args[])
    {
        System.out.println("before try");

        try

```

```

}

    System.out.println("iam in try");

    System.out.println(10/0);

}

catch(ArithmeticException ae)

{

    System.out.println("iam in catch");

    System.out.println(s.length());

    try

    {

        System.out.println("iam in try in catch block");

    }

    catch(NullPointerException be)

    {

        System.out.println("null pointer exception");

    }

}

System.out.println("program completed");

}

```

Output:

```

before try
iam in try
iam in catch
Exception in thread "main" java.lang.NullPointerException
        at TryCatchInCatch4.main(TryCatchInCatch4.java:15)

```

Case6: If exception is raised at statement2 and statement4 corresponding catches are found, then statements 1, 3, 5, 6 and normal termination

Example:

```

class TryCatchInCatch6

{

    static String s;

    public static void main(String args[])

    {

        System.out.println("before try");

```

```

try
{
    System.out.println("iam in try");
    System.out.println(10/0);
}

catch(ArithmeticException ae)
{
    System.out.println("divide by zero exception");
    System.out.println("iam in catch");

    try
    {
        System.out.println("iam in try in catch block");
        System.out.println(s.length());
    }

    catch(NullPointerException be)
    {
        System.out.println("null pointer exception");
    }
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try
divide by zero exception
iam in catch
iam in try in catch block
null pointer exception
program completed

```

Case7: If exception is raised at statement2 and statement4 and corresponding catch is not found for statement4, then statements 1, 3 and abnormal termination

Example:

```

class TryCatchInCatch7
{
    static String s;

```

```

public static void main(String args[])
{
    System.out.println("before try");

    try
    {
        System.out.println("iam in try");
        System.out.println(10/0);
    }

    catch(ArithmeticException ae)
    {

        System.out.println("divide by zero exception");
        System.out.println("iam in catch");

        try
        {
            System.out.println("iam in try in catch block");
            System.out.println(s.length());
        }

        catch(ArithmeticException be)
        {

            System.out.println("divide by zero exception");
        }
    }

    System.out.println("program completed");
}

```

Output:

```

before try
iam in try
divide by zero exception
iam in catch
iam in try in catch block
Exception in thread "main" java.lang.NullPointerException
        at TryCatchInCatch7.main(TryCatchInCatch7.java:19)

```

Note:

When we are writing exception in catch we have to follow the order of child to parent class i.e. ArithmeticException to Exception.

Example:

```
try
{
    System.out.println("exception raised");
}
catch(ArithmeticException ae)
{
}
catch(Exception e)
{
}
=>valid
```

```
try
{
    System.out.println("exception raised");
}
catch(Exception e)
{
}
catch(ArithmeticException ae)
{
}
=>invalid
```

Case8: If exception is raised at statement6 only then statements 1, 2 and abnormal termination

Example:

```
class TryCatchInCatch8
```

```
{
```

```
    static String s;
```

```
    public static void main(String args[])
    {
        System.out.println("before try");
        try
        {
            System.out.println("iam in try");
        }
        catch(ArithmeticException ae)
        {
            System.out.println("divide by zero exception");
            System.out.println("iam in catch");
            try
            {
                System.out.println("iam in try in catch block");
            }
            catch(ArithmeticException be)
            {
                System.out.println("divide by zero exception");
            }
        }
        System.out.println(10/0);
    }
}
```

```
        System.out.println("program completed");

    }

}
```

Output:

```
before try
iam in try
Exception in thread "main" java.lang.ArithmetricException: / by zero
        at TryCatchInCatch8.main(TryCatchInCatch8.java:24)
```

5. Try catch within both try and catch block:

Syntax:

```
Statement1;
try
{
    Statement2;
    try
    {
        Statement3;
    }
    catch()
    {
        Statement4;
    }
}
catch()
{
    Statement5;
    try
    {
        Statement6;
    }
    catch()
    {
        Statement7;
    }
}
Statement8;
```

FINALLY

It is never recommended to define cleanup code within try block. Because no guarantee of execution.

It is never recommended to define cleanup code within catch block. Because it won't be executed if there is no exception in try. Hence we require a place to maintain cleanup code it is should be executed with respective of whether exception is raised or not, exception raised and whether handle or not, such type of place to maintain cleanup process is finally.

The main purpose of finally block is to maintain cleanup code which should be executed always.

Syntax:

```
try
{
    Risky code;
}
catch()
{
    Exception handle;
}
finally
{
    Cleanup code;
}
```

Example 1:

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

        try
        {
            System.out.println("iam in try");
            int a=10/0;
        }
        catch(ArithmaticException ae)
        {
            System.out.println("catch block");
        }
        finally
        {
            System.out.println("finally block");
        }
    }
}
```

```
        System.out.println("program completed");

    }

}
```

Output:

```
before try
iam in try
catch block
finally block
program completed
```

Example 2:

```
//when exception is raised in try and corresponding catch is not found
```

```
class FinallyEx

{

    public static void main(String args[])

    {

        System.out.println("before try");

        try

        {

            System.out.println("iam in try");

            int a=10/0;

        }

        catch(NullPointerException ae)

        {

            System.out.println("catch block");

        }

        finally

        {

            System.out.println("finally block");

        }

        System.out.println("program completed");

    }

}
```

Output:

```
before try
iam in try
finally block
Exception in thread "main" java.lang.ArithmetricException: / by zero
at FinallyEx.main(FinallyEx.java:9)
```

Syntax for try-catch-finally in try block:

```
Statement1
try
{
    Statement2
    try
    {
        Statement3;
    }
    catch()
    {
        Statement4;
    }
    finally
    {
        Statement5;
    }
}
catch()
{
    Statement6;
}
finally
{
    Statement7;
}
Statement8;
```

Case1: If exception is raised at statement1, then program goes for abnormal termination

Example:

```
class FinallyEx1
{
    public static void main(String args[])
    {
        System.out.println("before try");

        int a=10/0;

        try
        {
            System.out.println("iam in try");
        }
        catch(ArithmetricException e)
        {
            System.out.println("iam in catch");
        }
        finally
        {
            System.out.println("iam in finally");
        }
    }
}
```

```

try
{
    System.out.println("inside try block");

}
catch(ArithmaticException ae)
{
    System.out.println("catch in try block");

}
finally
{
    System.out.println("finally in try block");
}

}
catch(ArithmaticException be)
{
    System.out.println("catch block");

}
finally
{
    System.out.println("finally block");
}

System.out.println("program completed");
}

```

Output:

```

before try
Exception in thread "main" java.lang.ArithmaticException: / by zero
        at FinallyEx1.main(FinallyEx1.java:6)

```

Case2: If exception is raised at statement2 and corresponding catch is not found, then statements 1, 5, 7 and normal termination

Example:

```

class FinallyEx2
{
    public static void main(String args[])

```

```

{
    System.out.println("before try");

    try
    {
        System.out.println("iam in try");

        int a=10/0;

        try
        {
            System.out.println("inside try block");
        }

        catch(ArithmaticException ae)
        {
            System.out.println("catch in try block");
        }

        finally
        {
            System.out.println("finally in try block");
        }
    }

    catch(NullPointerException be)
    {
        System.out.println("catch block");
    }

    finally
    {
        System.out.println("finally block");
    }
}

System.out.println("program completed");
}

```

Output:

```

before try
iam in try
finally block
Exception in thread "main" java.lang.ArithmaticException: / by zero
at FinallyEx2.main(FinallyEx2.java:9)

```

Case3: If exception is raised at statement3 corresponding catch is found immediately. Then statements 1, 2, 5, 6, 7 and normal termination

Example:

```
class FinallyEx3
{
    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            try
            {
                System.out.println("inside try block");
                int a=10/0;
            }
            catch(ArithmaticException ae)
            {
                System.out.println("catch in try block");
            }
            finally
            {
                System.out.println("finally in try block");
            }
        }
        catch(NullPointerException be)
        {
            System.out.println("catch block");
        }
        finally
        {
            System.out.println("finally block");
        }
        System.out.println("program completed");
    }
}
```

```
    }  
}  
}
```

Output:

```
before try  
iam in try  
inside try block  
catch in try block  
finally in try block  
finally block  
program completed
```

Case4: If exception is raised at statement4, then statements 1, 2, 3, 5, 7 and normal termination

Example:

```
class FinallyEx4  
{  
    public static void main(String args[])  
    {  
        System.out.println("before try");  
  
        try  
        {  
            System.out.println("iam in try");  
  
            try  
            {  
                System.out.println("inside try block");  
            }  
            catch(ArithmeticException ae)  
            {  
                System.out.println("catch in try block");  
                int a=10/0;  
            }  
            finally  
            {  
                System.out.println("finally in try block");  
            }  
        }  
        catch(NullPointerException be)  
        {  
            System.out.println("catch block");  
        }  
    }  
}
```

```

        }

    finally
    {
        System.out.println("finally block");
    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try
inside try block
finally in try block
finally block
program completed

```

Case5: If exception raised at statement5, then statements 1, 2, 3, 5, 7 with an exception in statement 5

Example:

```

class FinallyEx5
{
    public static void main(String args[])
    {
        System.out.println("before try");

        try
        {
            System.out.println("iam in try");

            try
            {
                System.out.println("inside try block");
            }
            catch(ArithmetricException ae)
            {
                System.out.println("catch in try block");
            }
        finally
        {

```

```
        System.out.println("finally in try block");

        int a=10/0;

    }

}

catch(NullPointerException be)

{

    System.out.println("catch block");

}

finally

{

    System.out.println("finally block");

}

System.out.println("program completed");

}

}
```

Output:

```
before try
iam in try
inside try block
finally in try block
finally block
Exception in thread "main" java.lang.ArithmetricException: / by zero
        at FinallyEx5.main(FinallyEx5.java:20)
```

Note1: In try-catch-finally, try cannot be without catch/finally that is try has to be associated with catch/finally.

Example:

```
class FinallyNote1

{

    public static void main(String args[])

    {

        System.out.println("before try");

        try

        {
```

```
        System.out.println("iam in try");

        int a=10/0;

    }

    catch(ArithmeticException be)

    {

        System.out.println("divide by zero");

    }

    System.out.println("program completed");

}

}
```

Output:

```
before try
iam in try
divide by zero
program completed
```

Note2: In try-catch-finally, the order is most important that is we have to write try→catch→finally only. If we don't follow this order we will get compilation error

Example:

```
class FinallyNote2

{

    public static void main(String args[])

    {

        try

        {

            System.out.println("iam in try");

            int a=10/0;

        }

        finally

        {

            System.out.println("finally block");

        }

        catch(ArithmeticException be)

        {

            System.out.println("divide by zero");

        }

    }

}
```

```
        System.out.println("program completed");

    }

}
```

Output:

```
D:\z\exp>javac FinallyNote2.java
FinallyNote2.java:14: error: 'catch' without 'try'
        catch(ArithmetricException be)
                  ^
FinallyNote2.java:14: error: ')' expected
        catch(ArithmetricException be)
                  ^
FinallyNote2.java:14: error: not a statement
        catch(ArithmetricException be)
                  ^
FinallyNote2.java:14: error: ';' expected
        catch(ArithmetricException be)
                  ^
4 errors
```

Note3: As we have try-catch and try-catch in catch. We can have try-catch-finally in try,try-catch-finally in catch and try-catch-finally in finally.

Example:

```
class FinallyNote3

{
    public static void main(String args[])
    {
        try
        {
            System.out.println("iam in try");
            int a=10/0;
        }
        catch(ArithmetricException be)
        {
            System.out.println("divide by zero");
        }
        finally
        {
            System.out.println("finally block");
            try
            {
                System.out.println("try in finally block");
            }
        }
    }
}
```

```

        }

        catch(ArithmetricException be)

        {

            System.out.println("catch in finally block");

        }

        finally

        {

            System.out.println("finally in finally block");

        }

    }

    System.out.println("program completed");
}

```

Output:

```

iam in try
divide by zero
finally block
try in finally block
finally in finally block
program completed

```

Note4: If we have exception statement in try-catch-finally of inside try which are return a inside try and there is no corresponding catch, then that exception object raised in inside try or catch or finally will be given to outside try which interns checks for the respective catch block after the outside try.

Example:

```

class FinallyNote4

{

    static String s;

    public static void main(String args[])

    {

        System.out.println("before try");

        try

        {

            System.out.println("iam in try");

            try

            {

                System.out.println("try in try block");

```

```

        }

        catch(ArithmeticException be)

        {

            System.out.println("divide by zero");

        }

        finally

        {

            System.out.println("finally in try block");

            System.out.println(s.length());

        }

    }

    catch(ArithmeticException be)

    {

        System.out.println("divide by zero");

    }

    catch(NullPointerException be)

    {

        System.out.println("null pointer exception");

    }

    System.out.println("program completed");

}

}

```

Output:

```

before try
iam in try
try in try block
finally in try block
null pointer exception
program completed

```

Note5: A finally statement should not consist of any statements that rises exceptions because the exceptions raised in finally cannot be using catch (if finally is the last statement in try-catch block)

Example:

```

class FinallyNote5

{

    static String s;

    public static void main(String args[])

```

```
{  
    System.out.println("before try");  
    try  
    {  
        System.out.println("iam in try");  
    }  
    catch(ArithmetricException be)  
    {  
        System.out.println("divide by zero");  
    }  
    finally  
    {  
        System.out.println(10/0);  
    }  
    System.out.println("program completed");  
}  
}
```

Output:

```
before try  
iam in try  
Exception in thread "main" java.lang.ArithmetricException: / by zero  
at FinallyNote5.main(FinallyNote5.java:17)
```

Note6: In order to skip finally statement there is only one way to write “System.exit(0);”

To include “System.exit(0);” in try block.

Example:

```
class FinallyNote6  
{  
    public static void main(String args[])  
    {  
        System.out.println("before try");  
        try  
        {  
            System.out.println("iam in try");  
            System.exit(0);  
        }
```

```
        catch(ArithmeticException be)
        {
            System.out.println("divide by zero");
        }
    finally
    {
        System.out.println("finally block");
    }
    System.out.println("program completed");
}
}
```

Output:

```
before try
iam in try
```

**

DIFFERENCE BETWEEN FINAL, FINALLY, FINALIZE:

- **Final:**

final is a keyword which can be used to declare anything as constant. Final keyword is utilized in following ways.

1. Final variable → not to change its value (making variable as constant).
2. Final method → not to override the implementation.
3. Final class → not to extend.

- **Finally:**

Finally is a clause in try-catch syntax which can be used to execute a block of instructions irrespective of getting exception in try block, irrespective of getting exception in catch block.

- **Finalize:**

Finalize is a method of object class which is used for making an object eligible for garbage collection.

THROW:

- i. A throw keyword is that which is used to create an exception object explicitly either for predefined or user defined
- ii. In general java applications we get exceptions automatically when we have exception situations in java app's
- iii. If we want raise an exception explicitly we have to use throw keyword to raise an exception explicitly by a programmer. We have to use following syntax.
Syntax:

```
throw new Exception_class([parameter_list]);
```
- iv. A throw keyword is just like a try block. The difference is the try block automatically finds the solution and creates an exception implicitly.

Example:

```
class ThrowEx
{
    public static void main(String args[])
    {
        try
        {
            System.out.println(10/0);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        System.out.println("program completed");
    }
}
```

Output:

```
java.lang.ArithmaticException: / by zero
program completed
```

- v. In the above program when an exception is raised in try block. The main method is responsible for creating exception object which is created implicitly and programmer is not responsible for this exception object.

Example:

```
class ThrowEx
{
    public static void main(String args[])
    {
        throw new ArithmaticException("this is exception raised by programmer");
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmaticException: this is exception raised by programmer
at ThrowEx.main(ThrowEx.java:5)
```

- vi. In the above program, the programmer is creating exception object explicitly. The main method is not responsible for creating exception object.

Example:

```
class ThrowEx
{
    public static void main(String args[])
    {
        String accno="232";
        int pinno=Integer.parseInt(args[0]);
    }
}
```

```

        System.out.println("accno="+accno);
        System.out.println("pinno="+pinno);
        if((pinno<100011)&&(pinno>9999))
        {
            System.out.println("transaction is failed");
            throw new ArithmeticException("this is exception raised by programmer");
        }
        else
        {
            System.out.println("transaction is success");
        }
    }
}

```

Output: 1

```
D:\z\exp>java ThrowEx 100
accno=232
pinno=100
transaction is success
```

Output: 2

```
D:\z\exp>java ThrowEx 100000
accno=232
pinno=100000
transaction is failed
Exception in thread "main" java.lang.ArithmeticException: this is exception raised by programmer
    at ThrowEx.main(ThrowEx.java:12)
```

Note: A throw can be used in try & catch

- vii. In a program, if we provide any instruction or statements immediately after throw keyword then we get unreachable statement.

THROWS:

- i. A throws keyword is that which is used to create exception object explicitly as required by the programmer or user
- ii. The two ways to handle exception as
 - 1) Try catch finally
 - 2) By throws keyword
- iii. Throws keyword is that which is used to bypass the exception
- iv. In general, the throw keyword is used in method body but throws keyword is used in method prototype
- v. Throws keyword will allow the exception name, it should be either same as the generated exception or super class exception of generated exception but it should not of subclass exception for generated exception.

Example:

```

class A
{
    void m1()throws Exception
    {
        m2();
    }
    void m2()throws ArithmeticException
    {
        throw new ArithmeticException(" ");
    }
}
class ThrowEx
```

```

{
    public static void main(String args[])throws Throwable
    {
        A a=new A();
        a.m1();
        System.out.println("hello");
    }
}

```

Output:

```

Exception in thread "main" java.lang.ArithmetcException:
    at A.m2(ThrowEx.java:9)
    at A.m1(ThrowEx.java:5)
    at ThrowEx.main(ThrowEx.java:17)

```

- vi. In above program, if we compile, the compiler is able to identify ArithmeticException at line 9 due to throws keyword in m2() method prototype. The ArithmeticException will bypass to m1() method prototype at line 5, due to throws keyword at main() method prototype at line 17, the ArithmeticException is converted into exception and it will be bypass to m1() method call & then as throws keyword in main() method, the exception raised will be converted to throwable and it will bypass to caller of the main() method by JVM.
- vii. When we execute the above program, the JVM goes to display the ArithmeticException at three locations 9, 5, 17.

CUSTOMIZED EXCEPTION OR USER EXCEPTION:

Based on the user requirement, the user can create an exception called as user defined exception. As per the application requirement

Example: invalid pinno, invalid age...

STEPS TO CREATE USER DEFINED EXCEPTION:

1. To create user defined exception, we have to create user defined class i.e. subclass to the exception class.
2. To create user defined exception, define a custom exception class which has to be extend from extended class.

```

class InvalidException extends Exception
{
}

```

3. Declare an string parameterized constructor and access the super class i.e. exception class string parameterized constructor by using super keyword.

Multithreading in Java

- Multithreading in java is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in games, animation etc.

Advantages of Java Multithreading:

1. It doesn't block the user because threads are independent and you can perform multiple operations at same time.
2. You can perform many operations together so it saves time.
3. Threads are independent so it doesn't affect other threads if exception occur in a single thread.

Multitasking

- Multitasking is a process of executing multiple tasks simultaneously.
 - We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:
Process-based Multitasking(Multiprocessing)
Thread-based Multitasking(Multithreading)
1. **Process-based Multitasking (Multiprocessing)**
Each process have its own address in memory i.e. each process allocates separate memory area.
Process is heavyweight.
Cost of communication between the process is high.
Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.
 2. **Thread-based Multitasking (Multithreading)**
Threads share the same address space.
Thread is lightweight.
Cost of communication between the thread is low.

Note: At least one process is required for each thread.

DIFFERENCE BETWEEN PROCESS BASED & THREAD BASED APPLICATIONS

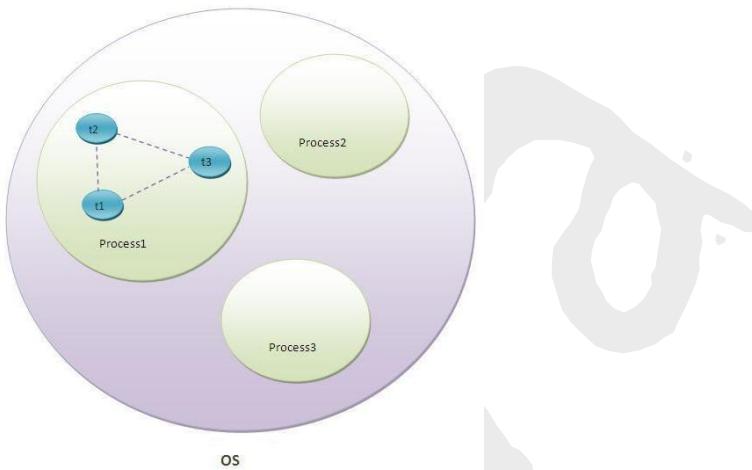
PROCESS BASED	THREAD BASED
Contains single flow of control	Contains multiple flow of control
All c, c++, cobol are process based	Java & .net are thread based
Context switching time is more in process based applications	Context switching time is less in thread based applications
In c, c++..etc for each and every sub program there exists a separate address page space	for each and every sub program there exists a single address space
All process based are treated as heavy weight	All thread based are treated as light weight
Process based applications are provide only sequential flow of execution but not concurrent flow of execution	Thread based applications provides both sequential and concurrent flow of execution

ADVANTAGES OF MULTITHREADING:

1. Perform multiple operations at a time, saves system resources.
2. As the threads are independent with each other if an exception occurs at one thread it will not affect the execution of other thread.

Thread:

- A thread is a flow of execution to accomplish a particular task.
- A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.
- Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.



- As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.
- As shown in the above figure, thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS and one process can have multiple threads.

Note: At a time one thread is executed only.

- Thread is classified into 2 types
 1. Single thread
 2. Multi thread

CREATION OF THREAD:

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

THREAD CLASS:

Java multithreading is dependent upon a class known as Thread which provides constructors and methods to create and perform operations on a thread.

Class declaration:

Thread extends object implements Runnable

Commonly used Constructors of Thread class:

```
Thread()  
Thread(String_name)  
Thread(Runnable r)  
Thread(Runnable r,String name)  
Thread(Thread group,String name)  
Thread(thread_group group,runnable target)  
Thread(thread_group group,runnable target,String name)
```

Commonly used methods of Thread class:

Sno	Methods	Description
1	public void run()	It is used to perform action for a thread.
2	public void start()	starts the execution of the thread.JVM calls the run() method on the thread.
3	public void sleep(long miliseconds)	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4	public void join()	waits for a thread to die.
5	public void join(long miliseconds)	waits for a thread to die for the specified miliseconds.
6	public int getPriority()	returns the priority of the thread.
7	public int setPriority(int priority)	changes the priority of the thread.
8	public String getName()	returns the name of the thread.
9	public void setName(String name)	changes the name of the thread.
10	public Thread currentThread()	returns the reference of currently executing thread.
11	public int getId()	returns the id of the thread.
12	public Thread.State getState()	returns the state of the thread.
13	public boolean isAlive()	tests if the thread is alive.
14	public void yield()	causes the currently executing thread object to temporarily pause and allow other threads to execute.
15	public void suspend()	is used to suspend the thread(deprecated).
16	public void resume()	is used to resume the suspended thread(deprecated).
17	public void stop()	is used to stop the thread(deprecated).
18	public boolean isDaemon()	tests if the thread is a daemon thread.
19	public void setDaemon(boolean b)	marks the thread as daemon or user thread.
20	public void interrupt()	interrupts the thread.
21	public boolean isInterrupted()	tests if the thread has been interrupted.
22	public static boolean interrupted()	tests if the current thread has been interrupted.

The previous methods are involved on a particular thread object the following methods in the thread class are static.

Life cycle of a Thread (Thread States)

A thread can be in several states depending upon the states of execution.

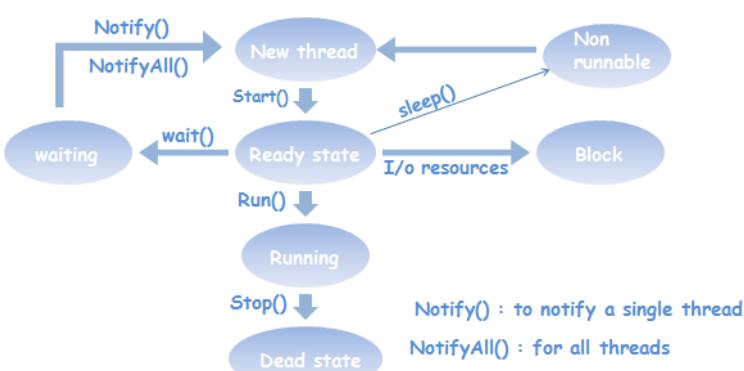
When you are executing a thread. A thread is going to be alive and it is any one of these states

1. New
2. Ready(Runnable)
3. Running
4. Non-Runnable (Blocked/sleeping/writing)
5. Dead(Terminated)

According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:



Thread life cycle in java

1. New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method. A new thread begins its life in a new state. It remains in this state until the program starts the state. It is also referred to as born thread.

Ex: A a=new A();
B b=new B();

2. Ready(Runnable):

After invocation of start method on new thread the thread becomes runnable.

After newly born thread is started and once it becomes runnable, this is considered for executing its task.

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

Ex: a.start();
b.start();

3. Blocked state:

Threads execution is stopped for i/o resources.

4. Dead state:

Execution of thread is explicitly terminated by stop().

5. Wait:

The thread is stopped temporarily for other thread to be executed until notify() or notifyall() are called first thread.

6. Sleep:

The thread is stopped temporarily for a specified period of time

7. Running state:

CPU is allocated and thread is in execution to accomplish its task.

Example program:

```
class Text
{
    public static void main(String args[])
    {
        A a=new A();
        a.start();
        B b=new B();
        b.start();
    }
}
class A extends Thread
{
    public void run()
    {
        for(int i=0;i<50;i++)
            System.out.println("hai A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int i=0;i<50;i++)
            System.out.println("hai B");
```

```
    }  
}
```

Steps for creation of thread

1. Executing the java.lang.Thread class
2. Create class for implementing java.lang.Runnable interface

By extending Thread class

Step 1: create a user defined thread class that extends the tread class and override the run()

```
class A extends Thread  
{  
    public void run()  
    {  
        for(int i=0;i<50;i++)  
            System.out.println("hai A");  
    }  
}
```

Step 2: create thread object

```
class Demo  
{  
    public static void main(String args[])  
    {  
        A a=new A();  
    }  
}
```

Step 3: start the execution of thread

```
class Demo  
{  
    public static void main(String args[])  
    {  
        a.start();  
    }  
}
```

Whenever we call start() the JVM searches for method in a class but the start() is not in A class. JVM searches for start() in parent class Thread class

As start() is present in the Thread class, as part of start() execution the run() is called by start()
Therefore whenever we call start(), run() is automatically executed

```
class Multi extends Thread  
{  
    public void run()  
    {  
        System.out.println("thread is running...");  
    }  
    public static void main(String args[])  
    {  
        Multi t1=new Multi();  
        t1.start();  
    }  
}
```

Output:thread is running...

By implementing Runnable interface

Step 1: create a class that implements runnable interface

```
class A implements Runnable
{
    public void run()
    {
        for(int i=0;i<50;i++)
            System.out.println("hai A");
    }
}
```

Step 2: create an object for the class

```
class Demo
{
    public static void main(String args[])
    {
        A a=new A();
    }
}
```

Step 3: create thread class object

```
class Demo
{
    public static void main(String args[])
    {
        A a=new A();
        Thread t=new Thread(a);
    }
}
```

Step 4: start the execution of thread

```
class Demo
{
    public static void main(String args[])
    {
        A a=new A();
        Thread t=new Thread(a);
        t.start();
    }
}
```

Example program

```
class Multi3 implements Runnable
{
    public void run()
    {
        System.out.println("thread is running...");
    }
    public static void main(String args[])
    {
        Multi3 m1=new Multi3();
        Thread t1 =new Thread(m1);
        t1.start();
    }
}
```

```
}
```

Output:thread is running...

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitly create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

Difference between t.start() and t.run()

t.start()	t.run()
t.start() indicated the thread class start is to be executed when a new thread is created which is responsible for run()	t.run() indicated a normal method execution flow i.e., no new thread will be created it is executed like a normal method

Thread controls:

Can java provides complete control over multithreading program. You can develop a multi threaded program which can be suspended, resume or stopped completely based on your requirements. There are the various static methods which you can use on thread object control their behaviour. Following table lists down these methods.

Sno	Method	Description
1	public void suspend()	This method puts a thread in the suspended state and can be resumed using resume()
2	public void stop()	This method stops a thread completely
3	public void resume()	This method resumes a thread which was suspended using suspend()
4	public void wait()	Causes the current thread to wait until another thread invokes the notify()
5	public void notify()	Wakes up a single thread that is waiting on this objects monitor

Naming Thread and Current Thread

Naming Thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

- **public String getName():** is used to return the name of a thread.
- **public void setName(String name):** is used to change the name of a thread.

Example of naming a thread

```
class TestMultiNaming1 extends Thread
{
    public void run()
    {
        System.out.println("running...");
    }
    public static void main(String args[])
    {
        TestMultiNaming1 t1=new TestMultiNaming1();
        TestMultiNaming1 t2=new TestMultiNaming1();
        System.out.println("Name of t1:"+t1.getName());
        System.out.println("Name of t2:"+t2.getName());
        t1.start();
        t2.start();
        t1.setName("Sonoo Jaiswal");
        System.out.println("After changing name of t1:"+t1.getName());
    }
}
```

Output:

```
Name of t1:Thread-0
Name of t2:Thread-1
id of t1:8
running...
After changeling name of t1:Sonoo Jaiswal
running...
```

Current Thread

The currentThread() method returns a reference of currently executing thread.

- **public static Thread currentThread()**

Example of currentThread() method

```
class TestMultiNaming2 extends Thread
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String args[])
}
```

```
{  
    TestMultiNaming2 t1=new TestMultiNaming2();  
    TestMultiNaming2 t2=new TestMultiNaming2();  
    t1.start();  
    t2.start();  
}  
}  
Output: Thread-0  
Thread-1
```

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is

NORM_PRIORITY=5.

MIN_PRIORITY is 1.

MAX_PRIORITY is 10.

Thread class defines 2 methods to get & set priority

- public final int getPriority();
- public void setPriority(int priority);

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread
{
    public void run()
    {
        System.out.println("running thread name is:" + Thread.currentThread().getName());
        System.out.println("running thread priority is:" + Thread.currentThread().getPriority());
    }
    public static void main(String args[])
    {
        TestMultiPriority1 m1=new TestMultiPriority1();
        TestMultiPriority1 m2=new TestMultiPriority1();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

Output:running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1

Sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

Syntax of sleep() method in java

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long milliseconds) throws InterruptedException
- public static void sleep(long milliseconds, int nanos) throws InterruptedException

Example of sleep method in java

```
class TestSleepMethod1 extends Thread
{
    public void run()
    {
        for(int i=1;i<5;i++)
        {
            try
            {
                Thread.sleep(500);
            }
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        TestSleepMethod1 t1=new TestSleepMethod1();
        TestSleepMethod1 t2=new TestSleepMethod1();
        t1.start();
        t2.start();
    }
}
```

Output:

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

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

IsAlive() method in java thread class

```
class Text
{
    public static void main(String args[])
    {
        A a=new A();
        System.out.println(a.isAlive());
        a.start();
        System.out.println(a.isAlive());
    }
}

class A extends Thread
{
    public void run()
    {
        System.out.println("adi");
    }
}
```

Output:

```
false
adi
true
```

ActiveCount() method in java thread class

This method is used to find number of threads are in active.

```
class A extends Thread
{
    public void run()
    {
        System.out.println("adi");
    }
}

class Text
{
    public static void main(String args[])
    {
        A a=new A();
        A a1=new A();
        A a2=new A();

        a.start();
        a1.start();
        a2.start();
        System.out.println(Thread.activeCount());
    }
}
```

Output:

```
adi
adi
adi
```

getName(), setName(String) and getId() method:

```
public String getName()
public void setName(String name)
public long getId()

class TestJoinMethod3 extends Thread{
    public void run(){
        System.out.println("running...");
    }
    public static void main(String args[]){
        TestJoinMethod3 t1=new TestJoinMethod3();
        TestJoinMethod3 t2=new TestJoinMethod3();
        System.out.println("Name of t1:"+t1.getName());
        System.out.println("Name of t2:"+t2.getName());
        System.out.println("id of t1:"+t1.getId());

        t1.start();
        t2.start();

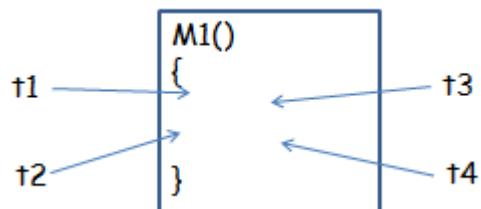
        t1.setName("Sonoo Jaiswal");
        System.out.println("After changing name of t1:"+t1.getName());
    }
}

Test it Now
Output:Name of t1:Thread-0
Name of t2:Thread-1
id of t1:8
running...
After changling name of t1:Sonoo Jaiswal
running...
```

Synchronization in Java

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- The synchronization is mainly used to
 1. To prevent thread interference.
 2. To prevent consistency problem.

Example:



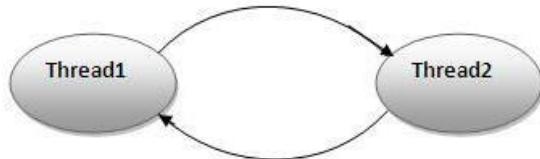
In the above case threads t1, t2, t3, t4 accessing the same method m1() simultaneously which gives the data inconsistency problem.

```
class A
{
    Synchronized void m1()
    {
        t1;
        t2;
        t3;
        t4;
    }
}
```

In the above case only one thread is able to access m1() to decrease the data inconsistency

Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



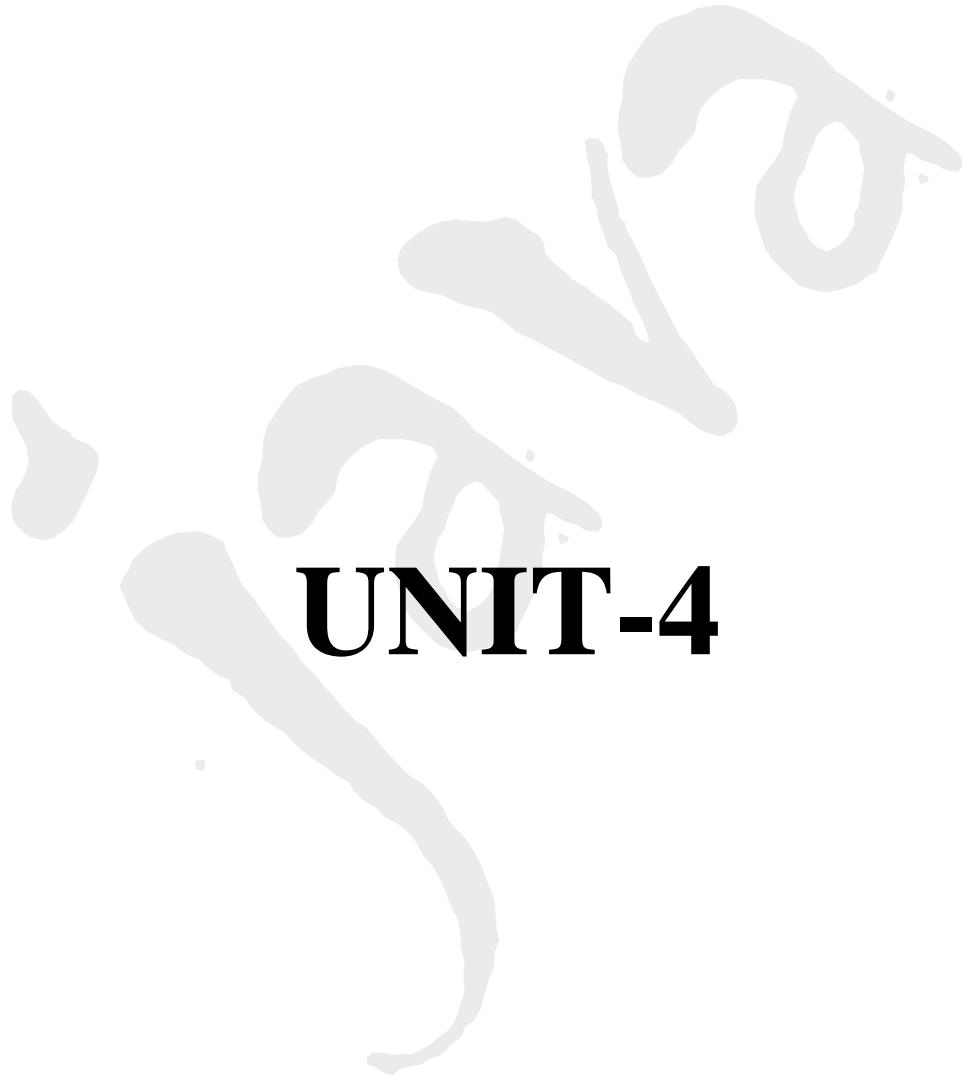
Deadlock in java

Example of Deadlock in java

```
public class TestDeadlockExample1
{
    public static void main(String[] args)
    {
        final String resource1 = "ratan jaiswal";
        final String resource2 = "vimal jaiswal";
        // t1 tries to lock resource1 then resource2
        Thread t1 = new Thread()
        {
            public void run()
            {
                synchronized (resource1)
                {
                    System.out.println("Thread 1: locked resource 1");
                    try
                    {
                        Thread.sleep(100);
                    }
                    catch (Exception e)
                    {
                    }
                    synchronized (resource2)
                    {
                        System.out.println("Thread 1: locked resource 2");
                    }
                }
            }
        };
        // t2 tries to lock resource2 then resource1
        Thread t2 = new Thread()
        {
            public void run()
            {
                synchronized (resource2)
```

```
{  
    System.out.println("Thread 2: locked resource 2");  
    try  
    {  
        Thread.sleep(100);  
    }  
    catch (Exception e)  
    {  
    }  
    synchronized (resource1)  
    {  
        System.out.println("Thread 2: locked resource 1");  
    }  
}  
}  
};  
t1.start();  
t2.start();  
}  
}
```

Output: Thread 1: locked resource 1
Thread 2: locked resource 2



UNIT-4

JDBC

1. JDBC-ODBC bridge driver
2. Native – API driver (partially java driver)
3. Network Protocol driver(fully java driver)
4. Thin driver(fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to THE database. The JDBC-ODBC bridge driver converts JDBC methods calls into the ODBC function calls. This is now discouraged because of thin driver.

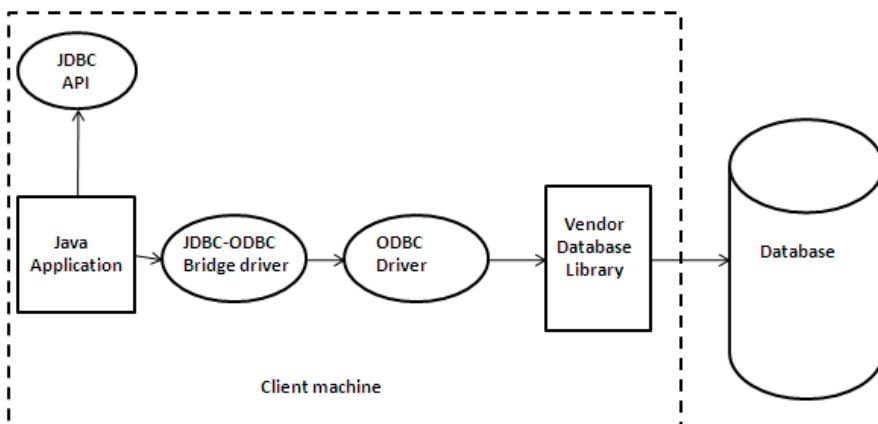


Figure- JDBC-ODBC Bridge Driver

Advantages:

- Easy to use.
- Can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native – API driver (partially java driver)

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into the native calls of the database API. It is not written entirely in java.

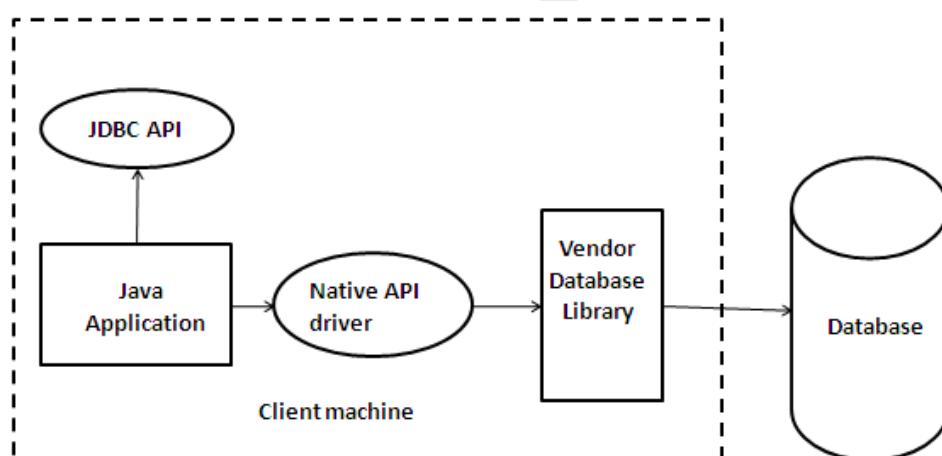


Figure- Native API driver

Advantages:

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantages:

- The Native API driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver(fully java driver)

The Network protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol .it is fully written in java.

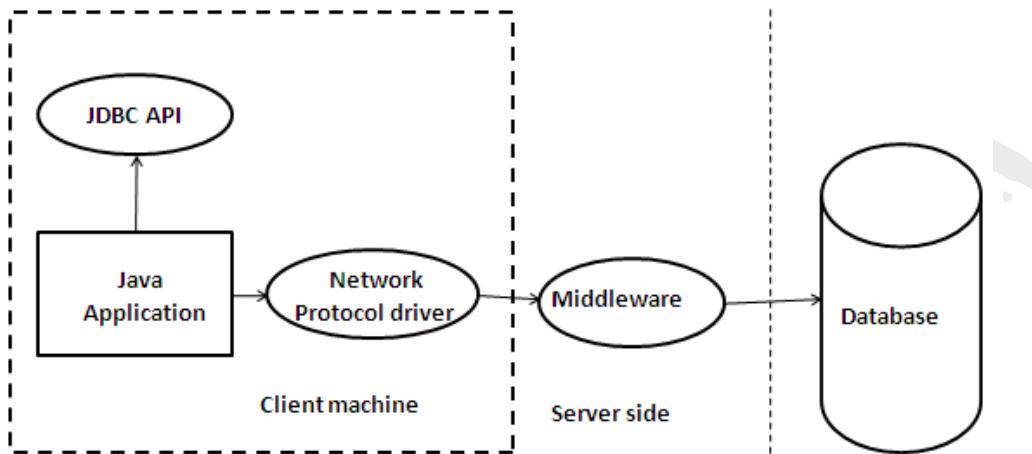


Figure- Network Protocol Driver

Advantages:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier

4) Thin driver(fully java driver)

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in java language.

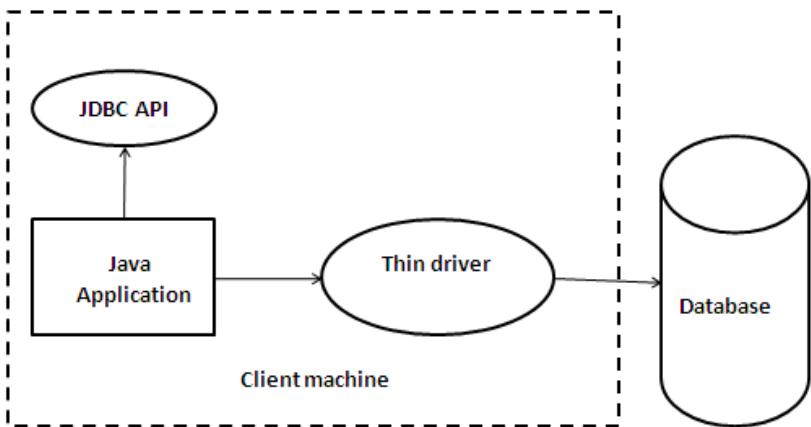


Figure- Thin Driver

Advantages:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantages:

- Drivers depends on the Database.

JDBC CONNECTIVITY

JDBC stands Java data base connectivity , which is a standard java API, which is used for different databases or to connect databases between java program and wide range of databases.

The JDBC libraries include API's for each of the tasks mentioned below

- i. Connection to a database
- ii. Creating SQL to MYSQL statements
- iii. Executing SQL or MYSQL queries.
- iv. View and modifying the results or records.

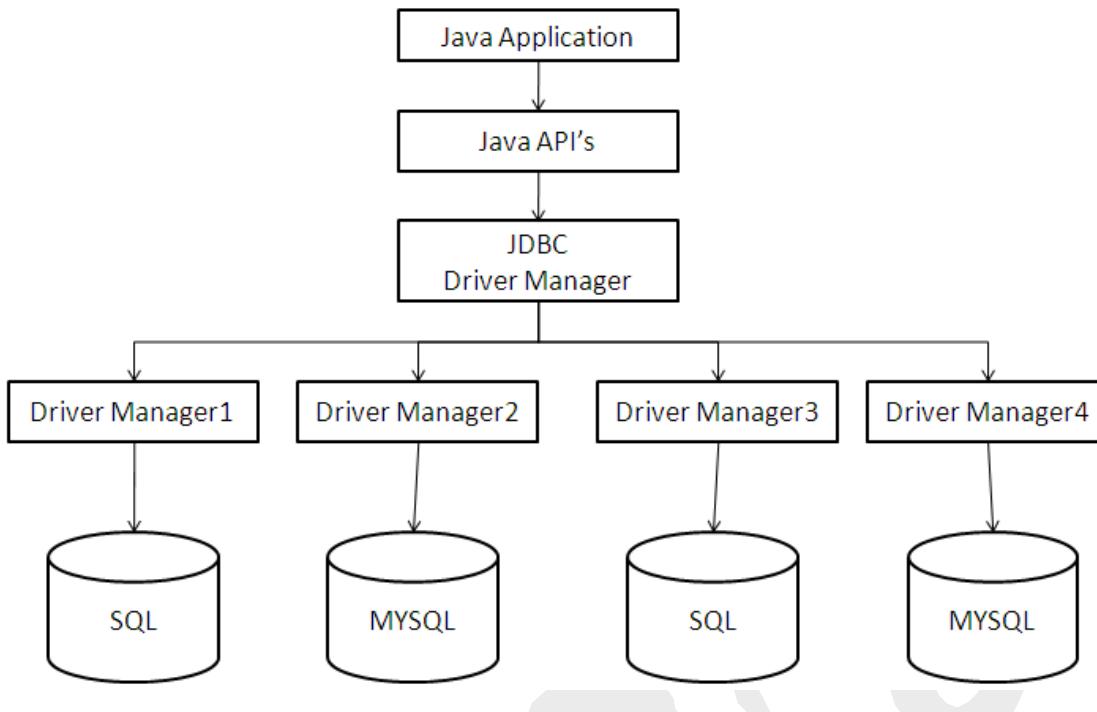
Fundamentally JDBC is a specification that provides a complete session of interface.

Java can be used to write different types of executables such as

- i. Java Application
- ii. Java Applets
- iii. Java Servlets
- iv. Java Server pages(JSP)
- v. Java Enterprise
- vi. Java Bean(EJB)

All the above executables are able to use a JDBC driver to access a database to perform different operations on databases.

JAVA ARCHITECTURE



The JDBC API supports 2 layers

- **JDBC API:**
This provides the connection to JDBC Manager to required JDBC driver
- **JDBC Driver Manager:**
This provides connection to jdbc manager to required database driver
The JDBC API uses a driver manager and database specifies a manager to provide connection of heterogeneous databases
Note:
API stands for application program interface which is a document that contains the description of the project interms of classes and interfaces that can be used for communication with each other

5steps to connect to the database in java

1. Register the driver class
2. Create the connection object
3. Create the statement object
4. Execute the query
5. Close the connection object.

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

1. Register the driver class

The `forName()` method of class is used to register with Driver class. this method is used to Dynamically load the Driver class.

Syntax:

```
Public static void forName(String classname) throws ClassNotFoundException
```

Example:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Create the connection object

The getConnection() of DriverManager class is used to establish connection with Data base.

```
Connection con=DriverManager.getConnection(URL,String,name)
```

3. Create the statement object

To create statement object we use create statement method of connection interface. The object of the statement is responsible to execute the queries in the Database.

Syntax:

```
Public Statement create Statement() throws SQLException
```

Example:

```
Statement stmt=con.createStatement();
```

Connection

```
con=DriverManager.getConnector("jdbc:Oracle:thin@localhost:1521:xe","system","anuarg");
```

4. Execute the query

The execute query() of statement interface is used to execute the queries to the database. This method returns the object of result set that can be used to get all the records of a table.

Syntax:

```
Public Resultset.executeQuery(String SQL) throws SQLException
```

Example:

```
Result rs=stmt.executeQuery("select * from emp");
```

5. Close the connection object:

By closing connection object statements ResultSet connection will be closed

In order to close the connection we have to use a method called close() of Connection interface

Syntax:

```
Public void close() throws SQLException
```

```
Ex: con.close();
```

Example program:

```
//step.1 import required packages
import java.sql.*;
public class JDBCExample
{
    //JDBC driver name and database URL
    static final String JDBC_DRIVER="oracle.jdbc.driver.OracleDriver";
    static final String DB_URL="jdbc:oracle:thin:@localhost:1521:xe";

    //database credentials
    static final String USER="system";
    static final String PASS="system";

    public static void main(String args[])
    {
        Connection conn=null;
        Statement stmt=null;
        try
```

```
{  
    //step 2: Register JDBC driver  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
  
    //step 3: Open a connection  
    System.out.println("Connecting to database. . .");  
    conn=DriverManager.getConnection(DB_URL, USER, PASS);  
  
    //step 4: Execute a query  
    System.out.println("Creating database. . .");  
    stmt=conn.createStatement();  
  
    String sql="SELECT * FROM EMPLOYEE";  
    ResultSet rs=stmt.executeQuery(sql);  
    while(rs.next())  
    {  
        String name=rs.getString("ename");  
        String city=rs.getString("city");  
        System.out.println("name : "+name+"city : "+city);  
    }  
    rs.close();  
    stmt.close();  
    conn.close();  
    stmt.executeUpdate(sql);  
    System.out.println("Database creates successfully . . .");  
}  
catch(SQLException se)  
{  
    //handle errors for JDBC  
    se.printStackTrace();  
}  
catch(Exception e)  
{  
    //handle errors for Class.forName  
    e.printStackTrace();  
}  
finally  
{  
    //finally block used to close resources  
    try  
    {  
        if(stmt!=null)  
            stmt.close();  
    }  
    catch(SQLException se2)  
    {  
        }//nothing we can do
```

```
try
{
    if(conn!=null)
        conn.close();
}//end finally try
catch(SQLException se)
{
    se.printStackTrace();
}//end try
}
System.out.println("Goodbye !");
}//end main
}//end JDBCExample
```

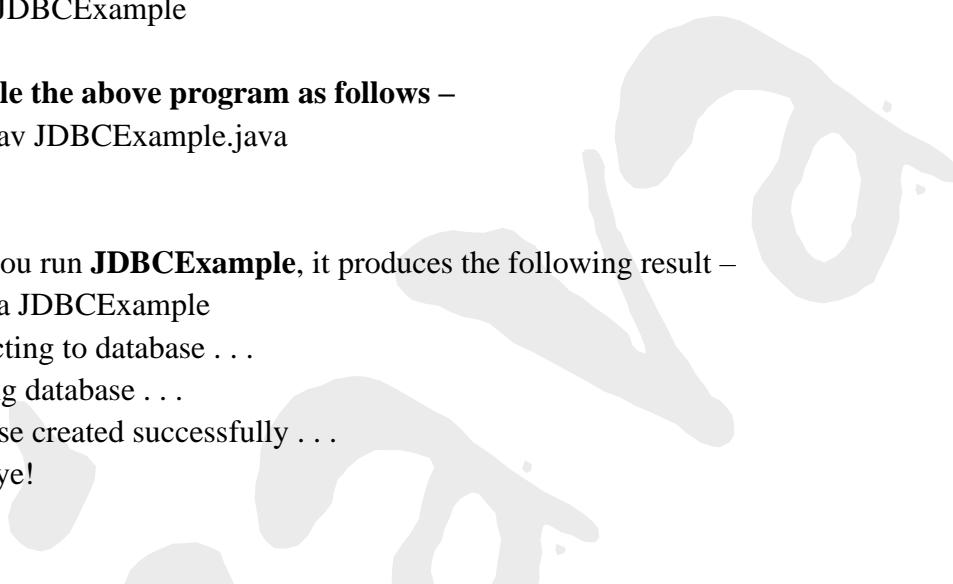
Compile the above program as follows –

```
c:\>jabav JDBCExample.java
c:\>
```

when you run **JDBCExample**, it produces the following result –

```
c:\>java JDBCExample
Connecting to database . . .
Creating database . . .
Database created successfully . . .
Goodbye!
```

Output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

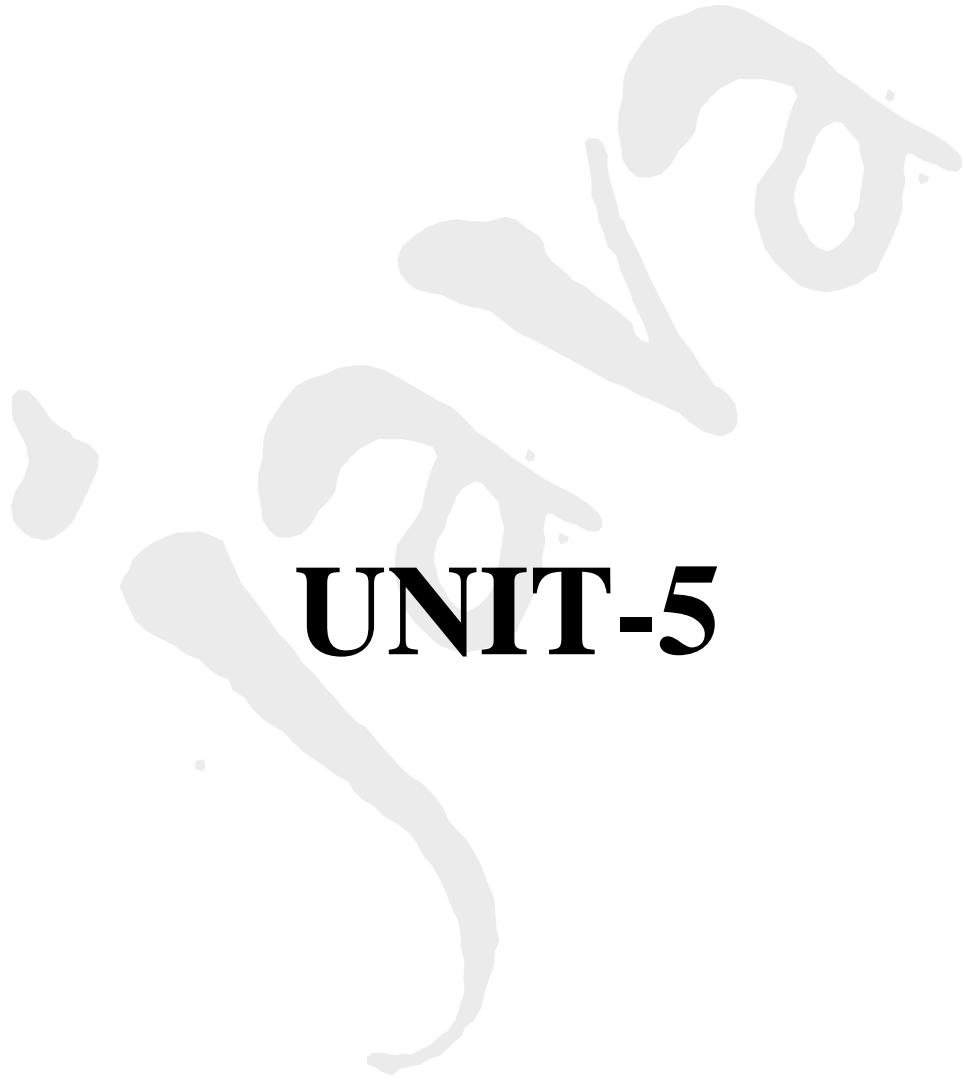
C:\Users\RANI>set classpath=.;C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\
lib\ojdbc14.jar;

C:\Users\RANI>e:

E:\>javac JDBCExamp1.java

E:\>java JDBCExamp1
Connecting to database . . .
Creating database . . .
name : ajaycity : bombay
name : sunilcity : delhi
name : anilcity : delhi
name : sunjaycity : bombay
name : sweetycity : nagpur
name : vinaycity : nagpur
name : johnicity : madras
name : vijaycity : nagpur
Database creates successfully . . .
Goodbye !

E:\>
```



UNIT-5

AWT

AWT(Abstract Window Toolkit)

By using java technology, we can design 2 types of applications

- 1) CUI(Character-based User Interface)
- 2) GUI(Graphical User Interface)

1. CUI:

CUI applications are the java applications which would be designed in such way to give input on the command prompt & to get output on the same.

Thus command prompt acts as user interface which is able to support only character data.

2. GUI:

GUI application is the java application which is used to develop to provide input on collection of GUI components & to get output on the same GUI.

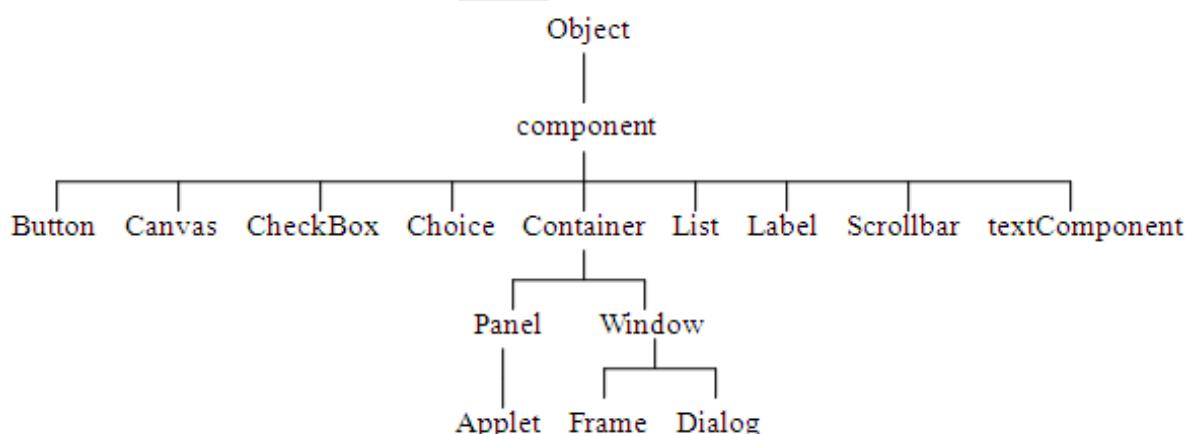
To prepare GUI application, java has provided a complete predefined in the form of following packages.

- 1) Java.awt
- 2) Javax.swing
- 3) Java.applet

AWT:

AWT is an API which provides predefined library window based application which are used to prepare components i.e. static components to provide static components we will be using java.awt package.

To provide events/life to static components we use java.awt.Event package.



1. Component:

Component is an object which is displayed pictorially on the frame.

Ex: Button

2. Container:

Container is a GUI component which accommodates all other GUI component

Ex: frame, window, applet, panel.

3. Event:

It is nothing but an action generator on the component or change mode to the state of the object.

Ex: checkbox check, button click, scrollbar scrolled horizontally & vertically..

4. Frame:

A frame is a container provided by java.awt package in the form of frame class.

CUI Vs GUI:

- CUI and GUI are acronyms that stand for different kinds of user interface systems. These are terms used in reference to computers.
- CUI stands for character user interface while GUI refers to graphical user interface.
- Though both are interfaces and serve the purpose of running the programs they differ in their features and the control provide to the user.

CUI:

- CUI means you have to take help of a keyboard to type commands to interact with computer. You can only type text to give commands to the computer as in MS DOS or command prompt.
- There are no images or graphics on the screen and it is a primitive interface and users who have seen it say that they had to contend with a black screen with white text only.
- In those days there was no use of mouse as CUI did not support the use of pointer devices.
- CUI's have generally become outdated with the more advanced GUI taking their places.
- However even the most modern computers have a modified version of CUI called CLI(command line interrupt).

GUI:

- GUI is what most modern computers make use of
- This is an interface that makes use of graphics, images and other visual clues such as icons.
- This interface mode if possible for a mouse to be used with a computer and interaction really become very easy as the user could interact with just a click of the mouse rather having to type every time to give commands to the computer.

DIFFERENCE BETWEEN CUI AND GUI:

- CUI and GUI are user interface used in connection with computers.
- CUI is the pre-cursor of GUI and starts for character user interface where user has to type on keyboard to proceed on the other hand GUI stands for Graphical user interface which makes it possible to use a mouse instead of keyboard.
- GUI is much easier to navigate than CUI.
- There is only text in case of CUI where as there are graphics and other visual clues in case of GUI.
- Most modern computers use GUI and not CUI.
- DOS is an example of CUI where as windows is an example of GUI predefined packages in the form of following packages.
 1. Java.awt
 - a. UI element
 - b. Container/layout mechanism
 - c. Event.

2. Java.swing
3. Java.applet

Java.awt:

- AWT stands for abstract window toolkit which provided predefined library web based applications
- AWT API is platform dependent for creating GUI.
- AWT is platform dependent because the java.awt calls native operating system subroutine for creating components such as textbox, checkbox, buttons etc that is and awt GUI may have different look and feel across platform like windows, Unix, Mac OS etc.
- This is because of platforms which have their own look and feel.
- AWT is heavy weighted when compared to other packages because of the native operating system takes responsibility of creating the components.
- AWT components are static in native i.e., they don't have any kind of action or life.
- In order to provide an action or life to the components have to use
 Java.awt.event package

COMPONENTS AND CONTAINERS:

- All the elements like buttons, text field, scrollbars etc are known as components.
- In AWT we have classes for each component
- To have everything placed on the screen to a particular position we have to add them to a container.
- A container is like a screen where in we are placing components like buttons, text fields, checkbox etc.
- In short a container contains and controls the layout of components.
- A container itself is a component, thus we can add a container inside container.

TYPES OF CONTAINER:

- As explained above, a container is a place where in we add components like text field, button, checkboxes etc.
- There are 4 types of containers available in AWT:
 1. Windows
 2. Frame
 3. Dialog box and
 4. panel
- Frame and dialog are sub classes of windows class
 1. **Window:**
An instance of the window class has no border and no title.
 2. **Dialog:**

Dialog class has border and title. An instance of dialog class cannot exist without an associated instance of frame class

3. Panel:

Panel does not contain the title bar, menu bar or border. It is a generic container for holding components. An instance of panel class provides a container to which to add components.

4. Frame:

A frame has title, border and menu bars. It can contain several components like buttons, text fields, scrollbars etc. This is most widely used container while developing an application in AWT.

Every AWT controls inherits properties from Component class.

1. Component

A Component is an abstract super class for GUI controls and it represents an object with graphical representation.

AWT UI Elements:

Following is the list of commonly used controls while designed GUI using AWT.

SNO	Control	Description
1.	Label	A Label object is a component for placing text in a container.
2.	Button	This class creates a labeled button.
3.	Check Box	A check box is a graphical component that can be in either an on true or off false state.
4.	Check Box Group	The CheckboxGroup class is used to group the set of checkbox.
5.	List	The List component presents the user with a scrolling list of text items.
6.	Text Field	A TextField object is a text component that allows for the editing of a single line of text.
7.	Text Area	A TextArea object is a text component that allows for the editing of a multiple lines of text.
8.	Choice	A Choice control is used to show popup menu of choices. Selected choice is shown on the top of the menu.
9.	Canvas	A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10.	Image	An Image control is super class for all image classes representing graphical images.
11.	Scroll Bar	A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12.	Dialog	A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13.	File Dialog	A FileDialog control represents a dialog window from which the user can select a file.

AWT LAYOUTS

INTRODUCTION:

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

LAYOUT MANAGER:

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

1. It is very tedious to handle a large number of controls within the container.

2. Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide us with various layout manager to position the controls.

The properties like size, shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Following are the interfaces defining functionalities of Layout Managers:

SNO	INTERFACE	DESCRIPTION
1.	LayoutManager	The LayoutManager interface declares those methods which need to be implemented by the class whose object will act as a layout manager.
2.	LayoutManager2	The LayoutManager2 is the sub-interface of the LayoutManager. This interface is for those classes that know how to layout containers based on layout constraint object.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

SNO	Layout manager	Description
1.	BorderLayout	The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
2.	CardLayout	The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3.	FlowLayout	The FlowLayout is the default layout. It layouts the components in a directional flow.
4.	GridLayout	The GridLayout manages the components in form of a rectangular grid.
5.	GridBagLayout	This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

FRAME CLASS

INTRODUCTION

The class **Frame** is a top level window with border and title. It uses BorderLayout as default layout manager.

CLASS DECLARATION

Following is the declaration for **java.awt.Frame** class:

```
public class Frame extends Window implements MenuContainer
```

FIELD

Following are the fields for **java.awt.Frame** class:

SNO	FIELD	DESCRIPTION
1.	static float BOTTOM_ALIGNMENT	Ease-of-use constant for getAlignmentY.
2.	static int CROSSHAIR_CURSOR	Deprecated. Replaced by Cursor.CROSSHAIR_CURSOR.
3.	static int DEFAULT_CURSOR	Deprecated. Replaced by Cursor.DEFAULT_CURSOR.
4.	static int E_RESIZE_CURSOR	Deprecated. Replaced by Cursor.E_RESIZE_CURSOR.
5.	static int HAND_CURSOR	Deprecated. Replaced by Cursor.HAND_CURSOR.
6.	static int ICONIFIED	This state bit indicates that frame is iconified.
7.	static int MAXIMIZED_BOTH	This state bit mask indicates that frame is fully maximized i.e. both horizontally & vertically.
8.	static int MAXIMIZED_HORIZ	This state bit indicates that frame is maximized in the horizontal direction.
9.	static int MAXIMIZED_VERT	This state bit indicates that frame is maximized in the vertical direction.
10.	static int MOVE_CURSOR	Deprecated. Replaced by Cursor.MOVE_CURSOR.
11.	static int N_RESIZE_CURSOR	Deprecated. Replaced by Cursor.N_RESIZE_CURSOR.
12.	static int NE_RESIZE_CURSOR	Deprecated. Replaced by Cursor.NE_RESIZE_CURSOR.
13.	static int NORMAL	Frame is in the "normal" state.
14.	static int NW_RESIZE_CURSOR	Deprecated. Replaced by Cursor.NW_RESIZE_CURSOR.
15.	static int S_RESIZE_CURSOR	Deprecated. Replaced by Cursor.S_RESIZE_CURSOR.
16.	static int SE_RESIZE_CURSOR	Deprecated. Replaced by Cursor.SE_RESIZE_CURSOR.
17.	static int SW_RESIZE_CURSOR	Deprecated. Replaced by Cursor.SW_RESIZE_CURSOR.
18.	static int TEXT_CURSOR	Deprecated. Replaced by Cursor.TEXT_CURSOR.
19.	static int W_RESIZE_CURSOR	Deprecated. Replaced by Cursor.W_RESIZE_CURSOR.
20.	static int WAIT_CURSOR	Deprecated. Replaced by Cursor.WAIT_CURSOR.

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1.	Frame	Constructs a new instance of Frame that is initially invisible.
2.	FrameGraphicsConfigurationgc	Constructs a new, initially invisible Frame with the specified GraphicsConfiguration.
3.	FrameStringtitle	Constructs a new, initially invisible Frame object with the specified title.
4.	FrameStringtitle, GraphicsConfigurationgc	Constructs a new, initially invisible Frame object with the specified title and a GraphicsConfiguration.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1.	void addNotify	Makes this Frame displayable by connecting it to a native screen resource.
2.	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated

		with this Frame.
3.	int getCursorType	Deprecated. As of JDK version 1.1, replaced by Component.setCursor.
4.	int getExtendedState	Gets the state of this frame.
5.	static Frame[] getFrames	Returns an array of all Frames created by this application.
6.	Image getIconImage	Returns the image to be displayed as the icon for this frame.
7.	Rectangle getMaximizedBounds	Gets maximized bounds for this frame.
8.	MenuBar getMenuBar	Gets the menu bar for this frame.
9.	int getState	Gets the state of this frame <i>obsolete</i> .
10.	String getTitle	Gets the title of the frame.
11.	boolean isResizable	Indicates whether this frame is resizable by the user.
12.	boolean isUndecorated	Indicates whether this frame is undecorated.
13.	protected String paramString	Returns a string representing the state of this Frame.
14.	void removeMenuComponent	Removes the specified menu bar from this frame.
15.	void removeNotify	Makes this Frame undisplayable by removing its connection to its native screen resource.
16.	void setCursor(int cursorType)	Deprecated. As of JDK version 1.1, replaced by Component.setCursor(Cursor).
17.	void setState(int state)	Sets the state of this frame.
18.	void setIconImage(Image image)	Sets the image to be displayed as the icon for this window.
19.	void setMaximizedBounds(Rectangle bounds)	Sets the maximized bounds for this frame.
20.	void setMenuBar(MenuBar mb)	Sets the menu bar for this frame to the specified menu bar.
21.	void setResizable(boolean resizable)	Sets whether this frame is resizable by the user.
22.	void setState(int state)	Sets the state of this frame <i>obsolete</i> .
23.	void setTitle(String title)	Sets the title for this frame to the specified string.
24.	void setUndecorated(boolean undecorated)	Disables or enables decorations for this frame.

METHODS INHERITED

This class inherits methods from the following classes:

- java.awt.Window
- java.awt.Container
- java.awt.Component
- java.lang.Object

CREATION OF FRAME

Frame can be created into two ways

1. by using frame class
2. by extending frame class

- i. In order to make frame visible, we have to use following method from Frame class
Syntax:

```
public void setVisible(Boolean b)
```
- ii. If b=true frame will be in visible mode else it is invisible (by default b=false).
- iii. When we prepare a frame, the frame will be available with zero height & zero width. To increase the size we have the following method
Syntax:

```
public void setSize(int width,int height)
```
- iv. When we prepare a frame, we will get 3 buttons i.e. maximize, minimize & close. By Default minimize and maximize works but close button does not work.
- v. To provide title to the frame explicitly, we have to use the following method.
Syntax:

```
public void setTitle(String title)
```
- vi. When a frame is created to set a background color to the frame, we have to use the following method
Syntax:

```
public void setBackground(Color.colormame)
```

Note: Color is a class defined in java.awt package which defines constants like red, green, yellow etc....
- vii. By using above approach that is by creating frame class object directly it is possible to provide customization on to the frame. In order to do customization, we have to declare a user defined where we have to define a user defined class to the frame class and we have to provide by taking its constructors.
- viii. To display the text on the screen
If we want to display the text/graphs/diagrams on to the frame then we have to override the paint() method from Frame class.
Syntax:

```
public void paint(Graphics g)
```
- ix. To set a particular font to the text we have to use font class present in java.awt.
Syntax:

```
Font f=new Font(String type,int style,int size)
```

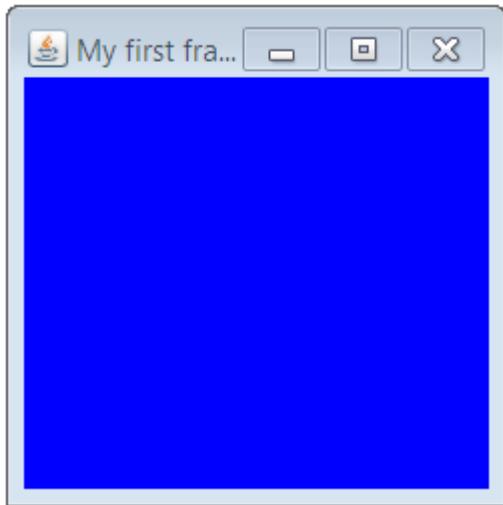
Ex: Font f=new Font("arial",font.BOLD,24)

BY USING FRAME CLASS

Example:

```
import java.awt.*;
class AwtEx
{
    public static void main(String args[])
    {
        Frame f=new Frame();
        f.setVisible(true);
        f.setSize(250,250);
        f.setBackground(Color.blue);
        f.setTitle("My first frame");
    }
}
```

Output:

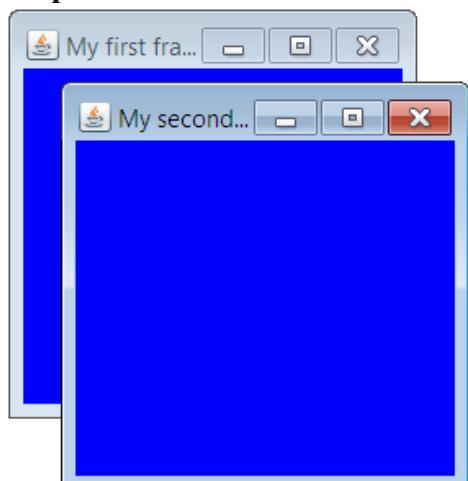


BY EXTENDING FRAME CLASS

Example:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
        setBackground(Color.blue);
    }
}
class AwtEx
{
    public static void main(String args[])
    {
        Myframe f1=new Myframe();
        Myframe f2=new Myframe();
        f1.setTitle("My first frame");
        f2.setTitle("My second frame");
    }
}
```

Output:



LABEL CLASS

INTRODUCTION

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

CLASS DECLARATION

Following is the declaration for **java.awt.Label** class:

```
public class Label extends Component implements Accessible
```

FIELD

Following are the fields for **java.awt.Component** class:

static int CENTER	Indicates that the label should be centered.
static int LEFT	Indicates that the label should be left justified.
static int RIGHT	Indicates that the label should be right justified.

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1.	Label	Constructs an empty label.
2.	LabelStringtext	Constructs a new label with the specified string of text, left justified.
3.	LabelStringtext, int alignment	Constructs a new label that presents the specified string of text with the specified alignment.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1.	void addNotify	Creates the peer for this label.
2.	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this Label.
3.	int getAlignment	Gets the current alignment of this label.
4.	String getText	Gets the text of this label.
5.	protected String paramString	Returns a string representing the state of this Label.
6.	void setAlignmentint alignment	Sets the alignment for this label to the specified alignment.
7.	void setTextStringtext	Sets the text for this label to the specified text.

METHODS INHERITED

This class inherits methods from the following classes:

java.awt.Component

java.lang.Object

EXAMPLE PROGRAM ON LABEL CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
```

```
Myframe()
{
    setVisible(true);
    setSize(250,250);
}
class LabelEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("Label example");
        Label l=new Label();
        l.setText("*** label example in a frame ***");
        Label l1=new Label("welcome");
        f.add(l);
        f.add(l1);
    }
}
```

OUTPUT:



AWT BUTTON CLASS

INTRODUCTION

Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button. The button's processEvent method receives all events for the button; it passes an action event along by calling its own processActionEvent method. The latter method passes the action event on to any action listeners that have registered an interest in action events generated by this button.

If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

CLASS DECLARATION

Following is the declaration for **java.awt.Button** class:

```
public class Button extends Component implements Accessible
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1.	Button	Constructs a button with an empty string for its label.
2.	ButtonStringtext	Constructs a new button with specified label.

CLASS METHODS

SNO	METHODS	DESCRIPTION
1.	void addActionListenerActionListener1	Adds the specified action listener to receive action events from this button.
2.	void addNotify	Creates the peer of the button.
3.	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this Button.
4.	String getActionCommand	Returns the command name of the action event fired by this button.
5.	ActionListener[] getActionListeners	Returns an array of all the action listeners registered on this button.
6.	String getLabel	Gets the label of this button.
7.	<T extends EventListener> T[] getListenersClass < T > listenerType	Returns an array of all the objects currently registered as FooListeners upon this Button.
8.	protected String paramString	Returns a string representing the state of this Button.
9.	protected void processActionEventActionEvente	Processes action events occurring on this button by dispatching them to any registered ActionListener objects.
10.	protected void processEventAWTEvente	Processes events on this button.
11.	void removeActionListenerActionListener1	Removes the specified action listener so that it no longer receives action events from this button.
12.	void setActionCommandStringcommand	Sets the command name for the action event fired by this button.
13.	void setLabelStringlabel	Sets the button's label to be the specified string.

Methods inherited

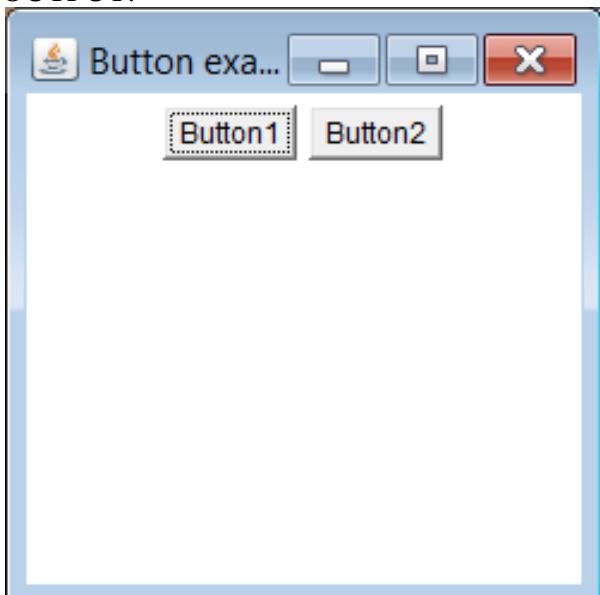
This class inherits methods from the following classes:

java.awt.Component
java.lang.Object

EXAMPLE PROGRAM ON BUTTON CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class ButtonEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("Button example");
        Button b1=new Button("Button1");
        Button b2=new Button("Button2");
        f.add(b1);
        f.add(b2);
    }
}
```

OUTPUT:



AWT TEXTFIELD CLASS

INTRODUCTION

The textField component allows the user to edit single line of text. When the user types a key in the text field the event is sent to the TextField. The key event may be key pressed, Key released or key typed. The key event is passed to the registered KeyListener. It is also possible to fire an ActionEvent if the ActionEvent is enabled on the textfield then ActionEvent may be fired by pressing the return key.

CLASS DECLARATION

Following is the declaration for **java.awt.TextField** class:

```
public class TextField extends TextComponent
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1.	TextField	Constructs a new text field.
2.	TextFieldintcolumns	Constructs a new empty text field with the specified number of columns.
3.	TextFieldStringtext	Constructs a new text field initialized with the specified text.
4.	TextFieldStringtext, intcolumns	Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	void addActionListenerActionListenerl	Adds the specified action listener to receive action events from this text field.
2	void addNotify	Creates the TextField's peer.
3	boolean echoCharIsSet	Indicates whether or not this text field has a character set for echoing.
4	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this TextField.
5	ActionListener[] getActionListeners	Returns an array of all the action listeners registered on this textfield.
6	int getColumns	Gets the number of columns in this text field.
7	char getEchoChar	Gets the character that is to be used for echoing.
8	<T extends EventListener> T[] getListenersClass < T > listenerType	Returns an array of all the objects currently registered as FooListeners upon this TextField.
9	Dimension getMinimumSize	Gets the minimum dimensions for this text field.
10	Dimension getMinimumSizeintcolumns	Gets the minimum dimensions for a text field with the specified number of columns.
11	Dimension getPreferredSize	Gets the preferred size of this text field.
12	Dimension getPreferredSizeintcolumns	Gets the preferred size of this text field with the specified number of columns.
13	Dimension minimumSize	Deprecated. As of JDK version 1.1, replaced by getMinimumSize.
14	Dimension minimumSizeintcolumns	Deprecated. As of JDK version 1.1, replaced by getMinimumSizeint.
15	protected String paramString	Returns a string representing the state of

		this TextField.
16	Dimension preferredSize	Deprecated. As of JDK version 1.1, replaced by getPreferredSize.
17	Dimension preferredSizeintcolumns	Deprecated. As of JDK version 1.1, replaced by getPreferredSizeint.
18	protected void processActionEventActionEvente	Processes action events occurring on this text field by dispatching them to any registered ActionListener objects.
19	protected void processEventAWTEvente	Processes events on this text field.
20	void removeActionListenerActionListenerl	Removes the specified action listener so that it no longer receives action events from this text field.
21	void setColumnsintcolumns	Sets the number of columns in this text field.
22	void setEchoCharchar	Sets the echo character for this text field.
23	void setEchoCharacterchar	Deprecated. As of JDK version 1.1, replaced by setEchoCharchar.
24	void setTextStringt	Sets the text that is presented by this text component to be the specified text.

Methods inherited

This class inherits methods from the following classes:

java.awt.TextComponent

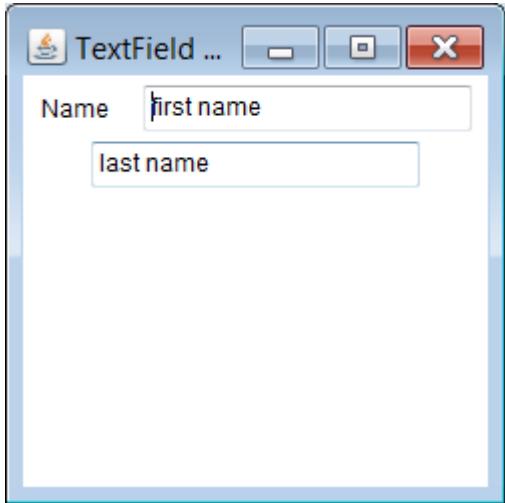
java.awt.Component

java.lang.Object

EXAMPLE PROGRAM ON TEXTFIELD CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class TextFieldEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("TextField example");
        Label name=new Label("Name");
        TextField fn,ln;
        fn=new TextField("first name",20);
        ln=new TextField("last name",20);
        f.add(name);
        f.add(fn);
        f.add(ln);
    }
}
```

OUTPUT:



AWT FLOWLAYOUT CLASS

INTRODUCTION

The class **FlowLayout** components in a left-to-right flow.

CLASS DECLARATION

Following is the declaration for **java.awt.FlowLayout** class:

```
public class FlowLayout extends Object implements LayoutManager, Serializable
```

FIELD

Following are the fields for **java.awt.BorderLayout** class:

- **static int CENTER** -- This value indicates that each row of components should be centered.
- **static int LEADING** -- This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
- **static int LEFT** -- This value indicates that each row of components should be left-justified.
- **static int RIGHT** -- This value indicates that each row of components should be right-justified.
- **static int TRAILING** -- This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

CLASS CONSTRUCTORS

1. **FlowLayout**

Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

2. **FlowLayout(int align)**

Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.

3. **FlowLayout(int align, int hgap, int vgap)**

Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

CLASS METHODS

1. **void addLayoutComponent(String name, Component comp)**

Adds the specified component to the layout.

2. **int getAlignment**

Gets the alignment for this layout.

3. **int getHgap**

Gets the horizontal gap between components.

4. **int getVgap**

Gets the vertical gap between components.

5. **void layoutContainer(Container target)**

Lays out the container.

6. **Dimension minimumLayoutSize(Container target)**

Returns the minimum dimensions needed to layout the visible components contained in the specified target container.

7. **Dimension preferredLayoutSize(Container target)**

Returns the preferred dimensions for this layout given the visible components in the specified target container.

8. **void removeLayoutComponent(Component comp)**

Removes the specified component from the layout.

9. **void setAlignment(int align)**

Sets the alignment for this layout.

10. **void setHgap(int hgap)**

Sets the horizontal gap between components.

11. **void setVgap(int vgap)**

Sets the vertical gap between components.

12. String `toString`

Returns a string representation of this FlowLayout object and its values.

METHODS INHERITED

This class inherits methods from the following classes:

`java.lang.Object`

AWT LIST CLASS

INTRODUCTION:

The List represents a list of text items. The list can be configured to that user can choose either one item or multiple items.

CLASS DECLARATION

Following is the declaration for **java.awt.List** class:

```
public class List extends Component implements Item Selectable, Accessible
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1	List	Creates a new scrolling list.
2	Listintrows	Creates a new scrolling list initialized with the specified number of visible lines.
3	Listintrows, booleanmultipleMode	Creates a new scrolling list initialized to display the specified number of rows.

CLASS METHODS

<T extends EventListener> T[] getListenersClass < T > *listenerType*

Returns an array of all the objects currently registered as FooListeners upon this List.

SNO	METHOD	DESCRIPTION
1	void addStringitem	Adds the specified item to the end of scrolling list.
2	void addStringitem, intindex	Adds the specified item to the the scrolling list at the position indicated by the index.
3	void addActionListenerActionListenerl	Adds the specified action listener to receive action events from this list.
4	void addItemStringitem	Deprecated. replaced by addString
5	void addItemStringitem, intindex	Deprecated. replaced by addString, int.
6	void addItemListenerItemListenerl	Adds the specified item listener to receive item events from this list.
7	void addNotify	Creates the peer for the list.
8	boolean allowsMultipleSelections	Deprecated. As of JDK version 1.1, replaced by isMultipleMode.
9	void clear	Deprecated. As of JDK version 1.1, replaced by removeAll.
10	int countItems	Deprecated. As of JDK version 1.1, replaced by getItemCount
11	void delItemintposition	Deprecated. replaced by removeString and removeint.
12	void delItemsintstart, intend	Deprecated. As of JDK version 1.1, Not for public use in the future. This method is expected to be retained only as a package private method.
13	void deselectintindex	Deselects the item at the specified index.
14	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this List.
15	ActionListener[] getActionListeners	Returns an array of all the action listeners registered on this list.
16	String getItemintindex	Gets the item associated with the specified index.
17	int getItemCount	Gets the number of items in the list.
18	ItemListener[] getItemListeners	Returns an array of all the item listeners registered on this list.

19	String[] getItems	Gets the items in the list.
20	Dimension getMinimumSize	Determines the minimum size of this scrolling list.
21	Dimension getMinimumSizeintrows	Gets the minumum dimensions for a list with the specified number of rows.
22	Dimension getPreferredSize	Gets the preferred size of this scrolling list.
23	Dimension getPreferredSizeintrows	Gets the preferred dimensions for a list with the specified number of rows.
24	int getRows	Gets the number of visible lines in this list.
25	int getSelectedIndex	Gets the index of the selected item on the list.
26	int[] getSelectedIndexes	Gets the selected indexes on the list.
27	String getSelectedItem	Gets the selected item on this scrolling list.
28	String[] getSelectedItems	Gets the selected items on this scrolling list.
29	Object[] getSelectedObjects	Gets the selected items on this scrolling list in an array of Objects.
30	int getVisibleIndex	Gets the index of the item that was last made visible by the method <code>makeVisible</code>
31	boolean isIndexSelectedintindex	Determines if the specified item in this scrolling list is selected.
32	boolean isMultipleMode	Determines whether this list allows multiple selections.
33	boolean isSelectedintindex	Deprecated. As of JDK version 1.1, replaced by <code>isIndexSelectedint</code> .
34	void makeVisibleintindex	Makes the item at the specified index visible.
35	Dimension minimumSize	Deprecated. As of JDK version 1.1, replaced by <code>getMinimumSize</code>
36	Dimension minimumSizeintrows	Deprecated. As of JDK version 1.1, replaced by <code>getMinimumSizeint</code> .
37	protected String paramString	Returns the parameter string representing the state of this scrolling list.
38	Dimension preferredSize	Deprecated. As of JDK version 1.1, replaced by <code>getPreferredSize</code> .
39	Dimension preferredSizeintrows	Deprecated. As of JDK version 1.1, replaced by <code>getPreferredSizeint</code> .
40	protected void processActionEventActionEvent	Processes action events occurring on this component by dispatching them to any registered ActionListener objects.
41	protected void processEventA WTEvent	Processes events on this scrolling list.
42	protected void processItemEventItemEvent	Processes item events occurring on this list by dispatching them to any registered ItemListener objects.
43	void removeintposition	Removes the item at the specified position from this scrolling list.
44	void removeStringitem	Removes the first occurrence of an item from the list.
45	void removeActionListenerActionListener	Removes the specified action listener so that it no longer receives action events from this list.
46	void removeAll	Removes all items from this list.
47	void removeItemListenerItemListener	Removes the specified item listener so that it no longer receives item events from this list.
48	void removeNotify	Removes the peer for this list.
49	void replaceItemStringnewValue,	Replaces the item at the specified index in the

	<i>int</i> index	scrolling list with the new string.
50	void select(int index)	Selects the item at the specified index in the scrolling list.
51	void setMultipleMode(boolean b)	Sets the flag that determines whether this list allows multiple selections.
52	void setMultipleSelections(boolean b)	Deprecated. As of JDK version 1.1, replaced by <code>setMultipleMode(boolean)</code> .

Methods inherited

This class inherits methods from the following classes:

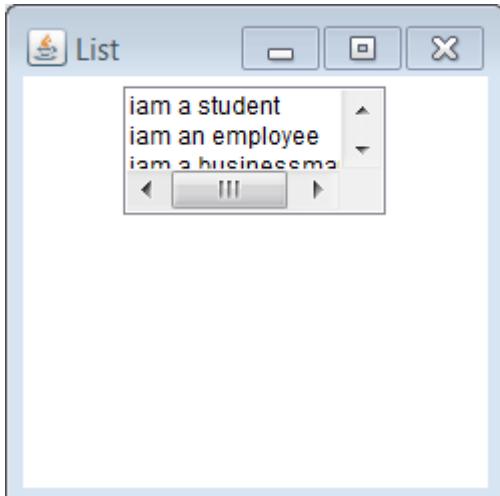
java.awt.Component

java.lang.Object

EXAMPLE PROGRAM ON LIST CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class Olform1
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("List");
        List iam=new List();
        iam.add("iam a student");
        iam.add("iam an employee");
        iam.add("iam a businessman");
        f.add(iam);
    }
}
```

OUTPUT:



AWT CHECKBOX CLASS

INTRODUCTION

Checkbox control is used to turn an option on *true* or off *false*. There is label for each checkbox representing what the checkbox does. The state of a checkbox can be changed by clicking on it.

CLASS DECLARATION

Following is the declaration for **java.awt.Checkbox** class:

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1	Checkbox	Creates a check box with an empty string for its label.
2	CheckboxStringlabel	Creates a check box with the specified label.
3	CheckboxStringlabel, booleanstate	Creates a check box with the specified label and sets the specified state.
4	CheckboxStringlabel, booleanstate, CheckboxGroupgroup	Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
5	CheckboxStringlabel, CheckboxGroupgroup, booleanstate	Creates a check box with the specified label, in the specified check box group, and set to the specified state.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	void addItemListenerItemListenerl	Adds the specified item listener to receive item events from this check box.
2	void addNotify	Creates the peer of the Checkbox.
3	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this Checkbox.
4	CheckboxGroup getCheckboxGroup	Determines this check box's group.
5	ItemListener[] getItemListeners	Returns an array of all the item listeners registered on this checkbox.
6	String getLabel	Gets the label of this check box.
7	<T extends EventListener>T[] getListenersClass < T > listenerType	Returns an array of all the objects currently registered as FooListeners upon this Checkbox.
8	Object[] getSelectedObjects	Returns an array <i>length1</i> containing the checkbox label or null if the checkbox is not selected.
9	boolean getState	Determines whether this check box is in the on or off state.
10	protected String paramString	Returns a string representing the state of this Checkbox.
11	protected void processEventAWTEvente	Processes events on this check box.
12	protected void processItemEventItemEvente	Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
13	void	Removes the specified item listener so that

	removeItemListener <i>ItemListener1</i>	the item listener no longer receives item events from this check box.
14	void setCheckboxGroup <i>CheckboxGroup</i>	Sets this check box's group to the specified check box group.
15	void setLabelString <i>label</i>	Sets this check box's label to be the string argument.
16	void setStateboolean <i>state</i>	Sets the state of this check box to the specified state.

Methods inherited

This class inherits methods from the following classes:

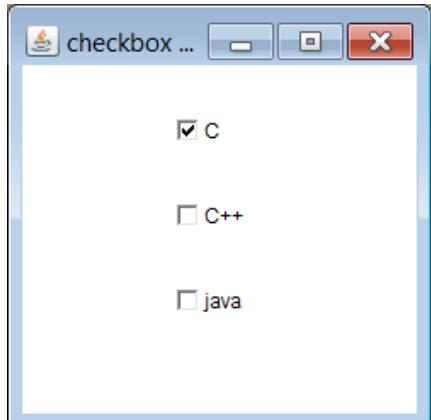
java.awt.Component

java.lang.Object

EXAMPLE PROGRAM ON CHECKBOX CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class CheckboxEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(null);
        f.setTitle("checkbox example");
        Checkbox cb1,cb2,cb3;
        cb1 = new Checkbox("C",true);
        cb1.setBounds(100,50,50,50);
        cb2 = new Checkbox("C++",false);
        cb2.setBounds(100,100,50,50);
        cb3 = new Checkbox("java",false);
        cb3.setBounds(100,150,50,50);
        f.add(cb1); f.add(cb2);f.add(cb3);
    }
}
```

OUTPUT:



AWT CHECKBOXGROUP CLASS

INTRODUCTION

The CheckboxGroup class is used to group the set of checkbox.

CLASS DECLARATION

Following is the declaration for **java.awt.CheckboxGroup** class:

public class CheckboxGroup extends Object implements Serializable

CLASS CONSTRUCTORS

SNO	CONSTRUCTORS	DESCRIPTION
1	CheckboxGroup	Creates a new instance of CheckboxGroup.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	Checkbox getCurrent	Deprecated. As of JDK version 1.1, replaced by <code>getSelectedCheckbox</code> .
2	Checkbox getSelectedCheckbox	Gets the current choice from this check box group
3	void setCurrentCheckbox	Deprecated. As of JDK version 1.1, replaced by <code>setSelectedCheckbox</code> .
4	void setSelectedCheckbox	Sets the currently selected check box in this group to be the specified check box.
5	String toString	Returns a string representation of this check box group, including the value of its current selection.

Methods inherited

This class inherits methods from the following classes:

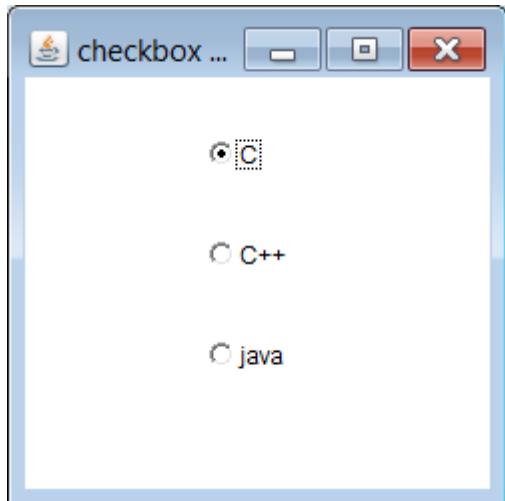
java.lang.Object

EXAMPLE PROGRAM ON CHECKBOXGROUP CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class CheckboxEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(null);
        f.setTitle("checkbox example");
        Checkbox cb1,cb2,cb3;
        CheckboxGroup cbg=new CheckboxGroup();
        cb1 = new Checkbox("C",cbg,true);
        cb1.setBounds(100,50,50,50);
        cb2 = new Checkbox("C++",false,cbg);
```

```
        cb2.setBounds(100,100,50,50);
        cb3 = new Checkbox("java",cbg,false);
        cb3.setBounds(100,150,50,50);
        f.add(cb1); f.add(cb2);f.add(cb3);
    }
}
```

OUTPUT:



AWT CHOICE CLASS

INTRODUCTION

Choice control is used to show popup menu of choices. Selected choice is shown on the top of the menu.

CLASS DECLARATION

Following is the declaration for **java.awt.Choice** class:

```
public class Choice extends Component implements ItemSelectable, Accessible
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1	Choice	Creates a new choice menu.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	void addStringItem	Adds an item to this Choice menu.
2	void addItemStringItem	Obsolete as of Java 2 platform v1.1.
3	void addItemClickListener	Adds the specified item listener to receive item events from this Choice menu.
4	void addNotify	Creates the Choice's peer.
5	int countItems	Deprecated. As of JDK version 1.1, replaced by getItemCount.
6	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this Choice.
7	String getItemAtIndex	Gets the string at the specified index in this Choice menu.
8	int getItemCount	Returns the number of items in this Choice menu.
9	ItemListener[] getItemListeners	Returns an array of all the item listeners registered on this choice.
10	<T extends EventListener> T[] getListenersClass <T> listenerType	Returns an array of all the objects currently registered as FooListeners upon this Choice.
11	int selectedIndex	Returns the index of the currently selected item.
12	String getSelectedItem	Gets a representation of the current choice as a string.
13	Object[] getSelectedObjects	Returns an array <i>length1</i> containing the currently selected item.
14	void insertStringItem, int index	Inserts the item into this choice at the specified position.
15	protected String paramString	Returns a string representing the state of this Choice menu.
16	protected void processEventAWTEvent	Processes events on this choice.
17	protected void processItemEventItemEvent	Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
18	void removeIndex	Removes an item from the choice menu at the specified position.
19	void removeStringItem	Removes the first occurrence of item from the Choice menu.
20	void removeAll	Removes all items from the choice menu.
21	void	Removes the specified item listener so that it

	removeItemListenerItemListenerl	no longer receives item events from this Choice menu.
22	void selectintpos	Sets the selected item in this Choice menu to be the item at the specified position.
23	void selectStringstr	Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

Methods inherited

This class inherits methods from the following classes:

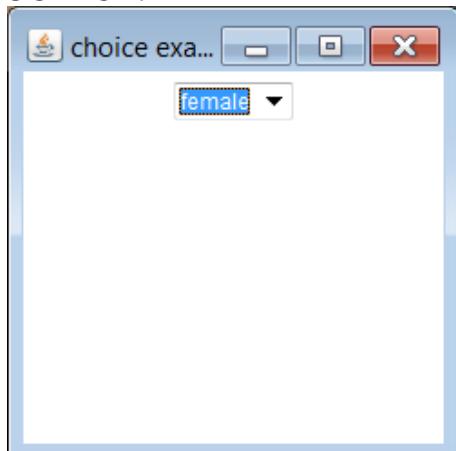
java.awt.Component

java.lang.Object

EXAMPLE PROGRAM ON CHOICE CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class ChoiceEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("choice example");
        Choice c=new Choice();
        c.insert("male",0);
        c.insert("female",1);
        f.add(c);
    }
}
```

OUTPUT:



AWT TEXTAREA CLASS

INTRODUCTION

The TextArea controls in AWT provide us multiline editor area. The user can type here as much as he wants. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up & down and right & left.

CLASS DECLARATION

Following is the declaration for **java.awt.TextArea** class:

```
public class TextArea extends TextComponent
```

FIELD

Following are the fields for **java.awt.TextArea** class:

- **static int SCROLLBARS_BOTH** -- Create and display both vertical and horizontal scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** -- Create and display horizontal scrollbar only.
- **static int SCROLLBARS_NONE** -- Do not create or display any scrollbars for the text area.
- **static int SCROLLBARS_VERTICAL_ONLY** -- Create and display vertical scrollbar only.

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1	TextArea	Constructs a new text area with the empty string as text.
2	TextArea <i>introws, intcolumns</i>	Constructs a new text area with the specified number of rows and columns and the empty string as text.
3	TextArea <i>Stringtext</i>	Constructs a new text area with the specified text.
4	TextArea <i>Stringtext, introws, intcolumns</i>	Constructs a new text area with the specified text, and with the specified number of rows and columns.
5	TextArea <i>Stringtext, introws, intcolumns, intscrollbars</i>	Constructs a new text area with the specified text, and with the rows, columns, and scrollbar visibility as specified.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	void addNotify	Creates the TextArea's peer.
2	void appendString <i>str</i>	Appends the given text to the text area's current text.
3	void appendText <i>Stringstr</i>	Deprecated. As of JDK version 1.1, replaced by <code>appendString</code> .
4	AccessibleContext getAccessibleContext	Returns the AccessibleContext associated with this TextArea.
5	int getColumns	Returns the number of columns in this text area.
6	Dimension getMinimumSize	Determines the minimum size of this text area.
7	Dimension getMinimumSize <i>introws, intcolumns</i>	Determines the minimum size of a text area with the specified number of rows and columns.
8	Dimension getPreferredSize	Determines the preferred size of this text area.
9	Dimension getPreferredSize <i>introws, intcolumns</i>	Determines the preferred size of a text area with the specified number of rows and

		columns.
10	int getRows	Returns the number of rows in the text area.
11	int getScrollbarVisibility	Returns an enumerated value that indicates which scroll bars the text area uses.
12	void insertStringstr, intpos	Inserts the specified text at the specified position in this text area.
13	void insertTextStringstr, intpos	Deprecated. As of JDK version 1.1, replaced by <code>insertString, int</code> .
14	Dimension minimumSize	Deprecated. As of JDK version 1.1, replaced by <code>getMinimumSize</code> .
15	Dimension minimumSizeintrows, intcolumns	Deprecated. As of JDK version 1.1, replaced by <code>getMinimumSizeint, int</code> .
16	protected String paramString	Returns a string representing the state of this TextArea.
17	Dimension preferredSize	Deprecated. As of JDK version 1.1, replaced by <code>getPreferredSize</code> .
18	Dimension preferredSizeintrows, intcolumns	Deprecated. As of JDK version 1.1, replaced by <code>getPreferredSizeint, int</code> .
19	void replaceRangeStringstr, intstart, intend	Replaces text between the indicated start and end positions with the specified replacement text.
20	void replaceTextStringstr, intstart, intend	Deprecated. As of JDK version 1.1, replaced by <code>replaceRangeString, int, int</code> .
21	void setColumnsintcolumns	Sets the number of columns for this text area.
22	void setRowsintrows	Sets the number of rows for this text area.

Methods inherited

This class inherits methods from the following classes:

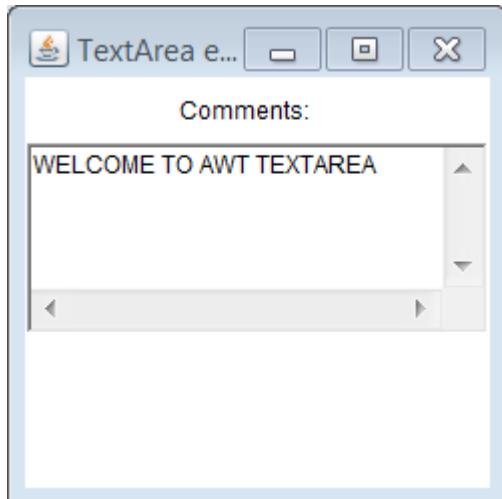
java.awt.TextComponent
 java.awt.Component
 java.lang.Object

EXAMPLE PROGRAM ON TEXTAREA CLASS:

```

import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(250,250);
    }
}
class TextAreaEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(new FlowLayout());
        f.setTitle("TextArea example");
        Label comments= new Label("Comments: ");
        TextArea ta = new TextArea(5,30);
        f.add(comments);
        f.add(ta);
    }
}
  
```

OUTPUT:



AWT PANEL CLASS

INTRODUCTION

The class **Panel** is the simplest container class. It provides space in which an application can attach any other component, including other panels. It uses FlowLayout as default layout manager.

CLASS DECLARATION

Following is the declaration for **java.awt.Panel** class:

```
public class Panel extends Container implements Accessible
```

CLASS CONSTRUCTORS

SNO	CONSTRUCTOR	DESCRIPTION
1	Panel	Creates a new panel using the default layout manager.
2	PanelLayoutManager layout	Creates a new panel with the specified layout manager.

CLASS METHODS

SNO	METHOD	DESCRIPTION
1	void addNotify	Creates the Panel's peer.
2	AccessibleContext getAccessibleContext	Gets the AccessibleContext associated with this Panel.

Methods inherited

This class inherits methods from the following classes:

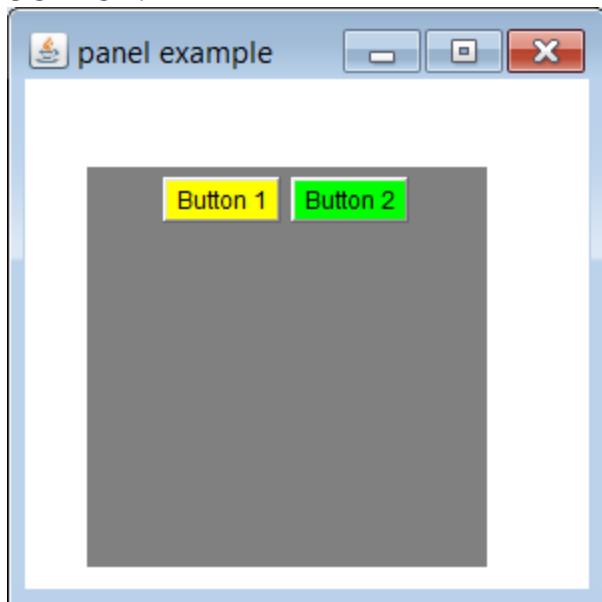
java.awt.Container
java.awt.Component
java.lang.Object

EXAMPLE PROGRAM ON PANEL CLASS:

```
import java.awt.*;
class Myframe extends Frame
{
    Myframe()
    {
        setVisible(true);
        setSize(300,300);
    }
}
class PanelEx
{
    public static void main(String args[])
    {
        Myframe f=new Myframe();
        f.setLayout(null);
        f.setTitle("panel example");
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        Button b2=new Button("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
```

}

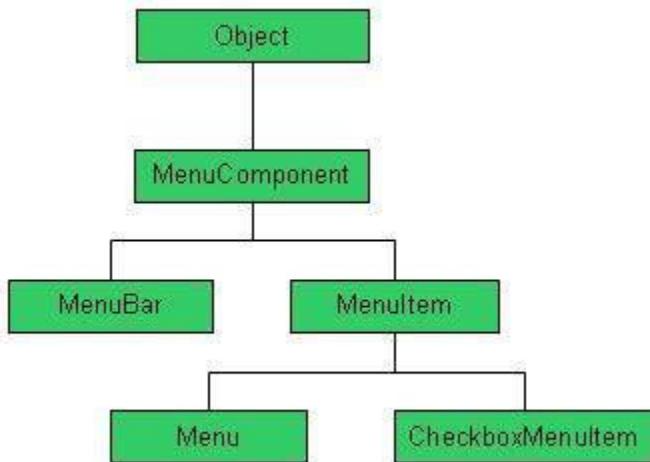
OUTPUT:



AWT Menu Classes

As we know that every top-level window has a menu bar associated with it. This menu bar consist of various menu choices available to the end user. Further each choice contains list of options which is called drop down menus. Menu and MenuItem controls are subclass of MenuComponent class.

Menu Hierarchy



Menu Controls

Sr. No.	Control & Description
1	<u>MenuComponent</u> It is the top level class for all menu related controls.
2	<u>MenuBar</u> TheMenuBar object is associated with the top-level window.
3	<u>MenuItem</u> The items in the menu must belong to the MenuItem or any of its subclass.
4	<u>Menu</u> The Menu object is a pull-down menu component which is displayed from the menu bar.
5	<u>CheckboxMenuItem</u> CheckboxMenuItem is subclass of MenuItem.
6	<u>PopupMenu</u> PopupMenu can be dynamically popped up at a specified position within a component.

AWT MenuComponent Class

Introduction

MenuComponent is an abstract class and is the superclass for all menu-related components.

Class declaration

Following is the declaration for java.awt.MenuComponent class:

```
public abstract class MenuComponent  
    extends Object  
    implements Serializable
```

Class constructors

S.N.	Constructor	Description
1	MenuComponent()	Creates a MenuComponent.

Class methods

void dispatchEvent(AWTEvent e)

S.N.	Method	Description
1	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this MenuComponent.
2	Font getFont()	Gets the font used for this menu component.
3	String getName()	Gets the name of the menu component.
4	MenuContainer getParent()	Returns the parent container for this menu component.
5	java.awt.peer.MenuComponentPeer getPeer()	Deprecated. As of JDK version 1.1, programs should not directly manipulate peers.
6	protected Object getTreeLock()	Gets this component's locking object (the object that owns the thread synchronization monitor) for AWT component-tree and layout operations.
7	protected String paramString()	Returns a string representing the state of this MenuComponent.
8	boolean postEvent(Event evt)	Deprecated. As of JDK version 1.1, replaced by dispatchEvent.
9	protected void processEvent(AWTEvent e)	Processes events occurring on this menu component.
10	void removeNotify()	Removes the menu component's peer.
11	void setFont(Font f)	Returns a representation of this menu component as a string.
12	String toString()	Sets the name of the component to the specified string.

13	void setName(String name)	Sets the font to be used for this menu component to the specified font.
----	----------------------------------	---

Methods inherited

This class inherits methods from the following classes:

- `java.lang.Object`

AWT MenuBar Class

Introduction

The MenuBar class provides menu bar bound to a frame and is platform specific.

Class declaration

Following is the declaration for `java.awt.MenuBar` class:

```
public class MenuBar
    extends MenuComponent
    implements MenuContainer, Accessible
```

Class constructors

S.N.	Constructor	Description
1	MenuBar()	Creates a new menu bar

Class methods

S.N.	Method	Description
1	void dispatchEvent(AWTEvent e)	
2	Menu add(Menu m)	Adds the specified menu to the menu bar.
3	void addNotify()	Creates the menu bar's peer.
4	int countMenus()	Creates the menu bar's peer.
5	void deleteShortcut(MenuShortcut s)	Deprecated. As of JDK version 1.1, replaced by getMenuItemCount().
6	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this MenuBar.
7	Menu getHelpMenu()	Gets the help menu on the menu bar.
8	Menu getMenu(int i)	Gets the specified menu.
9	int getMenuCount()	Gets the number of menus on the menu bar.
10	MenuItem getShortcutMenuItem(MenuShortcut s)	Gets the instance of MenuItem associated with the specified MenuShortcut object, or null if none of the menu items being managed by this menu bar is associated with the specified menu shortcut.
11	void remove(int index)	Removes the menu located at the specified index from this menu bar.
12	void remove(MenuComponent m)	Removes the specified menu component from this menu bar.
13	void removeNotify()	Removes the menu bar's peer.
14	Enumeration shortcuts()	Sets the specified menu to be this menu bar's help menu.
15	void setHelpMenu(Menu m)	Gets an enumeration of all menu shortcuts this menu bar is managing.

Methods inherited

This class inherits methods from the following classes:

- [java.awt.MenuComponent](#)
- [java.lang.Object](#)

AWT MenuItem Class

Introduction

The MenuBar class represents the actual item in a menu. All items in a menu should derive from class MenuItem, or one of its subclasses. By default, it embodies a simple labeled menu item.

Class declaration

Following is the declaration for java.awt.MenuItem class:

```
public class MenuItem extends MenuComponent implements Accessible
```

Class constructors

Sno	Constructor	Description
1	MenuItem()	Constructs a new MenuItem with an empty label and no keyboard shortcut.
2	MenuItem(String label)	Constructs a new MenuItem with the specified label and no keyboard shortcut.
3	MenuItem(String label, MenuShortcut s)	Create a menu item with an associated keyboard shortcut.

Class methods

Sno	Method	Description
1	void addActionListener(ActionListener l)	Adds the specified action listener to receive action events from this menu item.
2	void addNotify()	Creates the menu item's peer.
3	void deleteShortcut()	Delete any MenuShortcut object associated with this menu item.
4	void disable()	Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
5	protected void disableEvents(long eventsToDisable)	Disables event delivery to this menu item for events defined by the specified event mask parameter.
6	void enable()	Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
7	void enable(boolean b)	Deprecated. As of JDK version 1.1, replaced by setEnabled(boolean).
8	protected void enableEvents(long eventsToEnable)	Enables event delivery to this menu item for events to be defined by the specified event mask parameter.
9	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this MenuItem.
10	String getActionCommand()	Gets the command name of the action event that is fired by this menu item.
11	ActionListener[] getActionListeners()	Returns an array of all the action listeners registered on this menu item.
12	String getLabel()	Gets the label for this menu item.
13	EventListener[] getListeners(Class listenerType)	Returns an array of all the objects currently registered as FooListeners upon this MenuItem.
14	MenuShortcut getShortcut()	Get the MenuShortcut object associated with this menu item.

15	boolean isEnabled()	Checks whether this menu item is enabled.
16	String paramString()	Returns a string representing the state of this MenuItem.
17	protected void processActionEvent(ActionEvent e)	Processes action events occurring on this menu item, by dispatching them to any registered ActionListener objects.
18	protected void processEvent(AWTEvent e)	Processes events on this menu item.
19	void removeActionListener(ActionListener l)	Removes the specified action listener so it no longer receives action events from this menu item.
20	void setActionCommand(String command)	Sets the command name of the action event that is fired by this menu item.
21	void setEnabled(boolean b)	Sets whether or not this menu item can be chosen.
22	voidsetLabel(String label)	Sets the label for this menu item to the specified label.
23	void setShortcut(MenuShortcut s)	Set the MenuShortcut object associated with this menu item.

Methods inherited

This class inherits methods from the following classes:

- `java.awt.MenuComponent`
- `java.lang.Object`

AWT Menu Class

Introduction

The Menu class represents pull-down menu component which is deployed from a menu bar.

Class declaration

Following is the declaration for `java.awt.Menu` class:

```
public class Menu extends MenuItem implements MenuContainer, Accessible
```

Class constructors

Sno	Constructor	Description
1	Menu()	Constructs a new menu with an empty label.
2	Menu(String label)	Constructs a new menu with the specified label.
3	Menu(String label, boolean tearOff)	Constructs a new menu with the specified label, indicating whether the menu can be torn off.

Class methods

Sno	Method	Description
1	MenuItem add(MenuItem mi)	Adds the specified menu item to this menu.
2	void add(String label)	Adds an item with the specified label to this menu.
3	void addNotify()	Creates the menu's peer.
4	void addSeparator()	Adds a separator line, or a hyphen, to the menu at the current position.
5	int countItems()	Deprecated. As of JDK version 1.1, replaced by getItemCount().
6	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this Menu.
7	MenuItem getItem(int index)	Gets the item located at the specified index of this menu.
8	int getItemCount()	Get the number of items in this menu.
9	void insert(MenuItem menuitem, int index)	Inserts a menu item into this menu at the specified position.
10	void insert(String label, int index)	Inserts a menu item with the specified label into this menu at the specified position.
11	void insertSeparator(int index)	Inserts a separator at the specified position.
12	boolean isTearOff()	Indicates whether this menu is a tear-off menu.
13	String paramString()	Returns a string representing the state of this Menu.
14	void remove(int index)	Removes the menu item at the specified index from this menu.
15	void remove(MenuComponent item)	Removes the specified menu item from this menu.
16	void removeAll()	Removes all items from this menu.
17	void removeNotify()	Removes the menu's peer.

Methods inherited

This class inherits methods from the following classes:

- [java.awt.MenuItem](#)
- [java.awt.MenuComponent](#)
- [java.lang.Object](#)

AWT CheckboxMenuItem Class

Introduction

The CheckboxMenuItem class represents a check box which can be included in a menu. Selecting the check box in the menu changes control's state from on to off or from off to on.

Class declaration

Following is the declaration for java.awt.CheckboxMenuItem class:

```
public class CheckboxMenuItem extends MenuItem implements ItemSelectable, Accessible
```

Class constructors

Sno	Constructor	Description
1	CheckboxMenuItem()	Create a check box menu item with an empty label.
2	CheckboxMenuItem(label)	Create a check box menu item with the specified label.
3	CheckboxMenuItem(label, boolean state)	Create a check box menu item with the specified label and state.

Class methods

Sno	Method	Description
1	void addItemListener(ItemListener l)	Adds the specified item listener to receive item events from this check box menu item.
2	void addNotify()	Creates the peer of the checkbox item.
3	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this CheckboxMenuItem.
4	ItemListener[] getItemListeners()	Returns an array of all the item listeners registered on this checkbox menuitem.
5	<T extends EventListener> T[] getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as FooListeners upon this CheckboxMenuItem.
6	Object[] getSelectedObjects()	Returns the an array (length 1) containing the checkbox menu item label or null if the checkbox is not selected.
7	boolean getState()	Determines whether the state of this check box menu item is "on" or "off."
8	param()	Returns a representing the state of this CheckBoxMenuItem.
9	protected void processEvent(AWTEvent e)	Processes events on this check box menu item.
10	protected void processItemEvent(ItemEvent e)	Processes item events occurring on this check box menu item by dispatching them to any registered ItemListener objects.
11	void removeItemListener(ItemListener l)	Removes the specified item listener so that it no longer receives item events from this check box menu item.
12	void setState(boolean b)	Sets this check box menu item to the specified state.

Methods inherited

This class inherits methods from the following classes:

- java.awt.MenuItem
- java.awt.MenuComponent
- java.lang.Object

example program on all menu classes

```
import java.awt.*;
class Mfb extends Frame
{
    Mfb()
    {
        setBackground(Color.white);
        setVisible(true);
        setSize(500,400);
    }
}
class Mbar
{
    public static void main(String args[])
    {
        Mfb o=new Mfb();
        o.setTitle("Notepad");
        o.setLayout(new FlowLayout());
        MenuBar mb=new MenuBar();
        o.setMenuBar(mb);

        Menu m1,m2,m3,m4,m5;
        m1=new Menu("File");
        m2=new Menu("Edit");
        m3=new Menu("Format");
        m4=new Menu("View");
        m5=new Menu("Help");

        mb.add(m1);
        mb.add(m2);
        mb.add(m3);
        mb.add(m4);
        mb.add(m5);

        MenuItem mf1,mf2,mf3,mf4,mf5,mf6,mf7;
```

```
mf1=new MenuItem("New ");
mf2=new MenuItem("Open ");
mf3=new MenuItem("Save ");
mf4=new MenuItem("Save As ");
mf5=new MenuItem("Page Setup ");
mf6=new MenuItem("Print... ");
mf7=new MenuItem("Exit ");
```

```
m1.add(mf1);
m1.add(mf2);
m1.add(mf3);
m1.add(mf4);
m1.add(mf5);
m1.add(mf6);
m1.add(mf7);
```

```
MenuItem me1,me2,me3,me4,me5,me6,me7,me8,me9,me10,me11;
me1=new MenuItem("Undo");
me2=new MenuItem("Cut");
me3=new MenuItem("Copy");
me4=new MenuItem("Paste");
me5=new MenuItem("Delete");
me6=new MenuItem("Find");
me7=new MenuItem("Find Next");
me8=new MenuItem("Replace");
me9=new MenuItem("GoTo");
me10=new MenuItem("Select All");
me11=new MenuItem("Time/Date");
```

```
m2.add(me1);
m2.add(me2);
m2.add(me3);
m2.add(me4);
m2.add(me5);
m2.add(me6);
m2.add(me7);
m2.add(me8);
m2.add(me9);
m2.add(me10);
m2.add(me11);
```

```
MenuItem mt1,mt2;
mt1=new MenuItem("Word Wrap");
mt2=new MenuItem("Font");

m3.add(mt1);
m3.add(mt2);

MenuItem mv1;
mv1=new MenuItem("Status");

m4.add(mv1);

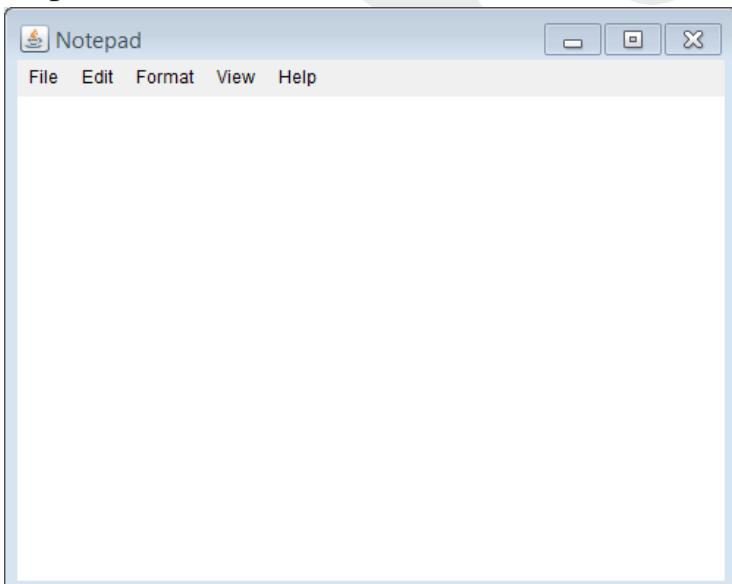
MenuItem mh1,mh2;
mh1=new MenuItem("View help");
mh2=new MenuItem("About Notepad");

m5.add(mh1);
m5.add(mh2);

}

}
```

Output:



AWT Scrollbar Class

Introduction

Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

Class declaration

Following is the declaration for java.awt.Scrollbar class:

```
public class Scrollbar extends Component implements Adjustable, Accessible
```

Field

Following are the fields for java.awt.Image class:

- **static int HORIZONTAL** --A constant that indicates a horizontal scroll bar.
- **static int VERTICAL** --A constant that indicates a vertical scroll bar.

Class constructors

Sno	Constructor	Description
1	Scrollbar()	Constructs a new vertical scroll bar.
2	Scrollbar(int orientation)	Constructs a new scroll bar with the specified orientation.
3	Scrollbar(int orientation, int value, int visible, int minimum, int maximum)	Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Class methods

Sno	Method	Description
1	void addAdjustmentListener(AdjustmentListener l)	Adds the specified adjustment listener to receive instances of AdjustmentEvent from this scroll bar.
2	void addNotify()	Creates the Scrollbar's peer.
3	int getBlockIncrement()	Gets the block increment of this scroll bar.
4	int getLineIncrement()	Deprecated. As of JDK version 1.1, replaced by getUnitIncrement().
5	int getMaximum()	Gets the maximum value of this scroll bar.
6	int getMinimum()	Gets the minimum value of this scroll bar.
7	int getOrientation()	Returns the orientation of this scroll bar.
8	int getPageIncrement()	Deprecated. As of JDK version 1.1, replaced by getBlockIncrement().
9	int getUnitIncrement()	Gets the unit increment for this scrollbar.
10	int getValue()	Gets the current value of this scroll bar.
11	Boolean getValueIsAdjusting()	Returns true if the value is in the process of changing as a result of actions being taken by the user.
12	int getVisible()	Deprecated. As of JDK version 1.1, replaced by getVisibleAmount().

13	int getVisibleAmount()	Gets the visible amount of this scroll bar.
14	protected String paramString()	Returns a string representing the state of this Scrollbar.
15	protected void processAdjustmentEvent(AdjustmentEvent e)	Processes adjustment events occurring on this scrollbar by dispatching them to any registered AdjustmentListener objects.
16	protected void processEvent(AWTEvent e)	Processes events on this scroll bar.
17	void removeAdjustmentListener(AdjustmentListener l)	Removes the specified adjustment listener so that it no longer receives instances of AdjustmentEvent from this scroll bar.
18	void setBlockIncrement(int v)	Sets the block increment for this scroll bar.
19	void setLineIncrement(int v)	Deprecated. As of JDK version 1.1, replaced by setUnitIncrement(int).
20	void setMaximum(int newMaximum)	Sets the maximum value of this scroll bar.
21	void setMinimum(int newMinimum)	Sets the minimum value of this scroll bar.
22	void setOrientation(int orientation)	Sets the orientation for this scroll bar.
23	void setPageIncrement(int v)	Deprecated. As of JDK version 1.1, replaced by setBlockIncrement().
24	void setUnitIncrement(int v)	Sets the unit increment for this scroll bar.
25	void setValue(int newValue)	Sets the value of this scroll bar to the specified value.
26	void setValueIsAdjusting(boolean b)	Sets the valueIsAdjusting property.
27	void setValues(int value, int visible, int minimum, int maximum)	Sets the values of four properties for this scroll bar: value, visibleAmount, minimum, and maximum.
28	void setVisibleAmount(int newAmount)	Sets the visible amount of this scroll bar.
29	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this Scrollbar.
30	AdjustmentListener[] getAdjustmentListeners()	Returns an array of all the adjustment listeners registered on this scrollbar.
31	<T extends EventListener>T[] getListeners(Class<T> listenerType)	Returns an array of all the objects currently registered as FooListeners upon this Scrollbar.

Methods inherited

This class inherits methods from the following classes:

- [java.awt.Component](#)
- [java.lang.Object](#)

AWT Graphics Class

Introduction

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

A Graphics object encapsulates all state information required for the basic rendering operations that Java supports. State information includes the following properties.

- The Component object on which to draw.
- A translation origin for rendering and clipping coordinates.
- The current clip.
- The current color.
- The current font.
- The current logical pixel operation function.
- The current XOR alternation color

Class declaration

Following is the declaration for java.awt.Graphics class:

```
public abstract class Graphics extends Object
```

Class constructors

Sno	Constructor	Description
1	Graphics()	Constructs a new Graphics object.

Class methods

Sno	Method	Description
1	abstract void clearRect(int x, int y, int width, int height)	Clears the specified rectangle by filling it with the background color of the current drawing surface.
2	abstract void clipRect(int x, int y, int width, int height)	Intersects the current clip with the specified rectangle.
3	abstract void copyArea(int x, int y, int width, int height, int dx, int dy)	Copies an area of the component by a distance specified by dx and dy.
4	abstract Graphics create()	Creates a new Graphics object that is a copy of this Graphics object.
5	Graphics create(int x, int y, int width, int height)	Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.
6	abstract void dispose()	Disposes of this graphics context and releases any system resources that it is using.
7	void draw3DRect(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
8	abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Draws the outline of a circular or elliptical arc covering the specified rectangle.
9	void drawBytes(byte[] data, int offset, int length, int x, int y)	Draws the text given by the specified byte array, using this graphics context's current font and color.
10	void drawChars(char[] data, int offset,	Draws the text given by the specified character array,

	int length, int x, int y)	using this graphics context's current font and color.
11	abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)	Draws as much of the specified image as is currently available.
12	abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)	Draws as much of the specified image as is currently available.
13	abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
14	abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
15	abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
16	abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
17	abstract void drawLine(int x1, int y1, int x2, int y2)	Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
18	abstract void drawOval(int x, int y, int width, int height)	Draws the outline of an oval.
19	abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)	Draws a closed polygon defined by arrays of x and y coordinates.
20	void drawPolygon(Polygon p)	Draws the outline of a polygon defined by the specified Polygon object.
21	abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)	Draws a sequence of connected lines defined by arrays of x and y coordinates.
22	void drawRect(int x, int y, int width, int height)	Draws the outline of the specified rectangle.
23	abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Draws an outlined round-cornered rectangle using this graphics context's current color.
24	abstract void drawString(AttributedCharacterIterator iterator, int x, int y)	Renders the text of the specified iterator applying its attributes in accordance with the specification of the TextAttribute class.
25	abstract void drawString(String str, int x, int y)	Draws the text given by the specified string, using this graphics context's current font and color.
26	void fill3DRect(int x, int y, int width, int height, boolean raised)	Paints a 3-D highlighted rectangle filled with the current color.
27	abstract void fillArc(int x, int y, int width, int height, int startAngle, int	Fills a circular or elliptical arc covering the specified rectangle.

	arcAngle)	
28	abstract void fillOval(int x, int y, int width, int height)	Fills an oval bounded by the specified rectangle with the current color.
29	abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)	Fills a closed polygon defined by arrays of x and y coordinates.
30	void fillPolygon(Polygon p)	Fills the polygon defined by the specified Polygon object with the graphics context's current color.
31	abstract void fillRect(int x, int y, int width, int height)	Fills the specified rectangle.
32	abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Fills the specified rounded corner rectangle with the current color.
33	void finalize()	Disposes of this graphics context once it is no longer referenced.
34	abstract Shape getClip()	Gets the current clipping area.
35	abstract Rectangle getClipBounds()	Returns the bounding rectangle of the current clipping area.
36	Rectangle getClipBounds(Rectangle r)	Returns the bounding rectangle of the current clipping area.
37	Rectangle getClipRect()	Deprecated. As of JDK version 1.1, replaced by getClipBounds().
38	abstract Color getColor()	Gets this graphics context's current color.
39	abstract Font getFont()	Gets the current font.
40	FontMetrics getFontMetrics()	Gets the font metrics of the current font.
41	abstract FontMetrics getFontMetrics(Font f)	Gets the font metrics for the specified font.
42	boolean hitClip(int x, int y, int width, int height)	Returns true if the specified rectangular area might intersect the current clipping area
43	abstract void setClip(int x, int y, int width, int height)	Sets the current clip to the rectangle specified by the given coordinates.
44	abstract void setClip(Shape clip)	Sets the current clipping area to an arbitrary clip shape.
45	abstract void setColor(Color c)	Sets this graphics context's current color to the specified color.
46	abstract void setFont(Font font)	Sets this graphics context's font to the specified font.
47	abstract void setPaintMode()	Sets the paint mode of this graphics context to overwrite the destination with this graphics context's current color.
48	abstract void setXORMode(Color c1)	Sets the paint mode of this graphics context to alternate between this graphics context's current color

		and the new specified color.
49	String toString()	Returns a String object representing this Graphics object's value.
50	abstract void translate(int x, int y)	Translates the origin of the graphics context to the point (x, y) in the current coordinate system.

Methods inherited

This class inherits methods from the following classes:

- [java.lang.Object](#)

SWINGS

Introduction:

Java swing is a part of java foundation classes which are used to create window based applications which is written on top of AWT(Abstract window toolkit), API and entirely written in java. Unlike AWT, java swings provide platform independent & light weight components.

Inorder to use java swing components we have to use or we have to import javax.swing package which provides classes for java swing API such as JButton,JTextField,JColorChooser etc

JFC(java foundation classes)

JFC are set of GUI components which simplifies the development of desktop applications .Using java programming language JFC are prewritten code in the form class libraries that give programmers a wide range of your GUI component.

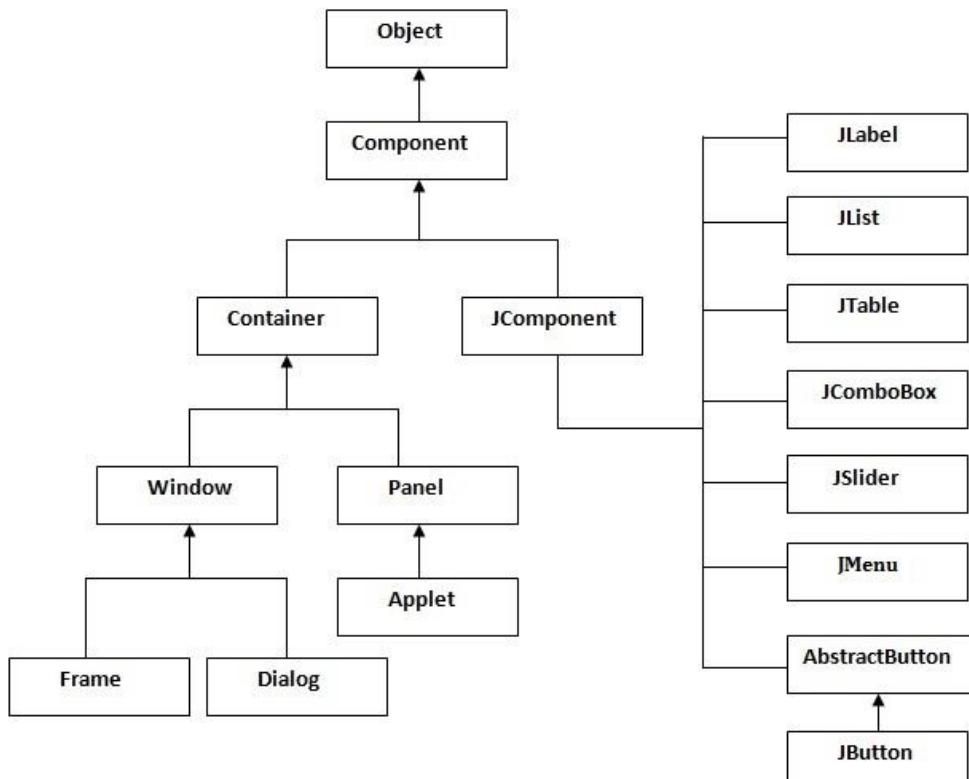
Sun microsystem introduce AWT components to prepare GUI application .

An alternative to AWT called JFC was developed by netscape communications

Differences between AWT & java swing

Java AWT	Java swing
• AWT stands for abstract window toolkit	• Swings are java foundation classes
• Heavy weight components	• Light weight components
• It requires java.awt packages	• It requires javax.swing package
• AWT components are platform dependent	• Swing components are platform independent
• AWT components are less in number When compare to swing components	• Swings has many more advanced components like JTable, JScrollPanes, JColorChooser..etc
• AWT does not follow MVC(model view controller) architecture	• Swing follows MVC(model view controller) architecture.
• In AWT we will add components directly to the frame	• In swings we will add all the components to panes

Java swing class hierarchy



Class constructors

sno	classname	description
1	component	a component is an abstract base class for non menu user interface controls of swing.
2.	conatiner	a container is a component that can contain or can be used to store other swing components
3.	JComponent	JComponent is a base class for all swing user interface components

Container

There are 2 ways to create a frame

- 1) by creating the object of frame class
- 2) by extending frame class

Swing user interface elements

sno	classname	description
1	JLabel	A JLabel object is a component for placing text in a container
2	JButton	JButton class is used to create labeled buttons
3	JCheckbox	JCheckbox which is used to create a graphical component that can be either true(checked) or false(uncheck)
4	JRadioButton	JRadioButton which is used to create a graphical component that can be either true(checked) or false(uncheck) in a group

5	JList	A JList components presents the user with a scrolling list of text items
6	JTextField	A JTextField object is a text components that allows for editing single line of text
7	JTextArea	A JTextArea object is a text components that allows for editing multiple lines of text.
8	JPasswordField	A JPasswordField object is a text component specialized for passwoed entry.
9	JComboBox	A JComboBoxcomponents presents the user with a menu of choices
10	JColorChooser	A JColorChooser provides a pane of control design to allow a user to manipulate & select a color
11	JScrollbar	A JScrollbar control represents the scrollbar inorder to enable the user to select from range of values.
12	JOptionPane	A JOptionPane provides a set of standard dialog boxes that prompts user for a value or informs to do something.
13	JFileChooser	A JFileChoosercontrol represents a dialog window from which the user can select a file.



JButton

Introduction:

The class **JButton** is an implementation of a push button. This component has a label and generates an event when pressed. It can also have an Image.

Class Declaration

Following is the declaration for **javax.swing.JButton** class –

Public class JButton extends AbstractButton implements Accessible

Class Constructors

Sno	Constructor	Description
1	JButton()	Creates a button with no set text or icon.
2	JButton(Action a)	Creates a button where properties are taken from the Action supplied.
3	JButton(Icon icon)	Creates a button with an icon.
4	JButton(String text)	Creates a button with the text.
5	JButton(String text, Icon icon)	Creates a button with an initial text and an icon.

Class Methods

Sno	Methods	Description
1.	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JButton.
2	String getUIClassID()	Returns a string that specifies the name of the L&F class which renders this component.
3	boolean isDefaultButton()	Gets the value of the defaultButton property, which if true means that this button is the current default button for its JRootPane.
4	boolean isDefaultCapable()	Gets the value of the defaultCapable property.
5	protected String paramString()	Returns a string representation of this JButton.
6	void removeNotify()	Overrides JComponent.removeNotify to check if this button is currently set as the default button on the RootPane. And if so, sets the RootPane's default button to null to ensure the RootPane doesn't hold onto an invalid button reference.
7	void setDefaultCapable(boolean defaultCapable)	Sets the defaultCapable property, which determines whether this button can be made the default button for its root pane
8	void updateUI()	Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.AbstractButton
- javax.swing.JComponent
- java.awt.Container

- java.awt.Component
- java.lang.Object

Example on JButton program:

```
import javax.swing.*;  
  
public class ButtonExample  
{  
  
    public static void main(String[] args)  
    {  
  
        JFrame f=new JFrame("Button Example");  
  
        JButton b=new JButton("Click Here");  
  
        b.setBounds(50,100,95,30);  
  
        f.add(b);  
  
        f.setSize(400,400);  
  
        f.setLayout(null);  
  
        f.setVisible(true);  
  
    }  
  
}
```

Output:

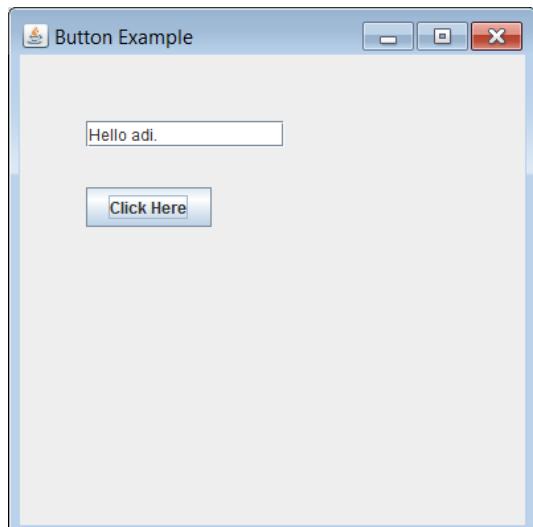


Example program on JButton with ActionListener:

```
import java.awt.event.*;  
import javax.swing.*;  
public class ButtonExample  
{
```

```
public static void main(String[] args)
{
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    b.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            tf.setText("Hello adi.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



Example program on displaying image on JButton:

```
import javax.swing.*;
public class ButtonExample
{
    ButtonExample()
    {
        JFrame f=new JFrame("Button Example");
```

```
 JButton b=new JButton(new ImageIcon("D:\\icon.png"));

b.setBounds(100,100,100, 40);

f.add(b);

f.setSize(300,400);

f.setLayout(null);

f.setVisible(true);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

public static void main(String[] args)

{

    new ButtonExample();

}

}

}

Output:
```



JLabel

Introduction

The class **JLabel** can display either text, an image, or both. Label's contents are aligned by setting the vertical and horizontal alignment in its display area. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

Class Declaration

Following is the declaration for **javax.swing.JLabel** class –

```
public class JLabel extends JComponent implements SwingConstants, Accessible
```

Field

Following are the fields for **javax.swing.JLabel** class –

```
protected Component labelFor
```

Class Constructors

Sno	Constructor	Description
1	JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
2	JLabel(Icon image)	Creates a JLabel instance with the specified image.
3	JLabel(Icon image, int horizontalAlignment)	Creates a JLabel instance with the specified image and horizontal alignment.
4	JLabel(String text)	Creates a JLabel instance with the specified text.
5	JLabel(String text, Icon icon, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.
6	JLabel(String text, int horizontalAlignment)	Creates a JLabel instance with the specified text and horizontal alignment.

Class Methods

Sno	Method	Description
1	protected int checkHorizontalKey(int key, String message)	Verify that key is a legal value for the horizontalAlignment properties.
2	protected int checkVerticalKey(int key, String message)	Verify that key is a legal value for the verticalAlignment or verticalTextPosition properties.
3	AccessibleContext getAccessibleContext()	Get the AccessibleContext of this object.
4	Icon getDisabledIcon()	Returns the icon used by the label when it's disabled.
5	int getDisplayedMnemonic()	Return the keycode that indicates a mnemonic key.
6	int getDisplayedMnemonicIndex()	Returns the character, as an index, that the look and feel should provide decoration for as representing the mnemonic character.
7	int getHorizontalAlignment()	Returns the alignment of the label's contents along the X axis.

8	int getHorizontalTextPosition()	Returns the horizontal position of the label's text, relative to its image.
9	Icon getIcon()	Returns the graphic image (glyph, icon) that the label displays.
10	int getIconTextGap()	Returns the amount of space between the text and the icon displayed in this label.
11	Component getLabelFor()	Get the component this is labelling.
12	String getText()	Returns the text string that the label displays.
13	LabelUI getUI()	Returns the L&F object that renders this component.
14	String getUIClassID()	Returns a string that specifies the name of the l&f class which renders this component.
15	int getVerticalAlignment()	Returns the alignment of the label's contents along the Y axis.
16	int getVerticalTextPosition()	Returns the vertical position of the label's text, relative to its image.
17	boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h)	This is overridden to return false if the current Icon's image is not equal to the passed in Image img.
18	protected String paramString()	Returns a string representation of this JLabel.
19	void setDisabledIcon(Icon disabledIcon)	Sets the icon to be displayed if this JLabel is "disabled" (JLabel.setEnabled(false)).
20	void setDisplayedMnemonic(char aChar)	Specifies the displayedMnemonic as a char value.
21	void setDisplayedMnemonic(int key)	Specifies a keycode that indicates a mnemonic key.
22	void setDisplayedMnemonicIndex(int index)	Provides a hint to the look and feel as to which character in the text should be decorated to represent the mnemonic.
23	void setHorizontalAlignment(int alignment)	Sets the alignment of the label's contents along the X axis.
24	void setHorizontalTextPosition(int textPosition)	Sets the horizontal position of the label's text, relative to its image.
25	void setIcon(Icon icon)	Defines the icon this component will display.
26	void setIconTextGap(int iconTextGap)	If both the icon and text properties are set, this property defines the space between them.
27	void setLabelFor(Component c)	Sets the component, this is labelling.
28	void setText(String text)	Defines the single line of text this component will display.
29	void setUI(LabelUI ui)	Sets the L&F object that renders this component
30	void setVerticalAlignment(int alignment)	Sets the alignment of the label's contents along the Y axis.
31	void setVerticalTextPosition(int textPosition)	Sets the vertical position of the label's text, relative to its image.

32	void updateUI()	Resets the UI property to a value from the current look and feel.
----	------------------------	---

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Example program on JLabel

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:



JTextField

Introduction

The class **JTextField** is a component which allows the editing of a single line of text.

Class Declaration

Following is the declaration for **javax.swing.JTextField** class –

```
public class JTextField extends JTextComponent implements SwingConstants
```

Field

Following are the fields for **javax.swing.JList** class –

- **static String notifyAction** – Name of the action to send notification that the contents of the field have been accepted.

Class Constructors

Sno	Constructor	Description
1	JTextField()	Constructs a new TextField.
2	JTextField(Document doc, String text, int columns)	Constructs a new JTextField that uses the given text storage model and the given number of columns.
3	JTextField(int columns)	Constructs a new empty TextField with the specified number of columns.
4	JTextField(String text)	Constructs a new TextField initialized with the specified text.
5	JTextField(String text, int columns)	Constructs a new TextField initialized with the specified text and columns.

Class Methods

Sno	Method	Description
1	protected void actionPropertyChanged(Action action, String propertyName)	Updates thetextfield's state in response to property changes in associated action.
2	void addActionListener(ActionListener l)	Adds the specified action listener to receive action events from this textfield.
3	protected void configurePropertiesFromAction(Action a)	Sets the properties on this textfield to match those in the specified Action.
4	protected PropertyChangeListener createActionPropertyChangeListener(Action a)	Creates and returns a PropertyChangeListener that is responsible for listening for changes from the specified Action and updating the appropriate properties.
5	protected Document createDefaultModel()	Creates the default implementation of the model to be used at construction if one isn't explicitly given.
6	protected void fireActionPerformed()	Notifies all listeners that have registered interest for notification on this event type.
7	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JTextField.

8	Action getAction()	Returns the currently set Action for this ActionEvent source, or null if no Action is set.
9	ActionListener[] getActionListeners()	Returns an array of all the ActionListeners added to this JTextField with addActionListener().
10	Action[] getActions()	Fetches the command list for the editor.
11	int getColumns()	Returns the number of columns in this TextField.
12	protected int getColumnWidth()	Returns the column width.
13	int getHorizontalAlignment()	Returns the horizontal alignment of the text.
14	BoundedRangeModel getHorizontalVisibility()	Gets the visibility of the text field.
15	Dimension getPreferredSize()	Returns the preferred size Dimensions needed for this TextField.
16	int getScrollOffset()	Gets the scroll offset, in pixels.
17	String getUIClassID()	Gets the class ID for a UI.
18	boolean isValidateRoot()	Calls to revalidate that come from within the textfield itself will be handled by validating the textfield, unless the textfield is contained within a JViewport, in which case this returns false.
19	protected String paramString()	Returns a string representation of this JTextField.
20	void postActionEvent()	Processes action events occurring on this textfield by dispatching them to any registered ActionListener objects.
21	void removeActionListener(ActionListener l)	Removes the specified action listener so that it no longer receives action events from this textfield.
22	void scrollRectToVisible(Rectangle r)	Scrolls the field left or right.
23	void setAction(Action a)	Sets the Action for the ActionEvent source.
24	void setActionCommand(String command)	Sets the command string used for action events.
25	void setColumns(int columns)	Sets the number of columns in this TextField, and then invalidate the layout.
26	void setDocument(Document doc)	Associates the editor with a text document.
27	void setFont(Font f)	Sets the current font.
28	void setHorizontalAlignment(int alignment)	Sets the horizontal alignment of the text.
29	void setScrollOffset(int scrollOffset)	Sets the scroll offset, in pixels.

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component

- java.lang.Object

Example program on JTextField

```
import javax.swing.*;  
  
class TextFieldExample  
{  
  
    public static void main(String args[])  
    {  
  
        JFrame f= new JFrame();  
  
        JTextField t1,t2;  
  
        t1=new JTextField("adi.");  
  
        t1.setBounds(50,100, 200,30);  
  
        t2=new JTextField("hi");  
  
        t2.setBounds(50,150, 200,30);  
  
        f.add(t1); f.add(t2);  
  
        f.setSize(400,400);  
  
        f.setLayout(null);  
  
        f.setVisible(true);  
  
    }  
}
```

Example program on JTextField with ActionListener

```
import javax.swing.*;  
  
import java.awt.event.*;  
  
public class TextFieldExample implements ActionListener  
{  
  
    JTextField tf1,tf2,tf3;  
  
    JButton b1,b2;  
  
    TextFieldExample()  
    {
```

```
JFrame f= new JFrame();  
  
tf1=new JTextField();  
  
tf1.setBounds(50,50,150,20);  
  
tf2=new JTextField();  
  
tf2.setBounds(50,100,150,20);  
  
tf3=new JTextField();  
  
tf3.setBounds(50,150,150,20);  
  
tf3.setEditable(false);  
  
b1=new JButton("+");  
  
b1.setBounds(50,200,50,50);  
  
b2=new JButton("-");  
  
b2.setBounds(120,200,50,50);  
  
b1.addActionListener(this);  
  
b2.addActionListener(this);  
  
f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);  
  
f.setSize(300,300);  
  
f.setLayout(null);  
  
f.setVisible(true);  
  
}  
  
public void actionPerformed(ActionEvent e)  
{  
  
String s1=tf1.getText();  
  
String s2=tf2.getText();  
  
int a=Integer.parseInt(s1);  
  
int b=Integer.parseInt(s2);  
  
int c=0;  
  
if(e.getSource()==b1)  
{
```

```
c=a+b;  
}  
  
else if(e.getSource()==b2)  
{  
  
c=a-b;  
}  
  
String result=String.valueOf(c);  
  
tf3.setText(result);  
}  
  
public static void main(String[] args)  
{  
  
new TextFieldExample();  
}  
}
```

SWING - JPasswordField Class

Introduction

The class JPasswordField is a component which is specialized to handle password functionality and allows the editing of a single line of text.

Class Declaration

Following is the declaration for javax.swing.JPasswordField class –

```
public class JPasswordField extends JTextField
```

Class constructors

Sno	Constructor	Description
1	JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
2	JPasswordField(Document doc, String txt, int columns)	Constructs a new JPasswordField that uses the given text storage model and the given number of columns.
3	JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
4	JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
5	JPasswordField(String text, int columns)	Constructs a new JPasswordField initialized with the specified text and columns.

Class Methods

Sno	Method	Description
1	void copy()	Invokes provideErrorFeedback on the current look and feel, which typically initiates an error beep.
2	void cut()	Invokes provideErrorFeedback on the current look and feel, which typically initiates an error beep.
3	boolean echoCharIsSet()	Returns true if this JPasswordField has a character set for echoing.
4	AccessibleContext getAccessibleContext()	Returns the AccessibleContext associated with this JPasswordField.
5	char getEchoChar()	Returns the character to be used for echoing.
6	char[] getPassword()	Returns the text contained in this TextComponent.
7	String getText()	Deprecated. As of Java 2 platform v1.2, replaced by getPassword.
8	String getText(int offs, int len)	Deprecated. As of Java 2 platform v1.2, replaced by getPassword.
9	String getUIClassID()	Returns the name of the L&F class that renders this component.
10	protected String paramString()	Returns a string representation of this JPasswordField.
11	void setEchoChar(char c)	Sets the echo character for this JPasswordField.

12	void updateUI()	Reloads the pluggable UI.
----	------------------------	---------------------------

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.JTextField
- javax.swing.text.JTextComponent
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Example program on JPasswordField

```
import javax.swing.*;
public class PasswordFieldExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Password Field Example");
        JPasswordField value = new JPasswordField();
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

SWING - JColorChooser Class

Introduction:

The class JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.

Class Declaration

Following is the declaration for javax.swing.JColorChooser class –

```
public class JColorChooser extends JComponent implements Accessible
```

Field

Following are the fields for javax.swing.JLabel class –

- **protected AccessibleContext accessibleContext**
- **static String CHOOSEN_PANELS_PROPERTY** – The chooserPanel array property name.
- **static String PREVIEW_PANEL_PROPERTY** – The preview panel property name.
- **static String SELECTION_MODEL_PROPERTY** – The selection model property name.

Class Constructors

Sno	Constructor	Description
1	JColorChooser()	Creates a color chooser pane with an initial color of white.
2	JColorChooser(Color initialColor)	Creates a color chooser pane with the specified initial color.
3	JColorChooser(ColorSelectionModel model)	Creates a color chooser pane with the specified ColorSelectionModel.

Class Methods

Sno	Method	Description
1	void addChooserPanel(AbstractColorChooserPanel panel)	Adds a color chooser panel to the color chooser.
2	static JDialog createDialog(Component c, String title, boolean modal, JColorChooser chooserPane, ActionListener okListener, ActionListener cancelListener)	Creates and returns a new dialog containing the specified ColorChooser pane along with "OK", "Cancel", and "Reset" buttons.
3	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JColorChooser.
4	AbstractColorChooserPanel[] getChooserPanels()	Returns the specified color panels.
5	Color getColor()	Gets the current color value from the color chooser.
6	boolean getDragEnabled()	Gets the value of the dragEnabled property.
7	JComponent getPreviewPanel()	Returns the preview panel that shows a chosen color.
8	ColorSelectionModel getSelectionModel()	Returns the data model that handles color selections.

9	ColorChooserUI getUI()	Returns the L&F object that renders this component.
10	String getUIClassID()	Returns the name of the L&F class that renders this component.
11	protected String paramString()	Returns a string representation of this JColorChooser.
12	AbstractColorChooserPanel removeChooserPanel(AbstractColorChooserPanel panel)	Removes the Color Panel specified.
13	void setChooserPanels(AbstractColorChooserPanel[] panels)	Specifies the Color Panels used to choose a color value.
14	void setColor(Color color)	Sets the current color of the color chooser to the specified color.
15	void setColor(int c)	Sets the current color of the color chooser to the specified color.
16	void setColor(int r, int g, int b)	Sets the current color of the color chooser to the specified RGB color.
17	void setDragEnabled(boolean b)	Sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
18	void setPreviewPanel(JComponent preview)	Sets the current preview panel.
19	void setSelectionModel(ColorSelectionModel newModel)	Sets the model containing the selected color.
20	void setUI(ColorChooserUI ui)	Sets the L&F object that renders this component.
21	static Color showDialog(Component component, String title, Color initialColor)	Shows a modal color-chooser dialog and blocks until the dialog is hidden.
22	void updateUI()	Notification from the UIManager that the L&F has changed.

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Example program on JColorChooser

```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener
{

```

```

 JButton b;
 Container c;
 ColorChooserExample()
 {
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
 }

 public void actionPerformed(ActionEvent e)
 {
    Color initialcolor=Color.RED;
    Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
    c.setBackground(color);
 }

 public static void main(String[] args)
 {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
 }

}

```

Example program on JColorChooser with ActionListener

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ColorChooserExample extends JFrame implements ActionListener
{
    JFrame f;
    JButton b;
    JTextArea ta;
    ColorChooserExample()
    {
        f=new JFrame("Color Chooser Example.");
        b=new JButton("Pad Color");

```

```
b.setBounds(200,250,100,30);
ta=new JTextArea();
ta.setBounds(10,10,300,200);
b.addActionListener(this);
f.add(b);f.add(ta);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);

}

public void actionPerformed(ActionEvent e)
{
    Color c=JColorChooser.showDialog(this,"Choose",Color.CYAN);
    ta.setBackground(c);
}

public static void main(String[] args)
{
    new ColorChooserExample();
}

}
```

SWING - JRadioButton Class

Introduction

The class JRadioButton is an implementation of a radio button - an item that can be selected or deselected, and which displays its state to the user.

Class Declaration

Following is the declaration for javax.swing.JRadioButton class –

```
public class JRadioButton extends JToggleButton implements Accessible
```

Class Constructors

Sno	Constructor	Description
1	JRadioButton()	Creates an initially unselected radio button with no set text.
2	JRadioButton(Action a)	Creates a radiobutton where properties are taken from the Action supplied.
3	JRadioButton(Icon icon)	Creates an initially unselected radio button with the specified image but no text
4	JRadioButton(Icon icon, boolean selected)	Creates a radio button with the specified image and selection state, but no text.
5	JRadioButton(String text, boolean selected)	Creates a radio button with the specified text and selection state.
6	JRadioButton(String text, Icon icon)	Creates a radio button that has the specified text and image, and which is initially unselected.
7	JRadioButton(String text, Icon icon, boolean selected)	Creates a radio button that has the specified text, image, and selection state.

Class Methods

Sno	Method	Description
1	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JRadioButton.
2	String getUIClassID()	Returns the name of the L&F class that renders this component.
3	protected String paramString()	Returns a string representation of this JRadioButton.
4	void updateUI()	Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes –

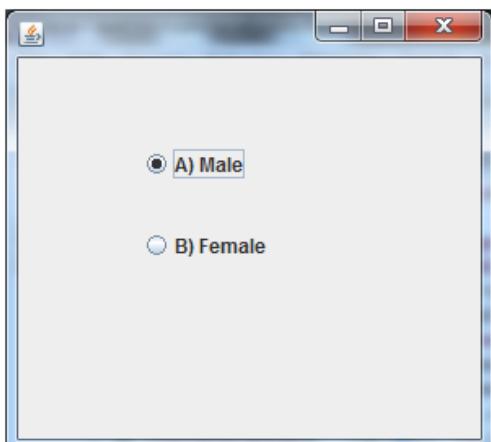
- javax.swing.AbstractButton
- javax.swing.JToggleButton
- javax.swing.JComponent
- java.awt.Container

- java.awt.Component
- java.lang.Object

Example program on JRadioButton:

```
import javax.swing.*;
public class RadioButtonExample
{
    JFrame f;
    RadioButtonExample()
    {
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);
        f.add(r1);f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
        new RadioButtonExample();
    }
}
```

Output:

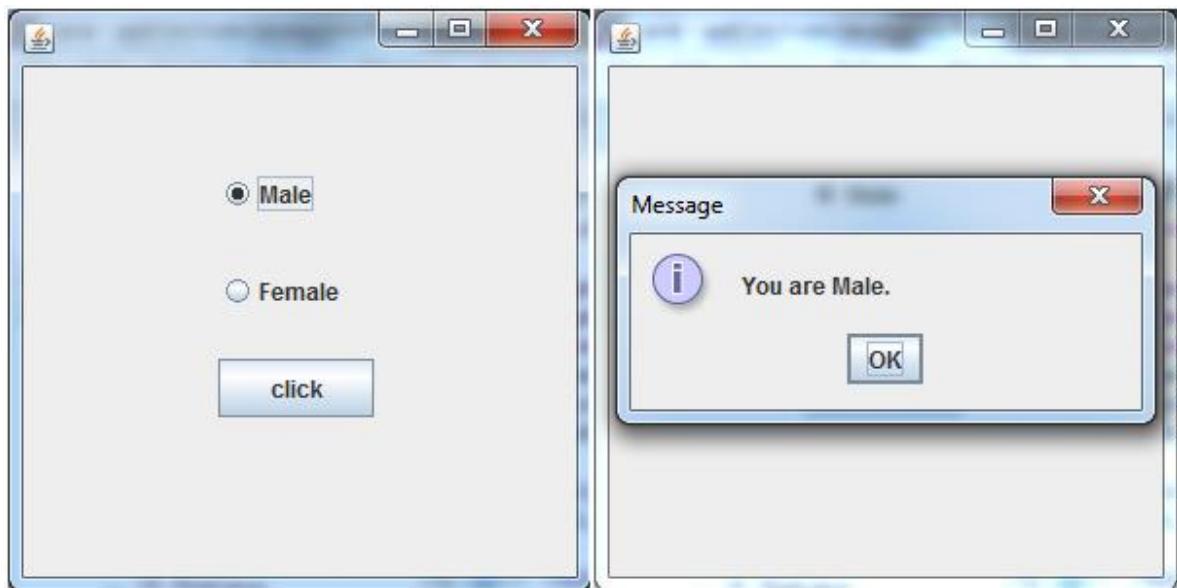


Example program on JRadioButton with ActionListener:

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener
{
    JRadioButton rb1,rb2;
    JButton b;
    RadioButtonExample()
    {
        rb1=new JRadioButton("Male");
        rb1.setBounds(100,50,100,30);
        rb2=new JRadioButton("Female");
        rb2.setBounds(100,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);bg.add(rb2);
        b=new JButton("click");
        b.setBounds(100,150,80,30);
        b.addActionListener(this);
        add(rb1);add(rb2);add(b);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(rb1.isSelected())
        {
            JOptionPane.showMessageDialog(this,"You are Male.");
        }
        if(rb2.isSelected())
        {
            JOptionPane.showMessageDialog(this,"You are Female.");
        }
    }
    public static void main(String args[])
    {
        new RadioButtonExample();
    }
}
```

}

Output:



SWING - JProgressBar Class

Introduction

The class JProgressBar is a component which visually displays the progress of some task.

Class Declaration

Following is the declaration for javax.swing.JProgressBar class –

```
public class JProgressBar extends JComponent implements SwingConstants, Accessible
```

Field

Following are the fields for javax.swing.JProgressBar class –

- **protected ChangeEvent changeEvent** – Only one ChangeEvent is needed per instance since the event's only interesting property is the immutable source, which is the progress bar.
- **protected ChangeListener changeListener** – Listens for change events sent by the progress bar's model, redispaching them to change-event listeners registered upon this progress bar.
- **protected BoundedRangeModel model** – The object that holds the data for the progress bar.
- **protected int orientation** – Whether the progress bar is horizontal or vertical.
- **protected boolean paintBorder** – Whether to display a border around the progress bar.
- **protected boolean paintString** – Whether to display a string of text on the progress bar.
- **protected String progressString** – An optional string that can be displayed on the progress bar.

Class Constructors

Sno	Constructor	Description
1	JProgressBar()	Creates a horizontal progress bar that displays a border but no progress string.
2	JProgressBar(BoundedRangeModel newModel)	Creates a horizontal progress bar that uses the specified model to hold the progress bar's data.
3	JProgressBar(int orient)	Creates a progress bar with the specified orientation, which can be either SwingConstants.VERTICAL or SwingConstants.HORIZONTAL.
4	JProgressBar(int min, int max)	Creates a horizontal progress bar with the specified minimum and maximum.
5	JProgressBar(int orient, int min, int max)	Creates a progress bar using the specified orientation, minimum, and maximum.

Class Methods

Sno	Method	Description
1	void addChangeListener(ChangeListener l)	Adds the specified ChangeListener to the progress bar.
2	protected ChangeListener createChangeListener()	Subclasses that want to handle change events from the model differently can override this to return an instance of a custom ChangeListener

		implementation.
3	protected void fireStateChanged()	Send a ChangeEvent, whose source is JProgressBar, to all ChangeListeners that have registered interest in ChangeEvents.
4	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this JProgressBar.
5	ChangeListener[] getChangeListeners()	Returns an array of all the ChangeListeners added to this progress bar with addChangeListener.
6	int getMaximum()	Returns the progress bar's maximum value from the BoundedRangeModel.
7	int getMinimum()	Returns the progress bar's minimum value from the BoundedRangeModel.
8	BoundedRangeModel getModel()	Returns the data model used by this progress bar.
9	int getOrientation()	Returns SwingConstants.VERTICAL or SwingConstants.HORIZONTAL, depending on the orientation of the progress bar.
10	double getPercentComplete()	Returns the percent complete for the progress bar.
11	String getString()	Returns a String representation of the current progress.
12	ProgressBarUI getUI()	Returns the look-and-feel object that renders this component.
13	String getUIClassID()	Returns the name of the look-and-feel class that renders this component.
14	int getValue()	Returns the progress bar's current value from the BoundedRangeModel.
15	boolean isBorderPainted()	Returns the borderPainted property.
16	boolean isIndeterminate()	Returns the value of the indeterminate property.
17	boolean isStringPainted()	Returns the value of the stringPainted property.
18	protected void paintBorder(Graphics g)	Paints the progress bar's border if the borderPainted property is true.
19	protected String paramString()	Returns a string representation of this JProgressBar.
20	void removeChangeListener(ChangeListener l)	Removes a ChangeListener from the progress bar.
21	void setBorderPainted(boolean b)	Sets the borderPainted property, which is true if the progress bar should paint its border.
22	void setIndeterminate(boolean newValue)	Sets the indeterminate property of the progress bar, which determines whether the progress bar is in determinate or indeterminate mode.
23	void setMaximum(int n)	Sets the progress bar's maximum value (stored in the progress bar's data model) to n.
24	void setMinimum(int n)	Sets the progress bar's minimum value (stored in the progress bar's data model) to n.
25	void setModel(BoundedRangeModel newModel)	Sets the data model used by the JProgressBar.

26	void setOrientation(int newOrientation)	Sets the progress bar's orientation to newOrientation, which must be SwingConstants.VERTICAL or SwingConstants.HORIZONTAL.
27	void setString(String s)	Sets the value of the progress string.
28	void setStringPainted(boolean b)	Sets the value of the stringPainted property, which determines whether the progress bar should render a progress string.
29	void setUI(ProgressBarUI ui)	Sets the look-and-feel object that renders this component.
30	void setValue(int n)	Sets the progress bar's current value to n.
31	void updateUI()	Resets the UI property to a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Example program on JProgressBar:

```

import javax.swing.*;
public class ProgressBarExample extends JFrame
{
    JProgressBar jb;
    int i=0,num=0;
    ProgressBarExample()
    {
        jb=new JProgressBar(0,2000);
        jb.setBounds(40,40,160,30);
        jb.setValue(0);
        jb.setStringPainted(true);
        add(jb);
        setSize(250,150);
        setLayout(null);
    }
}

```

```
}

public void iterate()
{
    while(i<=2000)
    {
        jb.setValue(i);

        i=i+20;

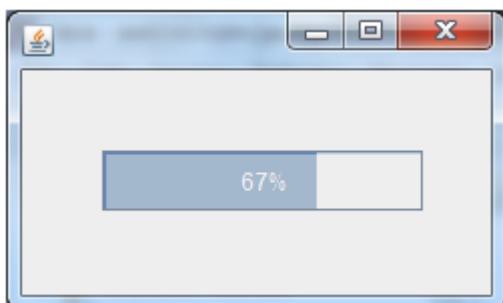
        try
        {
            Thread.sleep(150);
        }
        catch(Exception e)
        {
        }
    }
}

public static void main(String[] args)
{
    ProgressBarExample m=new ProgressBarExample();

    m.setVisible(true);

    m.iterate();
}
}
```

Output:



SWING - ImageIcon Class

Introduction

The class ImageIcon is an implementation of the Icon interface that paints Icons from Images.

Class Declaration

Following is the declaration for javax.swing.ImageIcon class –

```
public class ImageIcon extends Object implements Icon, Serializable, Accessible
```

Field

Following are the fields for javax.swing.ImageIcon class –

- protected static Component component
- protected static MediaTracker tracker

Class Constructors

Sno	Constructor	Description
1	ImageIcon()	Creates an uninitialized image icon.
2	ImageIcon(byte[] imageData)	Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
3	ImageIcon(byte[] imageData, String description)	Creates an ImageIcon from an array of bytes which were read from an image file containing a supported image format, such as GIF, JPEG, or (as of 1.3) PNG.
4	ImageIcon(Image image)	Creates an ImageIcon from an image object.
5	ImageIcon(Image image, String description)	Creates an ImageIcon from the image.
6	ImageIcon(String filename)	Creates an ImageIcon from the specified file.
7	ImageIcon(String filename, String description)	Creates an ImageIcon from the specified file.
8	ImageIcon(URL location)	Creates an ImageIcon from the specified URL.
9	ImageIcon(URL location, String description)	Creates an ImageIcon from the specified URL.

Class Methods

Sno	Method	Description
1	AccessibleContext getAccessibleContext()	Gets the AccessibleContext associated with this ImageIcon.

2	String getDescription()	Gets the description of the image.
3	int getIconHeight()	Gets the height of the icon.
4	int getIconWidth()	Gets the width of the icon.
5	Image getImage()	Returns this icon's Image.
6	int getImageLoadStatus()	Returns the status of the image loading operation.
7	ImageObserver getImageObserver()	Returns the image observer for the image.
8	protected void loadImage(Image image)	Loads the image, returning only when the image is loaded.
9	void paintIcon(Component c, Graphics g, int x, int y)	Paints the icon.
10	void setDescription(String description)	Sets the description of the image.
11	void setImage(Image image)	Sets the image displayed by this icon.
12	void setImageObserver(ImageObserver observer)	Sets the image observer for the image.
13	String toString()	Returns a string representation of this image.

Methods Inherited

This class inherits methods from the following classes –

- java.lang.Object

SWING - JFileChooser Class

Introduction

The class JFileChooser is a component which provides a simple mechanism for the user to choose a file.

Class Declaration

Following is the declaration for javax.swing.JFileChooser class –

```
public class JFileChooser extends JComponent implements Accessible
```

Field

Following are the fields for javax.swing.JFileChooser class –

- **static String ACCEPT_ALL_FILE_FILTER_USED_CHANGED_PROPERTY** – Identifies whether the AcceptAllFileFilter is used or not.
- **protected AccessibleContext accessibleContext**
- **static String ACCESSORY_CHANGED_PROPERTY** – Says that a different accessory component is in use (for example, to preview files).
- **static String APPROVE_BUTTON_MNEMONIC_CHANGED_PROPERTY** – Identifies change in the mnemonic for the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TEXT_CHANGED_PROPERTY** – Identifies change in the text on the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TOOL_TIP_TEXT_CHANGED_PROPERTY** – Identifies change in the tooltip text for the approve (yes, ok) button.
- **static int APPROVE_OPTION** – Return value if approve (yes, ok) is chosen.
- **static String APPROVE_SELECTION** – Instruction to approve the current selection (same as pressing yes or ok).
- **static int CANCEL_OPTION** – Return value if cancel is chosen.
- **static String CANCEL_SELECTION** – Instruction to cancel the current selection.
- **static String CHOOSABLE_FILE_FILTER_CHANGED_PROPERTY** – Identifies a change in the list of predefined file filters the user can choose from.
- **static String CONTROL_BUTTONS_ARE_SHOWN_CHANGED_PROPERTY** – Instruction to display the control buttons.
- **static int CUSTOM_DIALOG** – Type value indicating the JFileChooser supports a developer-specified file operation.
- **static String DIALOG_TITLE_CHANGED_PROPERTY** – Identifies a change in the dialog title.
- **static String DIALOG_TYPE_CHANGED_PROPERTY** – Identifies a change in the type of files displayed (files only, directories only, or both files and directories).
- **static int DIRECTORIES_ONLY** – Instruction to display only directories.
- **static String DIRECTORY_CHANGED_PROPERTY** – Identifies the user's directory change.

- **static int ERROR_OPTION** – Return value if an error occurred.
- **static String FILE_FILTER_CHANGED_PROPERTY** – Identifies the user changed the kind of files to display.
- **static String FILE HIDING_CHANGED_PROPERTY** – Identifies a change in the display-hidden-files property.
- **static String FILE_SELECTION_MODE_CHANGED_PROPERTY** – Identifies a change in the kind of selection (single, multiple, etc.).
- **static String FILE_SYSTEM_VIEW_CHANGED_PROPERTY** – Says that a different object is being used to find available drives on the system.
- **static String FILE_VIEW_CHANGED_PROPERTY** – Says that a different object is being used to retrieve file information.
- **static int FILES_AND_DIRECTORIES** – Instruction to display both files and directories.
- **static int FILES_ONLY** – Instruction to display only files.
- **static String MULTI_SELECTION_ENABLED_CHANGED_PROPERTY** – Enables multiple-file selections.
- **static int OPEN_DIALOG** – Type value indicating that the JFileChooser supports an "Open" file operation.
- **static int SAVE_DIALOG** – Type value indicating that the JFileChooser supports a "Save" file operation.
- **static String SELECTED_FILE_CHANGED_PROPERTY** – Identifies change in the user's single-file selection.
- **static String SELECTED_FILES_CHANGED_PROPERTY** – Identifies change in the user's multiple-file selection.

Class Constructors

Sno	Constructor	Description
1	JFileChooser()	Constructs a JFileChooser pointing to the user's default directory.
2	JFileChooser(File currentDirectory)	Constructs a JFileChooser using the given File as the path.
3	JFileChooser(File currentDirectory, FileSystemView fsv)	Constructs a JFileChooser using the given current directory and FileSystemView.
4	JFileChooser(FileSystemView fsv)	Constructs a JFileChooser using the given FileSystemView.
5	JFileChooser(String currentDirectoryPath)	Constructs a JFileChooser using the given path.
6	JFileChooser(String currentDirectoryPath, FileSystemView fsv)	Constructs a JFileChooser using the given current directory path and FileSystemView.

Class Methods

Here is the list of methods in Swing JFileChooser class.

Methods Inherited

This class inherits methods from the following classes –

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Example program on JFileChooser:

```
import javax.swing.*;  
import java.awt.event.*;  
import java.io.*;  
  
public class FileChooserExample extends JFrame implements ActionListener  
{  
  
    JMenuBar mb;  
    JMenu file;  
    JMenuItem open;  
    JTextArea ta;  
  
    FileChooserExample()  
    {  
  
        open=new JMenuItem("Open File");  
        open.addActionListener(this);  
        file=new JMenu("File");  
        file.add(open);  
        mb=new JMenuBar();  
        mb.setBounds(0,0,800,20);  
        mb.add(file);  
        ta=new JTextArea(800,800);  
        ta.setBounds(0,20,800,800);  
        add(mb);  
        add(ta);  
    }  
}
```

```
}

public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==open)
    {
        JFileChooser fc=new JFileChooser();
        int i=fc.showOpenDialog(this);
        if(i==JFileChooser.APPROVE_OPTION)
        {
            File f=fc.getSelectedFile();
            String filepath=f.getPath();
            try
            {
                BufferedReader br=new BufferedReader(new
FileReader(filepath));
                String s1="",s2="";
                while((s1=br.readLine())!=null)
                {
                    s2+=s1+"\n";
                }
                ta.setText(s2);
                br.close();
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }
        }
    }

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

```
FileChooserExample om=new FileChooserExample();
om.setSize(500,500);
om.setLayout(null);
om.setVisible(true);
om.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```



EVENT DELEGATION MODEL

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to delegate the processing of an event to a separate piece of code. In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

This is a more efficient way to handle events than the design used by the old Java 1.0 approach. Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

In the delegation event model, listener must register with a source in order to receiver an event notification .this provides an important benefits :notifications are sent only to listeners that want to recieve the.This is a more efficient way to handle events than the design used by the java 1.0 approach ,Previously an event was propagated up the containment hierarchy until it was handled by a component .This required components to receive events that they did not process and is wasted valuable time .The delegation event model eliminates this overhead

It is important to consider how users interact with the user interface when designing a graphical user interface (GUI). The GUI may require users to click, resize, or drag and drop components of the interface, and input data using the keyboard. These actions will result to an event and you need to write a code to handle them.

Event handling code deals with events generated by GUI user interaction. The best practices for coding event handlers are outlined in the event delegation model.

The event delegation model comprises three elements:

- Event source
- Event listener
- Adapter

Event source

An event source is a component, such as a GUI component, that generates an event. The event source can be generated by any type of user interaction. You can also combine a number of different types of events into one event object.

For example, if a user clicks and drags an icon, you can sum up the mouse-clicked event and the mouse-moved event into one event object.

In the event delegation model, a class represents each event type. Event objects are all defined in the `java.util.EventObject` subclasses.

A generated event object

- provides the methods to add or remove the source event
- manages the list of registered event listeners
- provides the appropriate class type to the registered event listeners

Event listener

Event listeners are objects that receive notification of an event. Components define the events they fire by registering objects called listeners for those event types. When an event is fired, an event object is passed as an argument to the relevant listener object method. The listener object then handles the event.

To receive notification of an event, the object must be registered with the event source. To subscribe to the event source, you implement the appropriate listener interface.

All listeners are implementations of the EventListener interface or one of its subinterfaces. The Java API provides a predefined listener interface for each set of event types that a source can fire. For example, the MouseListener interface deals with mouse events, and the ActionListener interface deals with action events fired by buttons and other components.

Each listener interface provides at least one method for delivering the event object to the listener, in addition to any methods required for the action. All methods take one parameter and are part of the EventObject class.

Adapter

Adapters are abstract classes that implement listener interfaces using predefined methods. These are provided for convenience.

You can use an adapter to apply one listener's methods without having to implement all other methods. Adapters provide empty implementations for all interfaces' methods, so you only need to override the method you are interested in.

The primary design goals of the new model in the AWT are the following:

- Simple and easy to learn
- Support a clean separation between application and GUI code
- Facilitate the creation of robust event handling code which is less error-prone (strong compile-time checking)
- Flexible enough to enable varied application models for event flow and propagation
- For visual tool builders, enable run-time discovery of both events that a component generates as well as the events it may observe

NOTE:

Event types are encapsulated in a class hierarchy rooted at java.util.EventObject. An event is propagated from a "Source" object to a "Listener" object by invoking a method on the listener and passing in the instance of the event subclass which defines the event type generated. A Listener is an object that implements a specific EventListener interface extended from the generic java.util.EventListener. An EventListener interface defines one or more methods which are to be invoked by the event source in response to each specific event type handled by the interface.

AWT Event Classes

The Event classes represent the event. Java provides us various Event classes.

EventObject class

It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the **source**, that is logically deemed to be the object upon which the Event in question initially occurred. This class is defined in `java.util` package.

Class declaration

Following is the declaration for **java.util.EventObject** class:

```
public class EventObject extends Object implements Serializable
```

Field

Following are the fields for **java.util.EventObject** class:

- **protected Object source** -- The object on which the Event initially occurred.

Class constructors

Sno	Constructor	Description
1	<code>EventObject(Object source)</code>	Constructs a prototypical Event.

Class methods

Sno	Method	Description
1	<code>Object getSource()</code>	The object on which the Event initially occurred.
2	<code>String toString()</code>	Returns a String representation of this EventObject.

Methods inherited

This class inherits methods from the following classes:

- `java.lang.Object`

AWT Event Classes:

Following is the list of commonly used event classes.

Sno	Control	Description
1	<u>AWTEvent</u>	It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.
2	<u>ActionEvent</u>	The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	<u>InputEvent</u>	The InputEvent class is root event class for all component-level input events.
4	<u>KeyEvent</u>	On entering the character the Key event is generated.
5	<u>MouseEvent</u>	This event indicates a mouse action occurred in a component.
6	<u>TextEvent</u>	The object of this class represents the text events.
7	<u>WindowEvent</u>	The object of this class represents the change in state of a window.
8	<u>AdjustmentEvent</u>	The object of this class represents the adjustment event emitted by Adjustable objects.
9	<u>ComponentEvent</u>	The object of this class represents the change in state of a window.
10	<u>ContainerEvent</u>	The object of this class represents the change in state of a window.
11	<u>MouseMotionEvent</u>	The object of this class represents the change in state of a window.
12	<u>PaintEvent</u>	The object of this class represents the change in state of a window.

AWT Event Listeners

The Event listener represent the interfaces responsible to handle events. Java provides us various Event listener classes, Every method of an event listener method has a single argument as an object which is subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListner interface

It is a marker interface which every listener interface has to extend. This class is defined in java.util package.

An event listener interface defines the methods used by a component to dispatch events. Each event type will have at least one corresponding dispatch method in a listener interface.

Class declaration

Following is the declaration for java.util.EventListener interface:

```
public interface EventListener
```

AWT Event Listener Interfaces:

Following is the list of commonly used event listeners.

Sno	Control	Description
1	<u>ActionListener</u>	This interface is used for receiving the action events.
2	<u>ComponentListener</u>	This interface is used for receiving the component events.
3	<u>ItemListener</u>	This interface is used for receiving the item events.
4	<u>KeyListener</u>	This interface is used for receiving the key events.
5	<u>MouseListener</u>	This interface is used for receiving the mouse events.
6	<u>TextListener</u>	This interface is used for receiving the text events.
7	<u>WindowListener</u>	This interface is used for receiving the window events.
8	<u>AdjustmentListener</u>	This interface is used for receiving the adjusmtent events.
9	<u>ContainerListener</u>	This interface is used for receiving the container events.
10	<u>MouseMotionListener</u>	This interface is used for receiving the mouse motion events.
11	<u>FocusListener</u>	This interface is used for receiving the focus events.

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event classes	Listener interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
 - `public void addActionListener(ActionListener a){ }`
- **MenuItem**
 - `public void addActionListener(ActionListener a){ }`
- **TextField**
 - `public void addActionListener(ActionListener a){ }`
 - `public void addTextListener(TextListener a){ }`
- **TextArea**
 - `public void addTextListener(TextListener a){ }`
- **Checkbox**
 - `public void addItemListener(ItemListener a){ }`
- **Choice**
 - `public void addItemListener(ItemListener a){ }`
- **List**
 - `public void addActionListener(ActionListener a){ }`
 - `public void addItemListener(ItemListener a){ }`

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener
{
    TextField tf;
    AEvent()
    {
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);

        //register listener
        b.addActionListener(this);//passing current instance

        //add components and set size, layout and visibility
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
}
```

```

public void actionPerformed(ActionEvent e)

{
    tf.setText("Welcome");

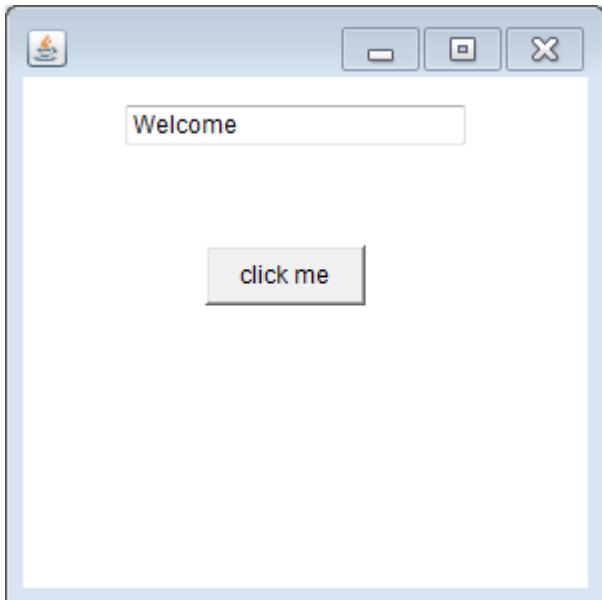
}

public static void main(String args[])
{
    new AEvent();
}

}

```

Output:



public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

2) Java event handling by outer class

```

a) import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame
{
    TextField tf;
    AEvent2()
    {
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
    }
}

```

```

        b.setBounds(100,120,80,30);

        //register listener
        Outer o=new Outer(this);
        b.addActionListener(o);//passing outer class instance
        //add components and set size, layout and visibility
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[])
    {
        new AEvent2();
    }
}

```

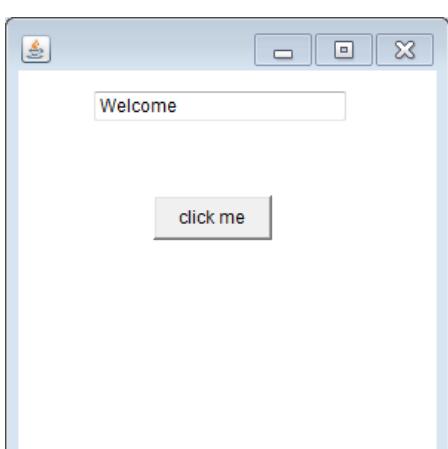
- b) import java.awt.event.*;
 class Outer implements ActionListener

```

{
    AEvent2 obj;
    Outer(AEvent2 obj)
    {
        this.obj=obj;
    }
    public void actionPerformed(ActionEvent e)
    {
        obj.tf.setText("welcome");
    }
}

```

OUTPUT:



Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Java Mouse Listener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in **java.awt.event** package. It has five methods.

Methods of MouseListener interface

The class which processes the MouseEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addMouseListener() method.

Interface declaration

Following is the declaration for **java.awt.event.MouseListener** interface:

```
public interface MouseListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void mouseClicked(MouseEvent e)	Invoked when the mouse button has been clicked (pressed and released) on a component.
2	void mouseEntered(MouseEvent e)	Invoked when the mouse enters a component.
3	void mouseExited(MouseEvent e)	Invoked when the mouse exits a component.
4	void mousePressed(MouseEvent e)	Invoked when a mouse button has been pressed on a component.
5	void mouseReleased(MouseEvent e)	Invoked when a mouse button has been released on a component.

Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

Java MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
    Label l;
    MouseListenerExample()
    {
        addMouseListener(this);
        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e)
    {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e)
    {
        l.setText("Mouse Exited");
    }
}
```

```
}

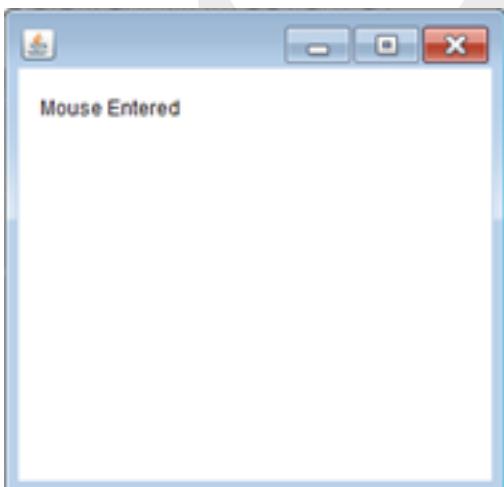
public void mousePressed(MouseEvent e)
{
    l.setText("Mouse Pressed");
}

public void mouseReleased(MouseEvent e)
{
    l.setText("Mouse Released");
}

public static void main(String[] args)
{
    new MouseListenerExample();
}

}
```

Output:



Java MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;

public class MouseListenerExample2 extends Frame implements MouseListener
{
    MouseListenerExample2()
    {
        addMouseListener(this);
    }

    public void mouseEntered(MouseEvent e)
    {
        l.setText("Mouse Entered");
    }

    public void mouseExited(MouseEvent e)
    {
        l.setText("Mouse Exited");
    }

    public void mousePressed(MouseEvent e)
    {
        l.setText("Mouse Pressed");
    }

    public void mouseReleased(MouseEvent e)
    {
        l.setText("Mouse Released");
    }
}
```

```
setSize(300,300);
setLayout(null);
setVisible(true);

}

public void mouseClicked(MouseEvent e)
{
    Graphics g=getGraphics();
    g.setColor(Color.BLUE);
    g.fillOval(e.getX(),e.getY(),30,30);
}

public void mouseEntered(MouseEvent e) {}

public void mouseExited(MouseEvent e) {}

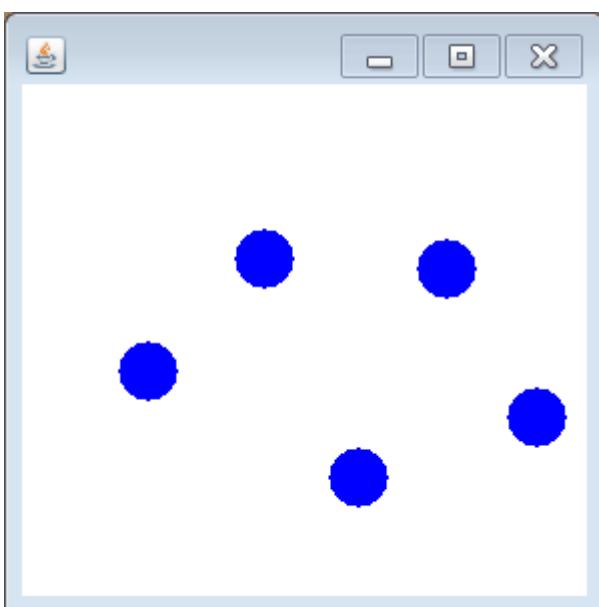
public void mousePressed(MouseEvent e) {}

public void mouseReleased(MouseEvent e) {}

public static void main(String[] args)
{
    new MouseListenerExample2();
}

}
```

Output:



Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

The class which processes the ActionEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addActionListener() method. When the action event occurs, that object's actionPerformed method is invoked.

Interface declaration

Following is the declaration for **java.awt.event.ActionListener** interface:

```
public interface ActionListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void actionPerformed(ActionEvent e)	Invoked when an action occurs.

Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

Java ActionListener Example: On Button click

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample
{
    public static void main(String[] args)
    {
        Frame f=new Frame("ActionListener Example");
        final TextField tf=new TextField();
        tf.setBounds(50,50, 150,20);
        Button b=new Button("Click Here");
        b.setBounds(50,100,60,30);
        b.addActionListener(new ActionListener()
        {

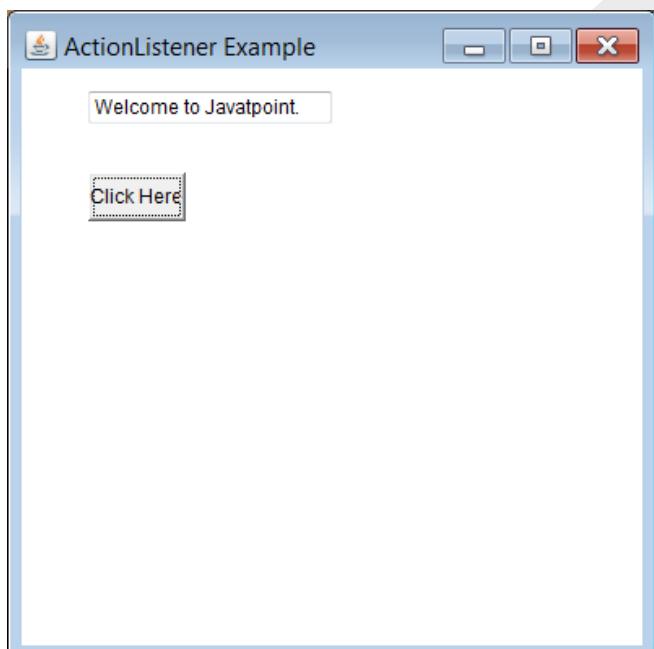
```

```
public void actionPerformed(ActionEvent e)
{
    tf.setText("Welcome to Javatpoint.");
}

});
f.add(b);f.add(tf);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}

}
```

Output:



Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in `java.awt.event` package. It has two methods.

The interface **MouseMotionListener** is used for receiving mouse motion events on a component. The class that process mouse motion events needs to implements this interface.

Class declaration

Following is the declaration for **java.awt.event.MouseMotionListener** interface:

```
public interface MouseMotionListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void mouseDragged(MouseEvent e)	Invoked when a mouse button is pressed on a component and then dragged.
2	void mouseMoved(MouseEvent e)	Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

Methods inherited

This class inherits methods from the following interfaces:

- `java.awt.event.EventListener`

Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;

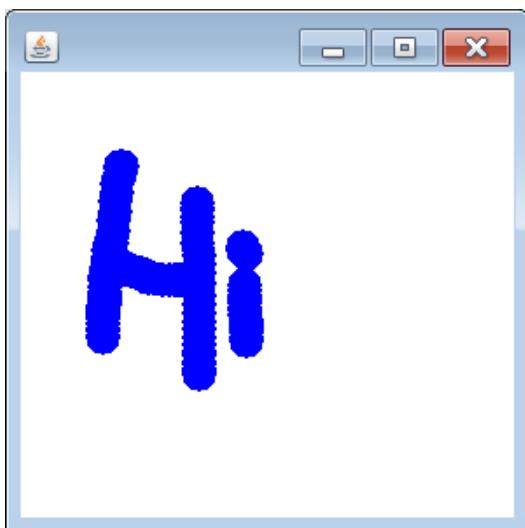
public class MouseMotionListenerExample extends Frame implements MouseMotionListener
{
    MouseMotionListenerExample()
    {
        addMouseMotionListener(this);

        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }

    public void mouseDragged(MouseEvent e)
```

```
{  
    Graphics g=getGraphics();  
    g.setColor(Color.BLUE);  
    g.fillOval(e.getX(),e.getY(),20,20);  
}  
  
public void mouseMoved(MouseEvent e) {}  
  
public static void main(String[] args)  
{  
    new MouseMotionListenerExample();  
}  
}
```

Output:



Java MouseMotionListener Example 2

```
import java.awt.*;  
  
import java.awt.event.MouseEvent;  
  
import java.awt.event.MouseMotionListener;  
  
public class Paint extends Frame implements MouseMotionListener  
{  
  
    Label l;
```

```
Color c=Color.BLUE;

Paint()

{

    l=new Label();

    l.setBounds(20,40,100,20);

    add(l);

    addMouseMotionListener(this);

    setSize(400,400);

    setLayout(null);

    setVisible(true);

}

public void mouseDragged(MouseEvent e)

{

    l.setText("X="+e.getX()+" , Y="+e.getY());

    Graphics g=getGraphics();

    g.setColor(Color.RED);

    g.fillOval(e.getX(),e.getY(),20,20);

}

public void mouseMoved(MouseEvent e)

{

    l.setText("X="+e.getX()+" , Y="+e.getY());

}

public static void main(String[] args)

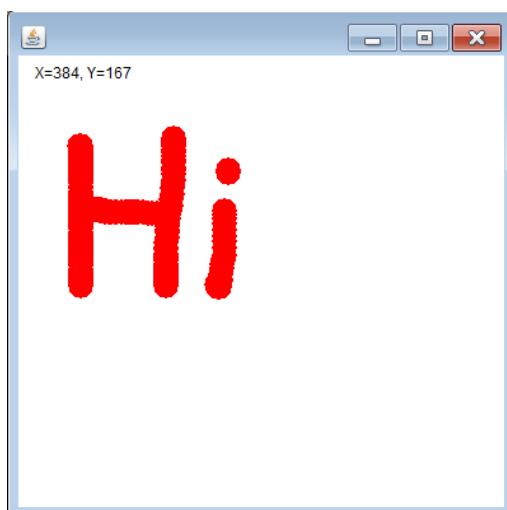
{

    new Paint();

}

}
```

Output:



Java ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

The class which processes the ItemEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addItemListener() method. When the action event occurs, that object's itemStateChanged method is invoked.

Interface declaration

Following is the declaration for **java.awt.event.ItemListener** interface:

```
public interface ItemListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void itemStateChanged(ItemEvent e)	Invoked when an item has been selected or deselected by the user.

Methods inherited

This interface inherits methods from the following interfaces:

1. java.awt.EventListener

Java ItemListener Example

```
import java.awt.*;  
  
import java.awt.event.*;  
  
public class ItemListenerEx implements ItemListener  
{  
  
    Checkbox checkBox1,checkBox2;  
  
    Label label;  
  
    ItemListenerEx()  
    {  
  
        Frame f= new Frame("CheckBox Example");  
  
        label = new Label();  
  
        label.setAlignment(Label.CENTER);
```

```
label.setSize(400,100);

checkBox1 = new Checkbox("C++");

checkBox1.setBounds(100,100, 50,50);

checkBox2 = new Checkbox("Java");

checkBox2.setBounds(100,150, 50,50);

f.add(checkBox1); f.add(checkBox2); f.add(label);

checkBox1.addItemListener(this);

checkBox2.addItemListener(this);

f.setSize(400,400);

f.setLayout(null);

f.setVisible(true);

}

public void itemStateChanged(ItemEvent e)

{

    if(e.getSource()==checkBox1)

        label.setText("C++" + (e.getStateChange()==1?"checked":"unchecked"));

    if(e.getSource()==checkBox2)

        label.setText("Java" + (e.getStateChange()==1?"checked":"unchecked"));

}

public static void main(String args[])

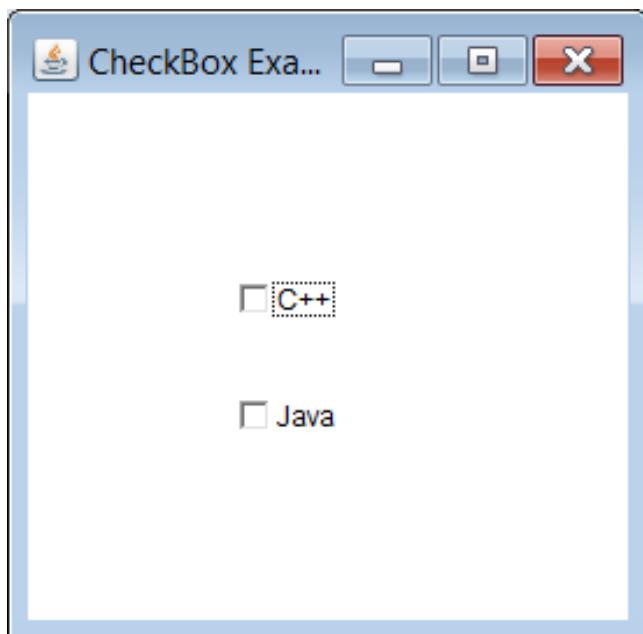
{

    new ItemListenerEx();

}

}
```

Output:



Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addKeyListener() method.

Interface declaration

Following is the declaration for **java.awt.event.KeyListener** interface:

```
public interface KeyListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void keyPressed(KeyEvent e)	Invoked when a key has been pressed.
2	void keyReleased(KeyEvent e)	Invoked when a key has been released.
3	void keyTyped(KeyEvent e)	Invoked when a key has been typed.

Methods inherited

This interface inherits methods from the following interfaces:

- java.awt.EventListener

Java KeyListener Example

```
import java.awt.*;  
  
import java.awt.event.*;  
  
public class KeyListenerExample extends Frame implements KeyListener  
  
{  
  
    Label l;  
  
    TextArea area;  
  
    KeyListenerExample()  
  
    {  
  
        l=new Label();  
  
        l.setBounds(20,50,100,20);  
  
        area=new TextArea();
```

```
area.setBounds(20,80,300, 300);

area.addKeyListener(this);

add(l);

add(area);

setSize(400,400);

setLayout(null);

setVisible(true);

}

public void keyPressed(KeyEvent e)

{

    l.setText("Key Pressed");

}

public void keyReleased(KeyEvent e)

{

    l.setText("Key Released");

}

public void keyTyped(KeyEvent e)

{

    l.setText("Key Typed");

}

public static void main(String[] args)

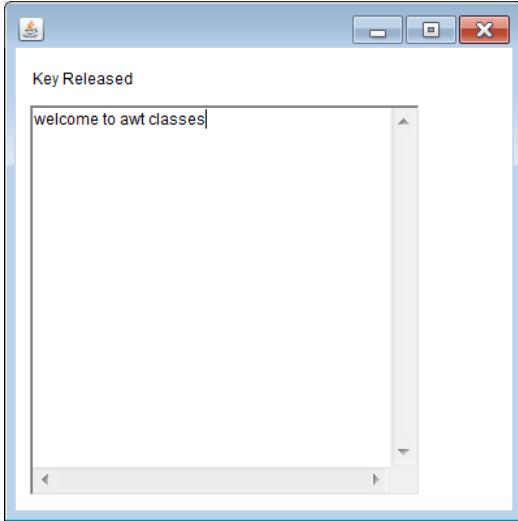
{

    new KeyListenerExample();

}

}
```

Output:



Java KeyListener Example 2: Count Words & Characters

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample2 extends Frame implements KeyListener
{
    Label l;
    TextArea area;
    KeyListenerExample2()
    {
        l=new Label();
        l.setBounds(20,50,200,20);
        area=new TextArea();
        area.setBounds(20,80,300, 300);
        area.addKeyListener(this);
        add(l);
        add(area);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
}
```

```
public void keyPressed(KeyEvent e) {}

public void keyReleased(KeyEvent e)

{

    String text=area.getText();

    String words[]=text.split("\\s");

    l.setText("Words: "+words.length+" Characters:"+text.length());

}

public void keyTyped(KeyEvent e) {}

public static void main(String[] args)

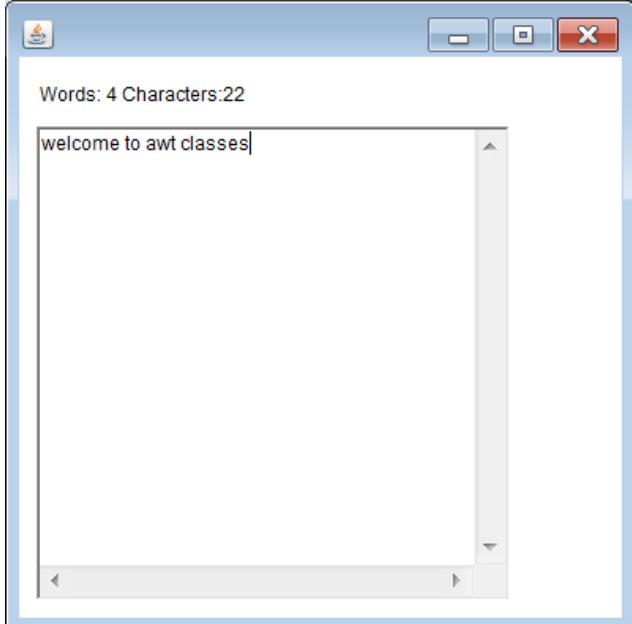
{

    new KeyListenerExample2();

}

}
```

Output:



Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in `java.awt.event` package. It has three methods.

The class which processes the WindowEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the `addWindowListener()` method.

Interface declaration

Following is the declaration for **java.awt.event.WindowListener** interface:

```
public interface WindowListener extends EventListener
```

Interface methods

Sno	Method	Description
1	void windowActivated(WindowEvent e)	Invoked when the Window is set to be the active Window.
2	void windowClosed(WindowEvent e)	Invoked when a window has been closed as the result of calling dispose on the window.
3	void windowClosing(WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
4	void windowDeactivated(WindowEvent e)	Invoked when a Window is no longer the active Window.
5	void windowDeiconified(WindowEvent e)	Invoked when a window is changed from a minimized to a normal state.
6	void windowIconified(WindowEvent e)	Invoked when a window is changed from a normal to a minimized state.
7	void windowOpened(WindowEvent e)	Invoked the first time a window is made visible.

Methods inherited

This interface inherits methods from the following interfaces:

- `java.awt.EventListener`

Java WindowListener Example

```
import java.awt.*;  
  
import java.awt.event.WindowEvent;  
  
import java.awt.event.WindowListener;  
  
public class WindowExample extends Frame implements WindowListener  
{  
  
    WindowExample()  
    {  
  
        addWindowListener(this);  
  
        setSize(400,400);  
  
        setLayout(null);  
    }  
}
```

```
        setVisible(true);

    }

public static void main(String[] args)
{
    new WindowExample();
}

public void windowActivated(WindowEvent arg0)
{
    System.out.println("activated");
}

public void windowClosed(WindowEvent arg0)
{
    System.out.println("closed");
}

public void windowClosing(WindowEvent arg0)
{
    System.out.println("closing");
    dispose();
}

public void windowDeactivated(WindowEvent arg0)
{
    System.out.println("deactivated");
}

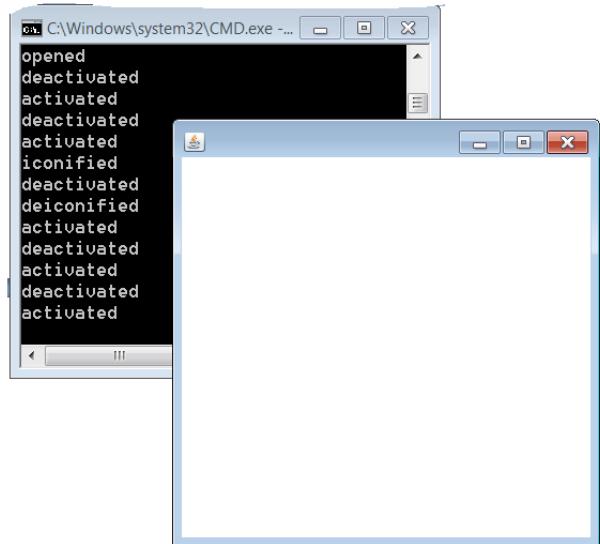
public void windowDeiconified(WindowEvent arg0)
{
    System.out.println("deiconified");
}

public void windowIconified(WindowEvent arg0)
{
    System.out.println("iconified");
}
```

```
public void windowOpened(WindowEvent arg0)
{
    System.out.println("opened");
}

}
```

Output:



How to close AWT Window in Java

We can close the AWT Window or Frame by calling `dispose()` or `System.exit()` inside `windowClosing()` method. The `windowClosing()` method is found in **WindowListener** interface and **WindowAdapter** class.

The `WindowAdapter` class implements `WindowListener` interfaces. It provides the default implementation of all the 7 methods of `WindowListener` interface. To override the `windowClosing()` method, you can either use `WindowAdapter` class or `WindowListener` interface.

If you implement the `WindowListener` interface, you will be forced to override all the 7 methods of `WindowListener` interface. So it is better to use `WindowAdapter` class.

Different ways to override `windowClosing()` method

There are many ways to override `windowClosing()` method:

- By anonymous class
- By inheriting `WindowAdapter` class
- By implementing `WindowListener` interface

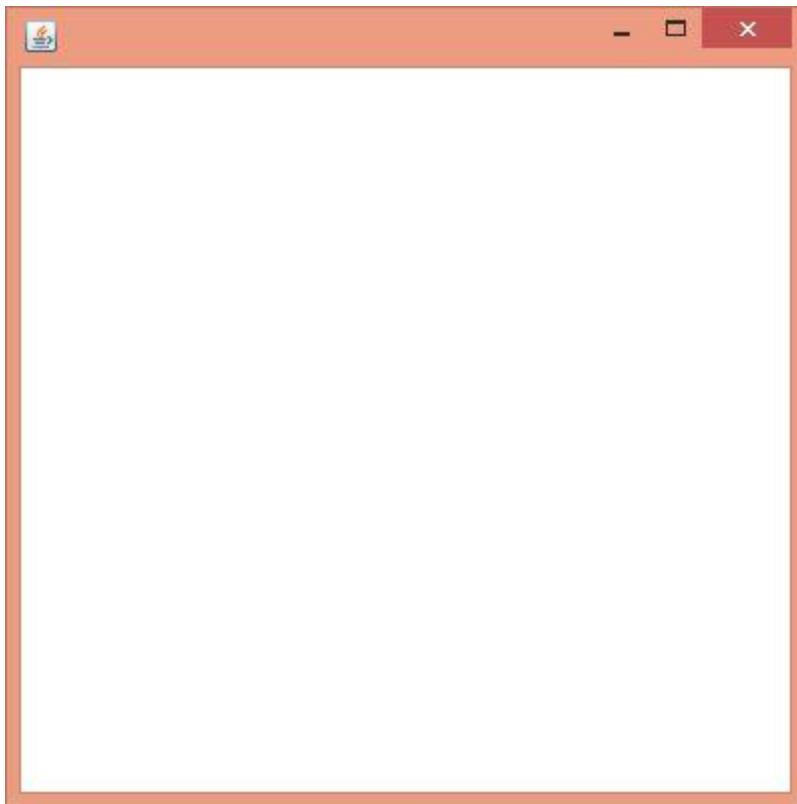
Close AWT Window Example 1: Anonymous class

```
import java.awt.*;  
  
import java.awt.event.WindowEvent;  
  
import java.awt.event.WindowListener;  
  
public class WindowExample extends Frame  
{  
  
    WindowExample()  
    {  
  
        addWindowListener(new WindowAdapter()  
        {  
  
            public void windowClosing(WindowEvent e)  
            {  
  
                dispose();  
            }  
  
        });  
  
        setSize(400,400);  
  
        setLayout(null);  
  
        setVisible(true);  
    }  
}
```

```
public static void main(String[] args)
{
    new WindowExample();
}

}
```

Output:



Close AWT Window Example 2: extending WindowAdapter

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample extends WindowAdapter
{
    Frame f;
    AdapterExample()
    {
        f=new Frame();
        f.addWindowListener(this);

        f.setSize(400,400);
        f.setLayout(null);
```

```
f.setVisible(true);

}

public void windowClosing(WindowEvent e)

{

    f.dispose();

}

public static void main(String[] args)

{

    new AdapterExample();

}

}
```

Close AWT Window Example 3: implementing WindowListener

```
import java.awt.*;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

public class WindowExample extends Frame implements WindowListener

{

    WindowExample()

    {

        addWindowListener(this);

        setSize(400,400);

        setLayout(null);

        setVisible(true);

    }

    public static void main(String[] args)

    {

        new WindowExample();

    }

    public void windowActivated(WindowEvent e)

    {

    }

}
```

```
public void windowClosed(WindowEvent e)
{
}

public void windowClosing(WindowEvent e)
{
    dispose();
}

public void windowDeactivated(WindowEvent e)
{
}

public void windowDeiconified(WindowEvent e)
{
}

public void windowIconified(WindowEvent e)
{
}

public void windowOpened(WindowEvent arg0)
{
}

}
```

Java AWT Panel

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

It doesn't have title bar.

AWT Panel class declaration

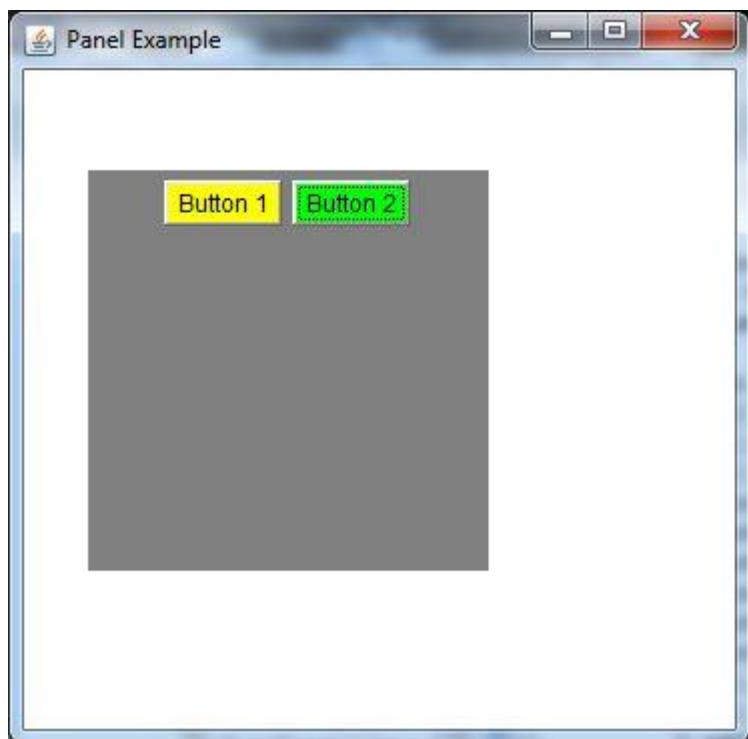
```
public class Panel extends Container implements Accessible
```

Java AWT Panel Example

```
import java.awt.*;  
  
public class PanelExample  
{  
  
    PanelExample()  
    {  
  
        Frame f= new Frame("Panel Example");  
  
        Panel panel=new Panel();  
  
        panel.setBounds(40,80,200,200);  
  
        panel.setBackground(Color.gray);  
  
        Button b1=new Button("Button 1");  
  
        b1.setBounds(50,100,80,30);  
  
        b1.setBackground(Color.yellow);  
  
        Button b2=new Button("Button 2");  
  
        b2.setBounds(100,100,80,30);  
  
        b2.setBackground(Color.green);  
  
        panel.add(b1); panel.add(b2);  
  
        f.add(panel);  
  
        f.setSize(400,400);  
  
        f.setLayout(null);  
  
        f.setVisible(true);  
    }  
  
    public static void main(String args[])  
    {  
  
        new PanelExample();  
    }  
}
```

```
    }  
}
```

Output:



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

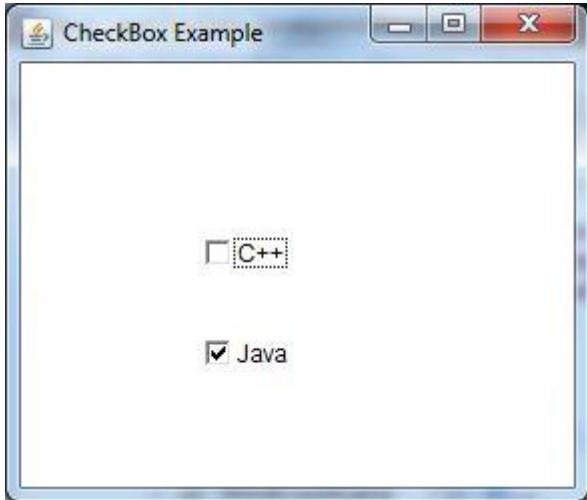
AWT Checkbox Class Declaration

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

Java AWT Checkbox Example

```
import java.awt.*;  
  
public class CheckboxExample  
{  
    CheckboxExample()  
    {  
        Frame f= new Frame("Checkbox Example");  
        Checkbox checkbox1 = new Checkbox("C++");  
        checkbox1.setBounds(100,100, 50,50);  
        Checkbox checkbox2 = new Checkbox("Java", true);  
        checkbox2.setBounds(100,150, 50,50);  
        f.add(checkbox1);  
        f.add(checkbox2);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
  
    public static void main(String args[])  
    {  
        new CheckboxExample();  
    }  
}
```

Output:



Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

AWT CheckboxGroup Class Declaration

```
public class CheckboxGroup extends Object implements Serializable
```

Java AWT CheckboxGroup Example

```
import java.awt.*;  
  
public class CheckboxGroupExample  
{  
    CheckboxGroupExample()  
    {  
        Frame f= new Frame("CheckboxGroup Example");  
        CheckboxGroup cbg = new CheckboxGroup();  
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);  
        checkBox1.setBounds(100,100, 50,50);  
        Checkbox checkBox2 = new Checkbox("Java", cbg, true);  
        checkBox2.setBounds(100,150, 50,50);  
        f.add(checkBox1);  
    }  
}
```

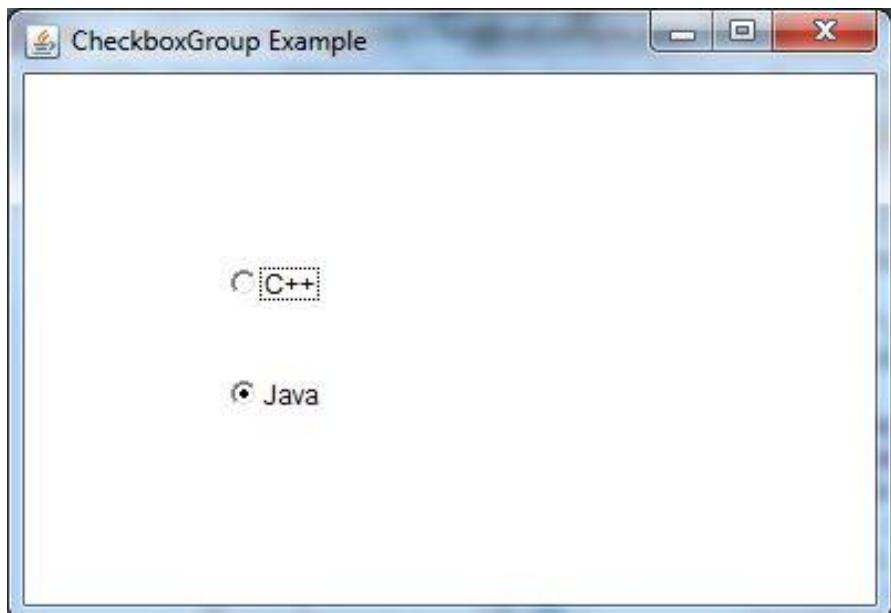
```
f.add(checkBox2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);

}

public static void main(String args[])
{
    new CheckboxGroupExample();
}

}
```

Output:



Java AWT PopupMenu

PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.

AWT PopupMenu class declaration

```
public class PopupMenu extends Menu implements MenuContainer, Accessible
```

Java AWT PopupMenu Example

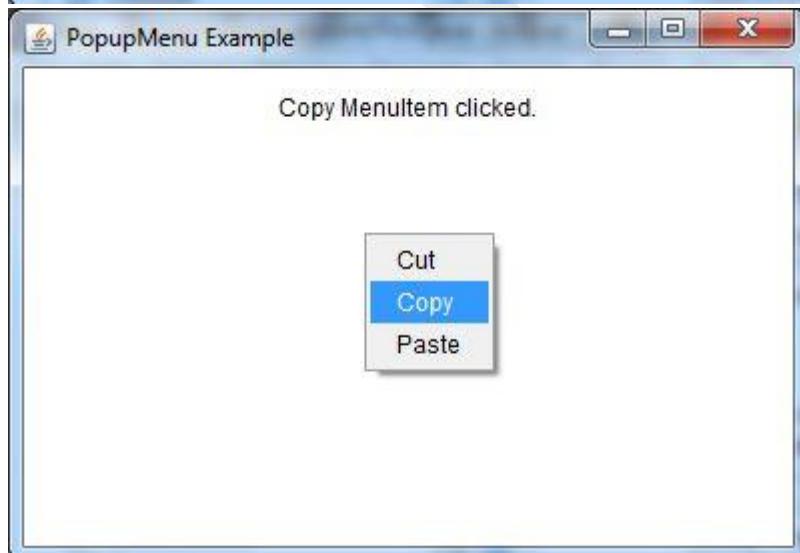
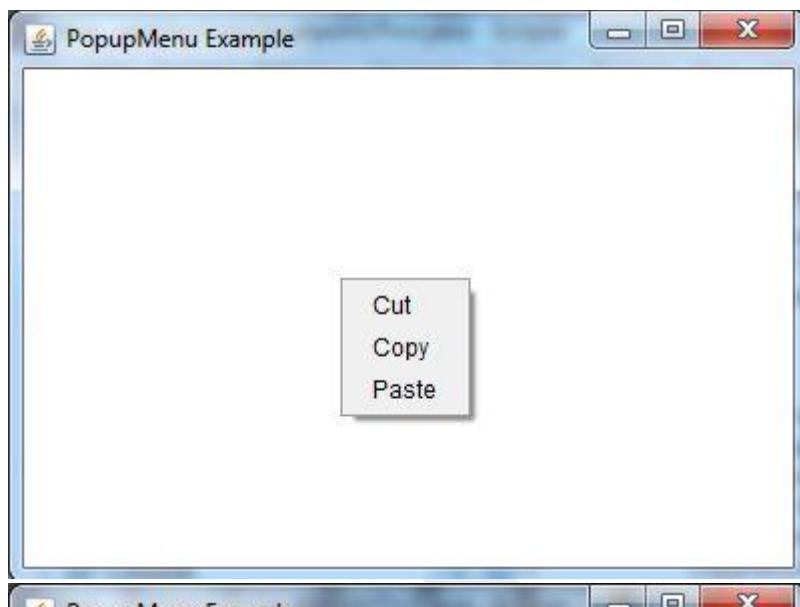
```
import java.awt.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample()
    {
        final Frame f= new Frame("PopupMenu Example");
        final PopupMenu popupmenu = new PopupMenu("Edit");
        MenuItem cut = new MenuItem("Cut");
        cut.setActionCommand("Cut");
        MenuItem copy = new MenuItem("Copy");
        copy.setActionCommand("Copy");
        MenuItem paste = new MenuItem("Paste");
        paste.setActionCommand("Paste");
        popupmenu.add(cut);
        popupmenu.add(copy);
        popupmenu.add(paste);
        f.addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent e)
            {
                popupmenu.show(f , e.getX(), e.getY());
            }
        });
        f.add(popupmenu);
```

```
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}

public static void main(String args[])
{
    new PopupMenuExample();
}

}
```

Output:



LayoutManagers:

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

BorderLayout:

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

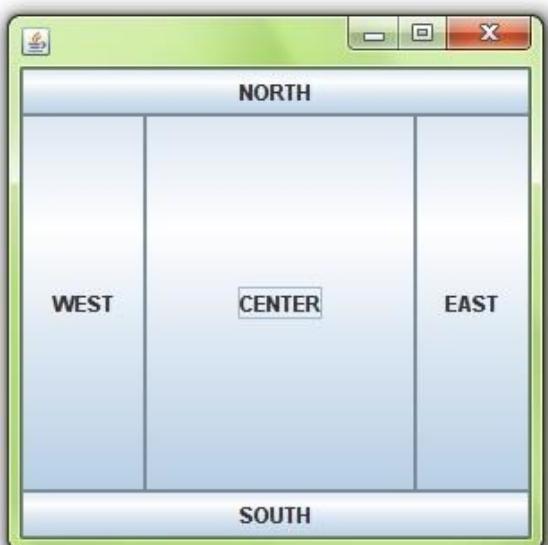
Constructors of BorderLayout class:

- o **BorderLayout():** creates a border layout but with no gaps between the components.
- o **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:

```
import java.awt.*;  
  
import javax.swing.*;  
  
public class Border  
{  
  
    JFrame f;  
  
    Border()  
    {
```

```
f=new JFrame();  
  
JButton b1=new JButton("NORTH");;  
  
JButton b2=new JButton("SOUTH");;  
  
JButton b3=new JButton("EAST");;  
  
JButton b4=new JButton("WEST");;  
  
JButton b5=new JButton("CENTER");;  
  
f.add(b1,BorderLayout.NORTH);  
  
f.add(b2,BorderLayout.SOUTH);  
  
f.add(b3,BorderLayout.EAST);  
  
f.add(b4,BorderLayout.WEST);  
  
f.add(b5,BorderLayout.CENTER);  
  
f.setSize(300,300);  
  
f.setVisible(true);  
  
}  
  
public static void main(String[] args)  
{  
    new Border();  
}  
}
```



Applet

Applet is a special type of program that is embedded in a web page.

An applet is a small application which is designed to be transmitted over the network dynamically executed as part of web documentation on a java compatible web browser.

An applet does not have main function but the applet will be executed when the name of the applet is passed to applet viewer or an html file which is executed from a browser.

Hence applet can be executed in two ways

1. Using web browser with the help of html file.
2. Using a tool known as applet viewer.

The compilation of the applet is similar to that of normal java program

HTML:

HTML stands for Hyper Text Markup Language & it is used to create web pages.

- HTML describes the structure of the web pages using markup
- HTML elements are building blocks of HTML pages. These elements are represented by “tags” which consists of a labeled pieces a content such as headings, paragraphs, tables..etc.

Simple HTML structure:

```
Start <HTML>
End </HTML>
<!DOCTYPE HTML>
<HTML>
  <head>
    <title>first page</title>
  </head>
  <body>
    <h1>first heading</h1>
    <p>hello welcome to html</p>
  </body>
</HTML>
```

<!DOCTYPE>	Defines the doc to be a file
<HTML>	Html is the root element of html page
<head>	This element contains information of this doc
<title>	Specifies title for this document
<body>	Elements contains the visible page content
<h1>	The h1 element designs the large heading
<p>	P element designs paragraph

HTML tags

The html tags are the names surrounded by angular brackets

Syntax: <tag>...content...</tag>

- The html tags are normally come in page like (<html></html> (<body></body>) (<p></p>))
- The first tag in the tag is start tag. The second tag is end tag

Note: The first version of the Html was released in 1991 and in 2014 html5 was released

HTML editor:

Normally we can use professional Html notations to write Html programs but commonly used editor is notepad,eclipse editor,CKE editor.

Write and executing a HTML program

Step1: open a notepad.

Step2: write html source code

Step3: save the html page with following syntax

“html_filename.html”

Step4: view the html page in a browser

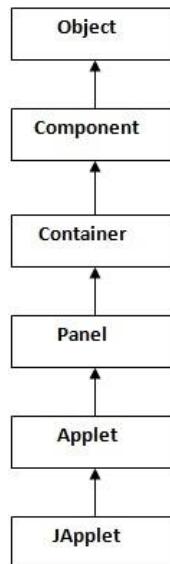
HTML basic tags:

1. For headings are of 3 types they are
<h1>,<h2>,<h3>
Ex: <h1> first page </h1>(variable in size of heading)
2. For paragraph: <P> and </P>
<P> hello welcome to html </P>
3. Html link:
<a href="<http://www.gmail.com>">
4. For image

Advantages:

1. It works at client side.
2. There is no risk of viewer because of an applet has limited
3. It can be executed by browsers running under any platform like linux, windows etc.

Applet hierarchy:



Difference between applet and application

Applet	Application
Applet internally implements GUI is application	In application GUI is option
JVM needs applet life cycle methods for execution	JVM is main method is as for execution
It requires more memory for execution	It requires less memory for execution
More secure	Less secure when compare to applet
Environmental inputs are passed through parm. Tag	Environmental inputs are passed through like arguments

Applet life cycle methods

Inorder to execute an applet program the life cycle methods of an applet are used.

The life cycle methods of an applet are the methods that executes from the applet creation till destruction of the applet.

For our an application to become an applet we must extend the applet class present in `java.applet.Applet` package.

We must override metods in applet in order to create our own applet. There are totally 5 methods of applet.

The following are the methods of an applet:

1. Public void init()
2. Public void start()
3. Public void paint(Graphics g)
4. Public void stop()
5. Public void destroy()

Note:

Lifecycle of Java Applet

Applet is initialized.

Applet is started.

Applet is painted.

Applet is stopped.

Applet is destroyed.

Lifecycle methods for Applet:

The `java.applet.Applet` class has 4 life cycle methods and `java.awt.Component` class provides 1 life cycle method for an applet.

`java.applet.Applet` class

For creating any applet `java.applet.Applet` class must be inherited. It provides 4 life cycle methods of applet.

public void init(): is used to initialize the Applet. It is invoked only once.

public void start(): is invoked after the `init()` method or browser is maximized. It is used to start the Applet.

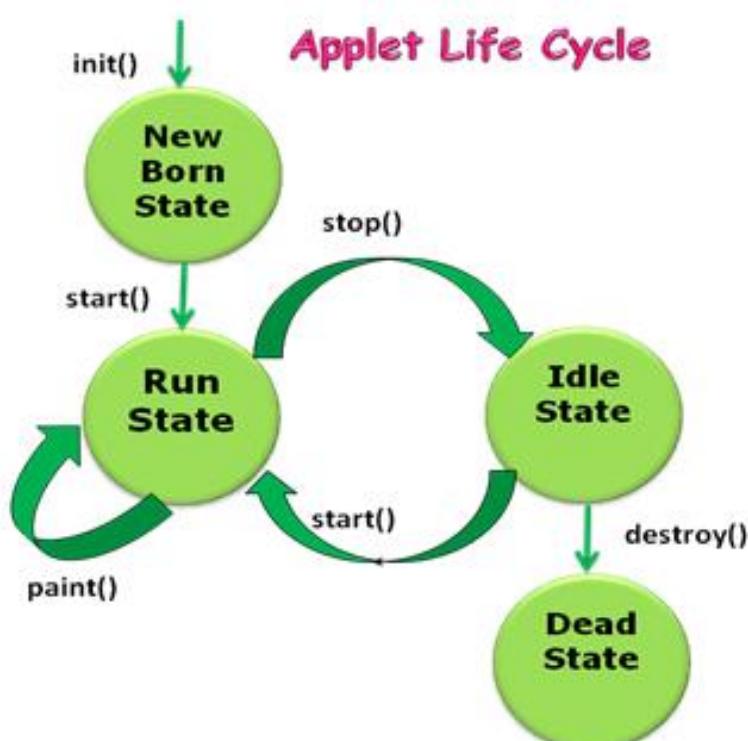
public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

public void destroy(): is used to destroy the Applet. It is invoked only once.

`java.awt.Component` class

The Component class provides 1 life cycle method of applet.

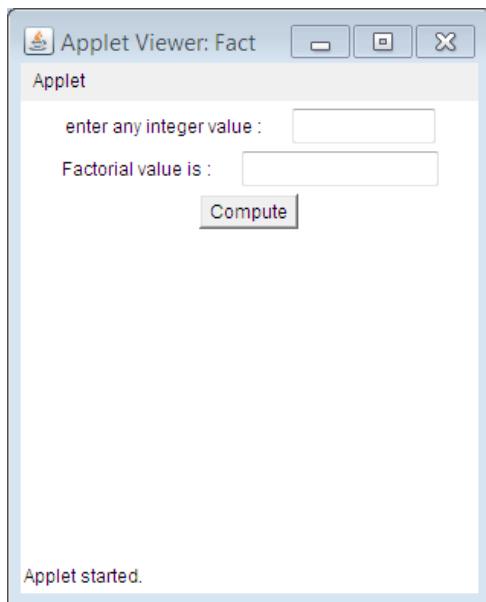
public void paint(Graphics g): is used to paint the Applet. It provides `Graphics` class object that can be used for drawing oval, rectangle, arc etc.



EXAMPLE:

```
import java.awt.*;
import java.lang.String;
import java.awt.event.*;
import java.applet.Applet;
/*<applet code="Fact" height="500" width="500"></applet>*/
public class Fact extends Applet implements ActionListener
{
    String str;
    Button b;
    TextField t1,t2;
    Label l1;
    public void init()
    {
        Panel p=new Panel();
        p.setLayout(new FlowLayout());
        add(new Label("enter any integer value : "));
        add(t1=new TextField(10));
        add(new Label("Factorial value is : "));
        add(t2=new TextField(15));
        add(b=new Button("Compute"));
        b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        int i,n,f=1;
        n=Integer.parseInt(t1.getText());
        for(i=1;i<=n;i++)
            f=f*i;
        t2.setText(String.valueOf(f));
        repaint();
    }
}
```

Output:



EXAMPLE:

```
import java.awt.*;
import java.lang.String;
import java.awt.event.*;
import java.applet.Applet;
/*<applet code="Add1" width="500" height="500"></applet>*/

public class Add1 extends Applet implements ActionListener
{
    String msg;
    TextField num1,num2,res;
    Label l1,l2,l3;
    Button div;
    public void init()
    {
        this.setVisible(true);
        this.setSize(500,500);
        this.setLayout(new FlowLayout());
        add(l1=new Label("Number 1 :"));
    }
}
```

```
add(num1=new TextField(10));
add(l2=new Label("Number 2 : "));
add(num2=new TextField(10));
add(l3=new Label("Result : "));
add(res=new TextField(10));
add(div=new Button("DIV"));
div.addActionListener(this);
//add(msg=new Label());
}

public void actionPerformed(ActionEvent e)
{
    String label=e.getActionCommand();
    String s1=num1.getText();
    String s2=num2.getText();
    int num1=Integer.parseInt(s1);
    int num2=Integer.parseInt(s2);

    if(label.equals("DIV"))
    {
        if(num2==0)
        {
            msg="ArithmaticException";
            repaint();
        }
        else if((num1<0) || (num2<0))
        {
            msg="NumberFormatException";
            repaint();
        }
        else
        {

```

```
        int num3=num1/num2;  
  
        msg=String.valueOf(num3);  
  
    }  
  
    res.setText(msg);  
  
}  
  
}  
  
public void paint(Graphics g)  
{  
  
    g.drawString(msg,30,70);  
  
}  
  
}
```

OUTPUT:

