Unit -1

Introduction to Java: Introduction to Object Oriented Paradigm, Concepts of OOP, Applications of OOP, History of Java, Java Features, JVM, Program Structure. Variables, Primitive Data Types, Constants, Java String Class, Expressions, Primitive type conversion and Casting, Control Structures.

Object Oriented Paradigm :

<u>Introduction to Object Oriented Programming:</u>

Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. Objects collaborate by sending messages to each other. An object is an entity that possess both state (called properties/attributes, in program defined as variables) and behaviour (the functionalities, in program defined as functions)

Principles of Object-Oriented Programming

- Class
- Object
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Class:

A class is a logical entity. It is blueprint from which individual objects are created. It represents the set of properties or methods that are common to all objects of one type.

Object:

Objects are the key for Object Oriented Programming. An Object is called the instance for a class (instantiation /initialization / memory allocation for a class) An Object possess two characteristics

State (Properties)

Behaviour (functions)

An Object stores its state in fields (Variables) and exposes its behaviour through methods.

Methods operate for functionalities which uses the state of object internally and serves as primary mechanism for object to object communication.

Example for Class and Object

Let’s take familiar entities Student and Teacher

First identify what are the properties and behaviour of these entities Student entity

State / properties include

Roll Number (String)

Name (String)

Year

Age (integer)

Marks (integer)

Branch (String)

Result (String) Behaviour / methods include

listenClasses()

onlineExam()

playGames()

let’s take a function writeExam

```
onlineExam()
{
// Use the properties, year and branch to retrieve respective online bits from data base
// allow the student to select their answers
// generate the result and update student’s property marks.
}
```

From the above function we can clearly observe that methods perform some actions using any properties if needed (like year and branch) and updates and property if needed (like marks).

Abstraction:

Abstraction is the process of hiding complexity (internal implementation) and showing essential information. For example, if you want to drive a car, you don’t need to know about its internal workings. The same is true of Java classes. You can hide internal implementation details by using abstract classes or interfaces. On the abstract level, you only need to define the method signatures (name and parameter list) and let other classes implement these interfaces in their own way.

Encapsulation:

Binding the state and behaviour together into a single unit is known as encapsulation. In encapsulation, the variables / data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

For example, if we write a single C program for defining state and behaviour of different entities (say student and teacher) it will be like

```
main ()
{
// following represent entity student properties
char [30] Rollnum;
char [30] name; int marks;
// following represent entity teacher properties char [30] name;
int empId;
char [20] PFnum;

}
// functions for entity student include listenClasses()
{

…. // implementation
}

writeExams()
{

… // implementation
}


// functionalities for entity teacher, include examEvaluation()
{
```

```
….. // implementation
}



postAttendence()
{

…. // implementation
}
```

In this approach maintaining, all entities state and behaviour together, programmer may willingly or unwillingly can access one entity variables under another entity functionalities.

With the help of classes we can eliminate such scenarios and bind properties and its related functions into single unit class also known as Encapsulation

```
class Student
{

// following represent entity student properties char [30] Rollnum;
char [30] name; int marks;
// functions for entity student include listenClasses()
{

…. // implementation
}

writeExams()
{

… // implementation
}

} // end of class Student.



Class Teacher
{
// following represent entity teacher properties char [30] name;
int empId;
char [20] PFnum;


}
// functions for entity student include listenClasses()
{

…. // implementation
}
writeExams()
{

… // implementation
}



// functionalities for entity teacher, include examEvaluation()
{

….. // implementation
}



postAttendence()
{

…. // implementation
}

} //end of class Teacher.
```

Applications of Object Oriented Programming User interface design such as windows, menu. Real Time Systems

Simulation and Modeling Object oriented databases AI and Expert System

Neural Networks and parallel programming

Decision support and office automation systems etc.

History of Java:

Java is invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems. in 1991. Most of the Java characteristics are inherited from C and C++ language. It was first named as Greentalk later called as "Oak" and was finally named as "Java" in 1995.

Other languages have the problem that they are designed to compile the code for a specific platform. To overcome this, Gosling and others started working on a portable and platform-independent language, this leads to the creation of Java.

Java had an extreme effect on the Internet by the innovation of a new type of networked program called the Applet. An applet is a Java program that is designed to be transmitted over the internet and executed by the web browser that is Java-compatible. Applets are the small program that is used to display data provided by the server, handle user input, provide a simple functions.

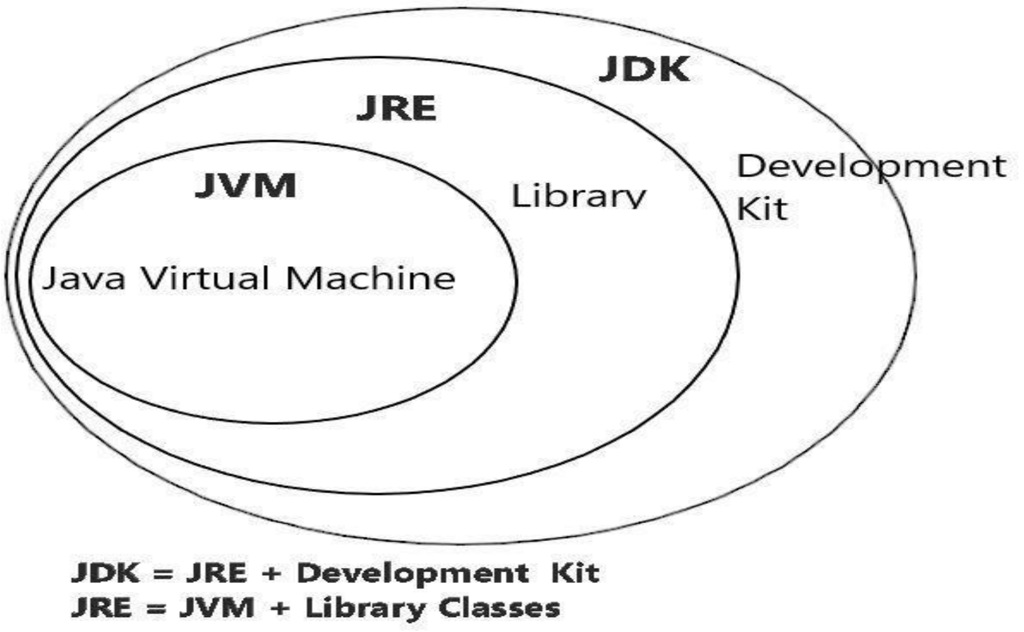
JDK :

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps to code and run Java programs.

JRE :

JRE is a software which is designed to provide runtime environment for java applications.

It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs. As all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.



Evaluation of Java Versions

Java SE Version	Version Number	Release Date
JDK 1.0	1.0	January 1996
JDK 1.1	1.1	February 1997
J2SE 1.2	1.2	December 1998
J2SE 1.3	1.3	May 2000
J2SE 1.4	1.4	February 2002
J2SE 5.0	1.5	September 2004
Java SE 6	1.6	December 2006
Java SE 7	1.7	July 2011
Java SE 8	1.8	March 2014
Java SE 9	9	September, 21st 2017

Java SE 10	10	March, 20th 2018
Java SE 11	11	September, 25th 2018
Java SE 12	12	March, 19th 2019
Java SE 13	13	September, 17th 2019
Java SE 14	14	March, 17th 2020
Java SE 15	15	September, 15th 2020
Java SE 16	16	March, 16th 2021
Java SE 17	*17*	*Expected on Sept. 2021*

Among these versions only Java 8 and Java 11 have LTS (Long Term Service). Java 8 is the default and recommended version to download

what is LTS ?

A Java LTS (long-term support) release is a version of Java that will remain the industry standard for several years. Java 8 which was released in 2014, will continue to receive updates until 2020, and extended support will end by 2025. This gives plenty of OS vendors like Microsoft and Red Hat the time to repackage their releases with Java 8, time for application developers to update their applications to take full advantage of Java 8 features. At this time, the only other Java version that have LTS service is Java 11

<u>Java Features</u>

1) **Simple:**

The Java programming language is easy to learn. Java is similar to C/C++ but it removes the drawbacks and complexities of C/C++ like pointers and multiple inheritances. So one having knowledge on these languages will find Java familiar and easy to learn.

Object-Oriented programming language:

Java is a object-oriented programming language. It has all OOP features such as

- Object
- Class
- Inheritance

- Polymorphism
- Abstraction
- Encapsulation

Robust:

Java uses strong memory management techniques so that there is no improper memory assignment during the running of a program. The unreferenced objects still being in the memory led to the wastage of space. Java's garbage collector solves the problem it will delete the objects which are not used or not referenced anymore by the program.

Secure:

The Java platform is designed with security features built into the language, You never hear about viruses attacking Java applications. Memory access via pointer and performing pointer arithmetic is unsafe, so Java has no support for pointers to provide more security.

High Performance:

Java is an interpreted language, so it cannot be as fast as a compiled language like C or C++. But, Java achieves high performance with the use of just-in-time compiler.

Java is Multithreaded:

With this feature, Java supports "Multitasking". Multitasking is when multiple jobs are executed simultaneously. Multitasking improves CPU and Main Memory Utilization.

Distributed

In the era of Internet, applications need to run in distributed environment. This is possible in java applications since the programmer can use the TCP/IP protocols in the code. Java offers Remote Method Invocation (RMI) package to implement such interfaces in a multi-user application.

Java is Platform Independence:

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java follows write-once, run-anywhere principle.

On compilation Java program is compiled into bytecode.

This bytecode is platform independent and can be run on any machine.

JVM (Java Virtual Machine)

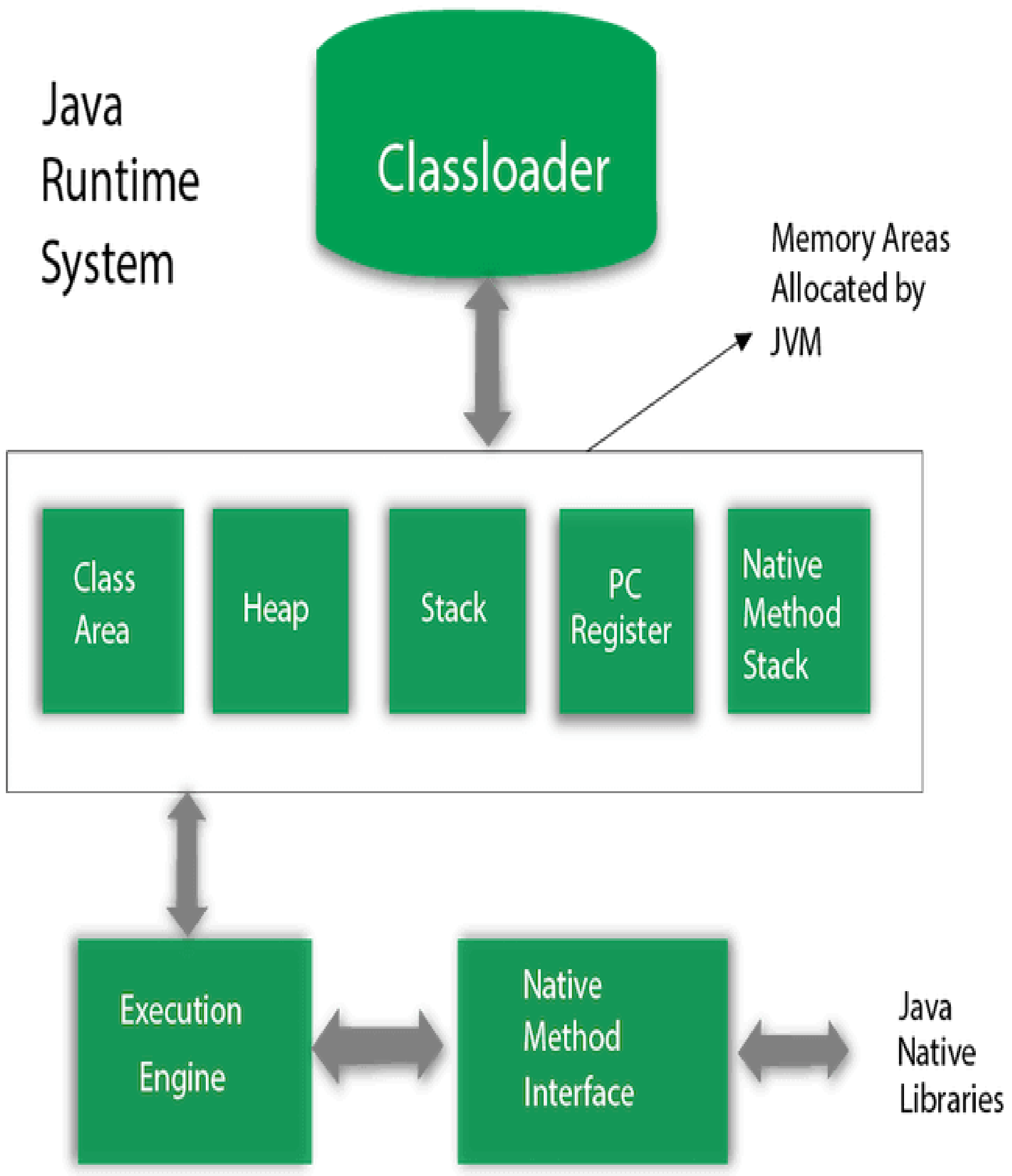
JVM (Java Virtual Machine) is a specification that provides runtime environment in which java bytecode can be executed. JVMs are platform dependent. The JVM will be provided separately for different machine languages(OS)

functionalities performed by the JVM

1. Loads code
2. Verifies code
3. Executes code
4. Provides runtime environment

JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.

System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. It is also known as Applicationclassloader.

Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

Heap

It is the runtime data area in which objects are allocated.

Stack

Java Stack stores frames. It holds local variables and partial results.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

Native Method Stack

It contains all the native methods used in the application.

Execution Engine

The execution engine is the Central Component of the java virtual machine(JVM). It communicates with various memory areas of the JVM. Each thread of a running application is a distinct instance of the virtual machine's execution engine. Execution engine executes the byte code which is assigned to the run time data areas in JVM via class loader. Java Class files are executed by the execution engine.

ExecutionEnginecontainsthreemaincomponentsforexecutingJavaClasses.Theyare:

Interpreter: It reads the byte code and interprets(convert) into the machine code(native code) and executes them in a sequential manner.

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Profiler: This is a tool which is the part of JIT Compiler is responsible to monitor the java bytecode constructs and operations at the JVM level.

Garbage Collector: This is a program in java that manages the memory automatically. It is a daemon thread which always runs in the background. This basically frees up the heap memory by destroying unreachable methods.

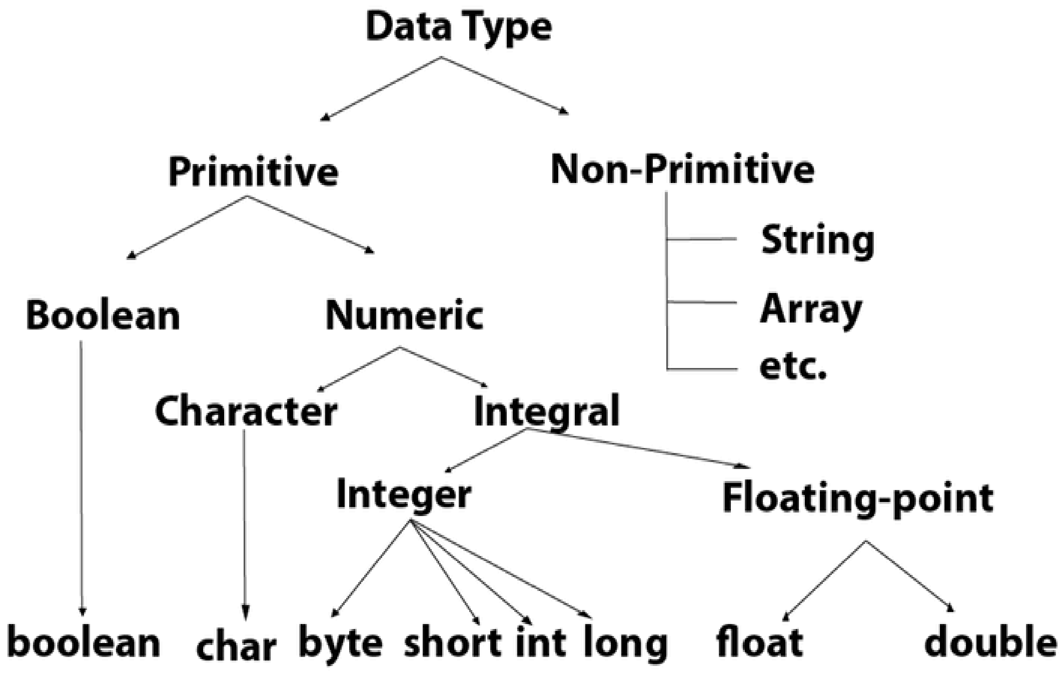
Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

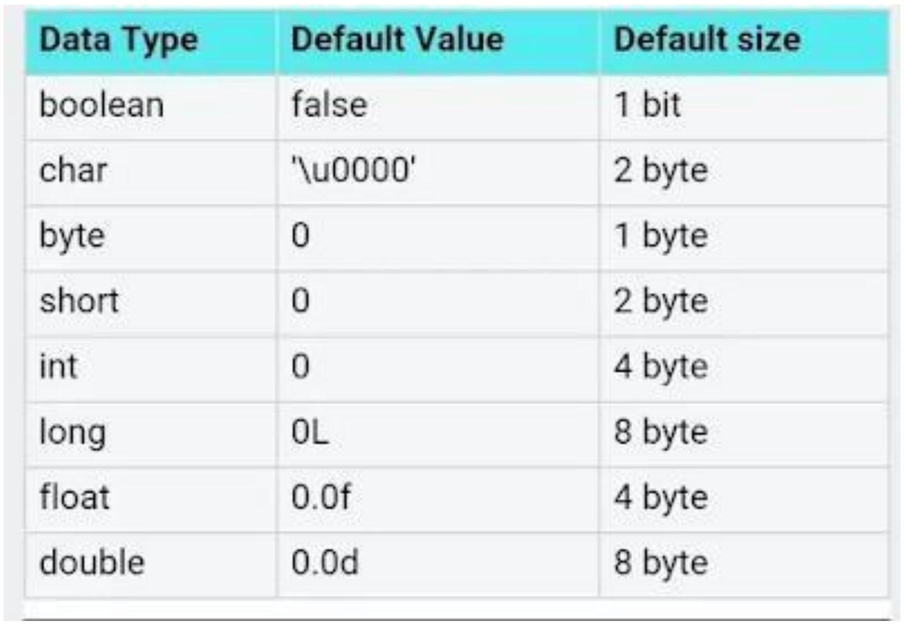
Data Types

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

1) Primitive data types
2) Non-primitive data type



}



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Note: char default value '\u0000' indicates nul

Variables and Data Types in Java

Variable is a reference for memory location. There are three types of variables in java - local, instance and static.

Types of Variable :

There are three types of variables in java: o local variable

1) instance variable
2) static variable
3) Local Variable

A variable which is declared inside the method is called **local variable**. The scope and lifetime are limited to the method itself. the arguments of a function will also be treated as local variables to that method

```
void m(int a, int b)
{
int sum = a+b  // here a,b and c all three are local variables to function/method – m
}
```

Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object.

Static variable

A variable that is declared as static is called static variable. It cannot be local variable. It is a variable which belongs to the class and not to object(instance). These variables will be initialized first, before the initialization of any instance variables.

A single copy to be shared by all instances of the class

A static variable can be accessed directly by the class name and doesn't need any object

Example to understand the types of variables in java class A

```
{
int data=50;    //instance variable static int m=100;   //static variable void method()
{
int n=90;       //local variable
}
}       //end of class A
```

Constants in Java

A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword.

Syntax

```
final dataType variableName = value;          final int a =10;
static final dataType variableName = value;  static final int b = 40;
```

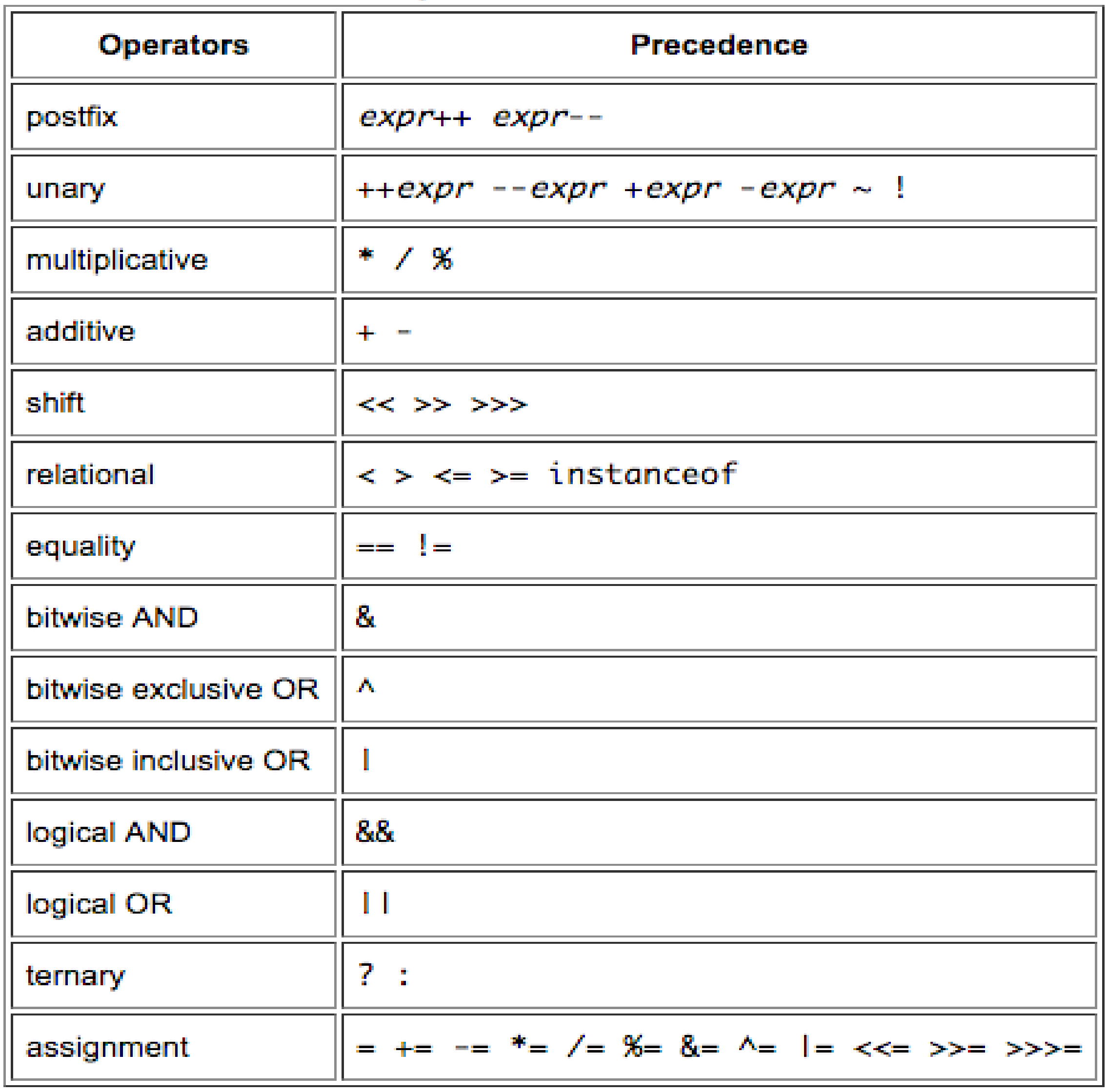
Operators in java

Operator in java is a symbol that is used to perform operation over the operands. For example: +, -, *, / etc.

There are many types of operators in java which are given below: o Unary Operator,

1. Arithmetic Operator,
2. shift Operator,
3. Relational Operator,
4. Bitwise Operator,
5. Logical Operator,
6. Ternary Operator and
7. Assignment Operator

Operator Precedence



Operators	Precedence
postfix	*expr++ expr--*
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Expressions

Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable.

Expressions can be built using values, variables, operators and method calls.

Types of Expressions -

While an expression frequently produces a result, it doesn't always. An expression can be Those that produce a value, i.e. the result of (1 + 1)

Those that assign a variable, for example (v = 10)

Java Type casting and Type conversion

Implicit type casting/Automatic Type Conversion -

Also known as widening conversion takes place when two data types can be automatically converted without any loss of data. This happens

when:

The two data types are compatible.

When we assign value of a smaller data type to a bigger data type.

For Example,

In java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char (or) 15boolean. Also, char and 15boolean are not compatible with each other.

type conversion/Implicit type casting example int a; short b = 50;
a = b; // conversion from b to a, smaller type to larger

Narrowing or Explicit Conversion -

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.Here, target-type specifies the desired type to convert the specified value to.

example,

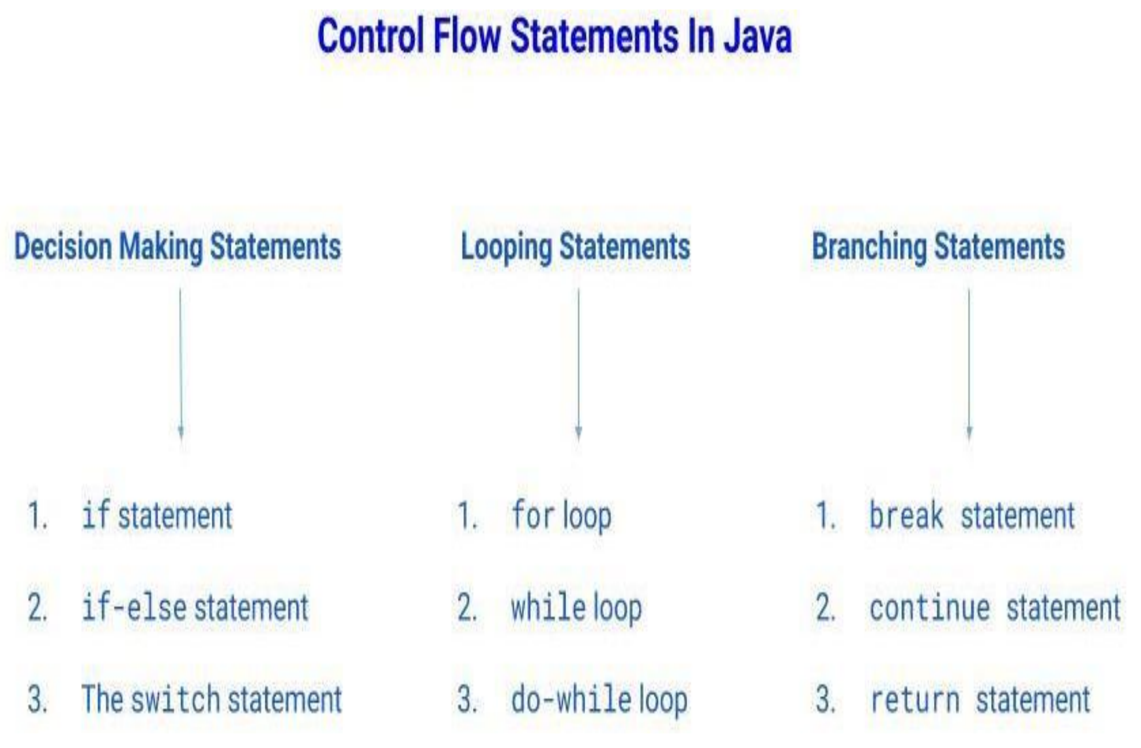
```
    int a = 10;
    byte b;
   b = (byte) a;       //we are explicitly mentioning to convert a of type int to byte and then assign
to b
```

Control Flow Statements

The control flow statements let you control the flow of the execution in your program. Java programming language, supports decision making, branching, looping, and adding conditional blocks.

The control flow statements can be categorized into

1) Decision Making Statements
2) Looping Statements
3) Branching Statements



Decision Making Statements

decision-making statements are used when we have to change the flow of execution based on a condition

There are three types of decision-making statements.

1) if statement
2) if-else statement
3) The switch statement

if statement

if statement is the most used decision-making statement in the java programming language.

syntax :

```
if(condition)
{
// code to be executed
}
```

if-else statement

it is similar to the if statement, here we will add a block of statements to be executed when condition fails, which will be written under else case. If the value of the condition statement is true, then if block will be executed, otherwise the else block will be executed.

```
if (condition statement) {
// code to be executed
}
else {
// code to be executed
}
```

Nested if- else

In Nested if – else we can add one if-else block in another if-else block. In following we are adding an inner if-else block in the outer if block.

```
if(condition 1)
{
code to be executed
}
else
{
if(condition 2)
{
code to be executed -
}
else
{
code to be executed
}
}
```

Switch statement

In the switch statement, there could be several execution paths, each block the control will be transferred based on case value

```
switch(week)
{
case 1:
printf("Monday"); break;
case 2:
printf("Tuesday"); break;
case 3:
printf("Wednesday"); break;
case 4:
printf("Thursday"); break;
case 5:
printf("Friday"); break;
case 6:
printf("Saturday"); break;
case 7:
printf("Sunday"); break;
default:
printf("Invalid input! Please enter week number between 1-7.");
}
```

In the above example switch will receive an integer value in variable week, based on week value corresponding case statements will be executed. if value doesn’t match with any case value the default will be executed.

Looping Statements

In looping statements, we are making a decision and executing the block of code multiple times. Until the condition is true, we are looping over the block of the code.

Each time we will check if the result of our decision statement is true or not, until and unless the result is true, we will execute the block of the code.

We can classify the lopping statements as follows:

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

1) **for loop**

In the for loop, we are going to check the value of the condition statement. If the value is true, the block of the code will be executed. After the successful execution of the code block, control again goes to the condition statement. Now, if the value is true, again the block of the code will be executed. Also, we are declaring one variable which stores the number of iteration. Each time the for loop runs, the value of i will be increased or decreased.

```
for(init variable declaration ; condition ; increment /decrement){

// code to be executed
}
```

while loop

In a while loop, we do apply the initialization, condition statement and an increment or decrement operator like the for loop but the syntax differs

```
init variable declaration while (condition){

// code to be executed
// increment or decrement statement
}
```

Example :

```
int i = 1; while(i<=5)  {
System.out.println( i ); i++;
}
System.out.print("End of while loop");
```

In the first line of the code, we are initializing a variable i of integer type with the value 1. In the next statement, we are checking the condition, if the value of the condition statement is true, the block of code will be executed.

When looking at the block of the code, at the end of the block you can see an increment operator, the value of i will increase by one, and the control goes to the condition statement. If the value of i is less than the 5, the block of code executes repeatedly.
When the value of the condition statement is false, the control goes to the next line of code after the while loop.

do -while loop

In the while loop, we are checking the condition statement first and then executing the block of code. But in the do-while loop, we are first executing the block of code and then checking the condition. If the value of the condition statement is true, the control goes at the starting of the code block, and the whole block of the code will be executed. Once the value of the condition statement is false, the control goes to the next line of code after the do-while loop.

```
init variable declaration do {
//code to be executed
} while ( condition statement);
```

The difference between the while and do-while loop is, in a while loop, we check the condition and executes the block of code, but in the do-while loop, we first execute the block of code and then check the condition.

Branching Statements

1. break statement
2. continue statement
3. return statement

break statement

break statement terminates the control flow. Usually, we do use the break statement to terminate the flow of for, while and do-while loop.

```
for(int i = 0; i<=5; i++)
{
if(i == 4) { break;
}
System.out.println( i );
}
```

The above code will print number from 1 to 3 and when i value becomes 4, the break statement will be executed and terminates from loop

continue statement

continue statement skips the current iteration of the for, while and do-while loop. The simple continue statement skips the iteration of the loop and sends the control back to the condition statement. The code after the continue statement will not be executed for the current iteration

```
for(int i = 0; i<=5; i++)
{
if(i == 4) { continue;
}
System.out.print( i );
}
```

The above loop will print numbers from 1 to 3, when i equals 4, the following statements will be skipped and controller will go back to the condition to execute next iteration of statements.It will print 1 2 4 5

4. return statement

In general return statement will be last line of the method. The return statement exits from the current method and control flow return to the line from which the method was invoked.

```
void m1()
{
int a = 5; int b = 6; int sum;
sum = add(a,b);
System.out.println("result is "+ sum)
}

int add(int I, int j)
{
int s = i+j;
return s;
}
```

In the above example when add function is called, controller will reach the add function, after executing statements of add, the return statement will return the controller back to calling method, using return we can even return values to calling method.

UNIT-II:

Classes and Objects: Classes and objects, Class declaration, Creating objects, Methods, Constructors and Constructor Overloading, Importance of Static Keyword and Examples, this Keyword, Arrays, Command Line Arguments, Nested Classes.

Classes and Objects:

Java achieves oops principles with the help of classes and objects.

Class: A class is a blueprint from which the individual objects are created. It represents the state and behaviour of an entity (anything that has existence with some properties and behaviour like car, student, laptop, bag, university, player, etc). A class is a logical entity

Object: An Object is the instance for a class (instance – initialization). An Object is a physical entity because it has memory allocated for all variables in the class. we can create any no of objects for the single class.

example:

let say I want to store 60 student's information like roll_num, name, age, marks. before knowing their property values, we will define a prototype suitable for them called as 'class'.

A class is declared using the keyword 'class'. all the properties and functions related to an entity are defined with in that class.

create class Student -

```
class Student
{
        String roll_num;
        String name;
        int age;
        int marks;
        void writeExams()
        {
        ::::::
        }
        void attendClasses()
        {
        ::::::
        }
}       // end of class
```

Now whenever we want to store a student information, we request memory for storing the property values, the allocated memory to store student information is called as 'object/instance'.

General Form of a Class:

A class generally contain three sections - variables, constructors and methods.

Variables represent its state/properties in form of fields. Class can have static and instance variables. functionality/behaviour will be implemented in methods under a class. Class can have static and instance methods. Constructors will initialize the instance variables of a class when an abject is created.

General form or Syntax of a Class

```
class NameOfClass
{
        // instance variable declaration
        type1 varName1 = value1;
        type2 varName2 = value2;
        :
        :
        typeN varNameN = valueN;

        //  Constructors
        // no argument constructor
        NameOfClass()
        {
                // body of constructor
        }
        :
        :
        // constructor with arguments, we can define any more than one constructor to same
same
        NameOfClass (cparamN)
        {
                // body of constructor
        }
        // Methods
        static returnType1 methodName1(mParams1)
        {
                // body of static method
        }
        :
        :
        returnTypeN methodNameN(mParamsN)
        {
                // body of instance/non-static method
        }
}
```

class is keyword used to define the new class

Variables named varName1 through varNameN are declared inside the class and outside of all methods are called instance variables. Each variable must be assigned a type shown as type1 through typeN(any primitive data type like int,float,char,..) and may be initialized to a value shown as valueN. if a variable is declared by keyword static then they belong to all objects called as static or class variables (we will cover about static later, in this chapter)

Constructors are used to initialize the instance variables of an object, executed when an object is created. It always has the same name as the class. They will not return any value. A constructor can be defined with or without arguments. We can define more than one constructor for a class.

Methods named mthName1 through mthNameN should be defined with in the class. A static method can be called directly from main/any static method. a non-static method should be called through an object only.

when we want to run a class, a main method should be defined in it with following signature. Java program execution starts from main method. A main method is the starting point for execution by JVM.

```
        public static void main(String arguments[])
        {
        }
```

Declaring Objects using new

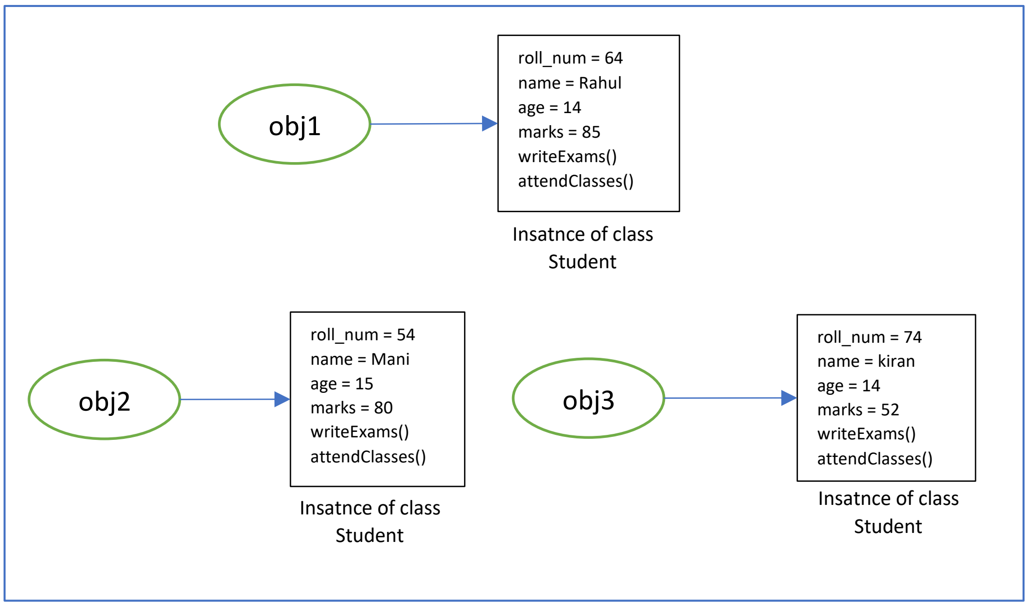
The *new* operator instantiates a class by dynamically allocating (i.e, allocation at run time) memory for a new object and returning a reference to that memory. This reference is then stored in the variable declared with type class name.

creating objects for class Student –

an object for class can be created using the key word **'new'**

```
Student obj1 = new Student ();
Student obj2 = new Student ();
Student obj3 = new Student ();
```

```
obj1.roll_num = 64;    obj1.roll_num = "Rahul";    obj1.age = 14;    obj1.age = 85;
obj2.roll_num = 54;    obj2.roll_num = "Mani";     obj2.age = 15;    obj2.age = 80;
obj3.roll_num = 74;    obj3.roll_num = "Kiran";    obj3.age = 14;    obj3.age = 52;
```



As shown in above diagram whenever a new object is created using new, an instance for class will be created, and can be accessed through class reference variables like obj1, obj2, obj3.

Write a program in single java file, which contains two classes Teacher and Student,

Student should have properties rollnum, name and marks. and functions increment, decrement which will increment and decrement marks by 10.

In class Teacher write a main method, in that create two student class objects initialize their properties and later call increment() on 1st student object and call decrement() on second Student object.

```
class Student
{
        int roll_num;
        String name;
        int marks;
        void increment()
        {
                marks = marks + 10;
        }
        void decrement()
        {
                marks = marks - 10;
        }
}

class Teacher
{
   public static void main(String args[])
   {
        Student s1 = new Student();
        Student s2 = new Student();

        s1.roll_num = 54;
        s1.name = "Rahul";
        s1.marks = 55;

        s2.roll_num = 64;
        s2.name = "Kiran";
        s2.marks = 80;

        System.out.println("intial marks of student "+s1.name+" are "+s1.marks);
        s1.increment();
        System.out.println("after increment marks of student "+s1.name+" are
"+s1.marks+"\n");
        System.out.println("intial marks of student "+s2.name+" are "+s2.marks);
        s2.decrement();
        System.out.println("after decrement marks of student "+s2.name+" are "+s2.marks);
    }

}
```

OutPut –

```
intial marks of student Rahul are 55
after increment marks of student Rahul are 65
intial marks of student Kiran are 80
after decrement marks of student Kiran are 70
```

Methods –

A method is a set of statements written with in the block to perform certain operation. Methods allow us to **reuse** the code, we can call a method any no of times to repeat the execution of statements under it. A method must be declared within the class. Following is the syntax to define a method.

```
modifier returnType nameOfMethod (Parameter List)
{
  // method body
}
```

The components in a method include −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return value of any type. if returns nothing type is void.
- **nameOfMethod** − This is name of the method.
- **Parameter** List − The list of parameters, also called as arguments. A method can have any no of arguments. These are optional
- **method body** − The method body defines statements for implementing the functionality.

Let’s practice some programs on methods for better understanding.

Write a program with class name MethodExample, this class should contain a method called display, which prints “This is an example for method ” and it returns nothing.

```
class MethodExample
{
        void display()
        {
                System.out.println(“This is an example for method”);
        }
        public static void main(String args[])
        {
                ~~display ();~~ // a non-static method cannot be called without an object
                MethodExample obj = new MethodExample();
                **obj.display();**
        }
}
```

```
OutPut:
This is an example for method
```

Write a program with class name MethodExample, this class should contain a method called display, it takes a String parameter of user name and prints “welcome to methods username“ and it returns nothing.

```
class MethodExample
{
        void display(String userName)
        {
                System.out.println("Welcome to methods " + userName);
        }

        public static void main(String args[])
        {
                MethodExample obj = new MethodExample ();
                obj.display("Rahul");           // string literal is passed directly

                String name = "kiran";
                obj.display(name);      // String literal is stored in a variable and then passed
        }
}
```

```
OutPut:
Welcome to methods Rahul
Welcome to methods kiran
```

Write a program with class name MyMethod, this class should contain a method called sum, it takes two integer parameters and prints their sum and it returns nothing.

```
class MyMethod
{
        void sum(int num1, int num2)
        {
                int sum;
                sum = num1+num2;
                System.out.println("sum of variables is " + sum);
        }

        public static void main(String args[])
        {
        MyMethod obj = new MyMethod ();
        obj.sum(2,4);           // integer literals are passed directly

        int i=12,j=4;
        obj.sum(i,j);           // integer literals are stored in variables and then passed
        }
}
```

```
OutPut:
sum of variables is 6
sum of variables is 16
```

Write a program with class name MyMethod, this class should contain a method called sum, it takes two integer parameters and returns their sum, you should print sum value in main method after completing the function call.

```
class MyMethod
{
        int sum(int num1, int num2)
        {
                int sum;
                sum = num1+num2;
                return sum;
        }
        public static void main(String args[])
        {
        MyMethod obj = new MyMethod ();
        int result;
        result = obj.sum(2,4);
        System.out.println("received sum value from function "+result);
```

```
		int i=12,j=4;
		result = obj.sum(i,j);
		System.out.println("received sum value from function "+result);
		}
}
```

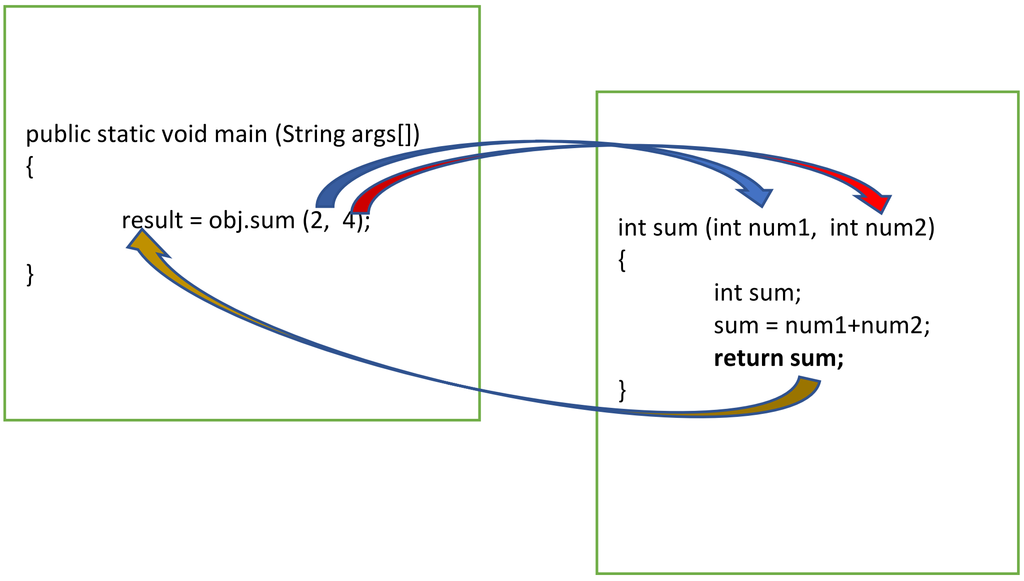
OutPut:

```
received sum value from function 6
received sum value from function 16
```

Note:

***** We can return a value or variable with keyword return.

* It is must to hold the returned value at function call. In our example the sum is returned and received value is captured under the variable result.


public static void main (String args[])
{
result = obj.sum (2, 4);
}
int sum (int num1, int num2)
{
int sum;
sum = num1+num2;
return sum;
}

Constructors in Java

Constructors are used to initialize the instance variables of class, at the time of creating object.

rules to be followed for defining a constructor

1. Constructor name must be the same as its class name
2. Constructor **should not** be associated with any return type, not even void. A constructor will never return anything.

There are two types of constructors in Java:

1. Default constructor (constructor without any arguments)
2. Parameterized constructor (constructor with arguments)

Default Constructor (constructor without any arguments)

A constructor that has no parameter is known as default constructor. If we don't define a constructor in a class, then compiler creates **default constructor** for that class.

Write a program with class name MyDefaultConstructor, which have a default constructor that prints "My program using default constructor".

```
class MyDefaultConstructor
{
        MyDefaultConstructor ()
        {
                System.out.println(“My program using default constructor”);
        }

        public static void main(String args[ ])
        {
                MyDefaultConstructor obj = new MyDefaultConstructor();
        }

}
```

OutPut :

```
My program using default constructor.
```

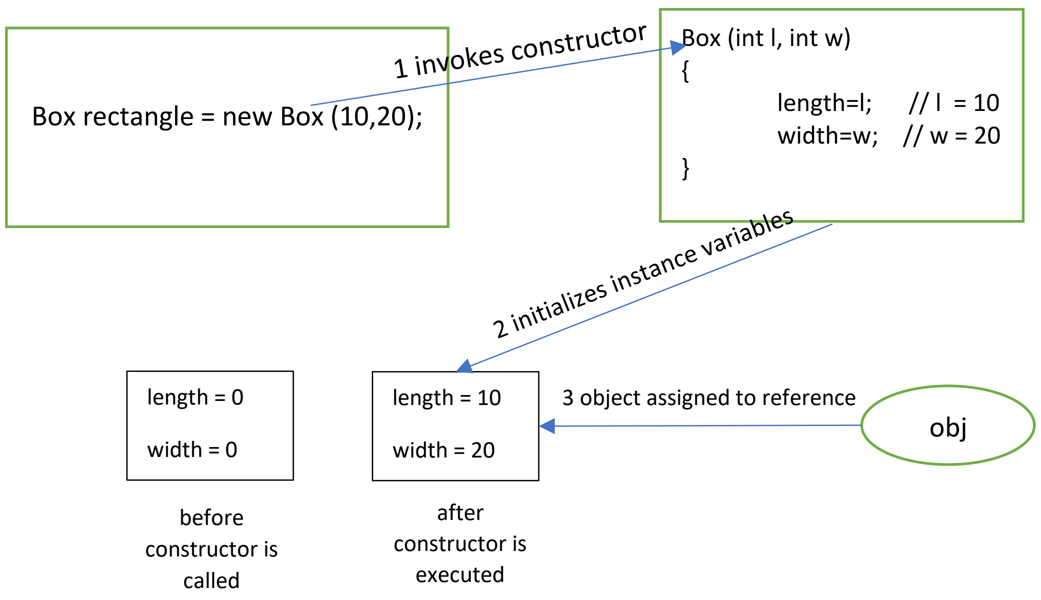
<u>Parameterized constructor</u> <u>(constructor with arguments)</u>

A constructor that has parameters is known as parameterized constructor. If we want to initialize variables of the class with your own values, then use a parameterized constructor.

Write a program with class name Box, define a parameterized constructor to initialize instance variables. write a method areRectangle to calculate area of rectangle.

```
class Box
{
        int length;
        int width;

        //Constructor for rectangle when two parameters are passed.
        Box (int l, int w)
        {
                length=l;
                width=w;
        }

        int areaRectangle()
        {
                return length*width;
        }

        public static void main(String args[])
        {
                Box obj = new Box (10,20);
                int area;
                area = obj.areaRectangle();
                System.out.println("Area of rectangle is " + area);
        }
}
```

OutPut:
Area of rectangle is 200


Box rectangle = new Box (10,20);
1 invokes constructor
Box (int l, int w)
{
length=l;   // l = 10
width=w;   // w = 20
}
2 initializes instance variables
length = 0
width = 0
before constructor is called
length = 10
width = 20
after constructor is executed
3 object assigned to reference
obj

constructor overloading

Constructor overloading means defining two or more constructors for same class. The name of all constructors will be same, only number of arguments or type of arguments will be different.

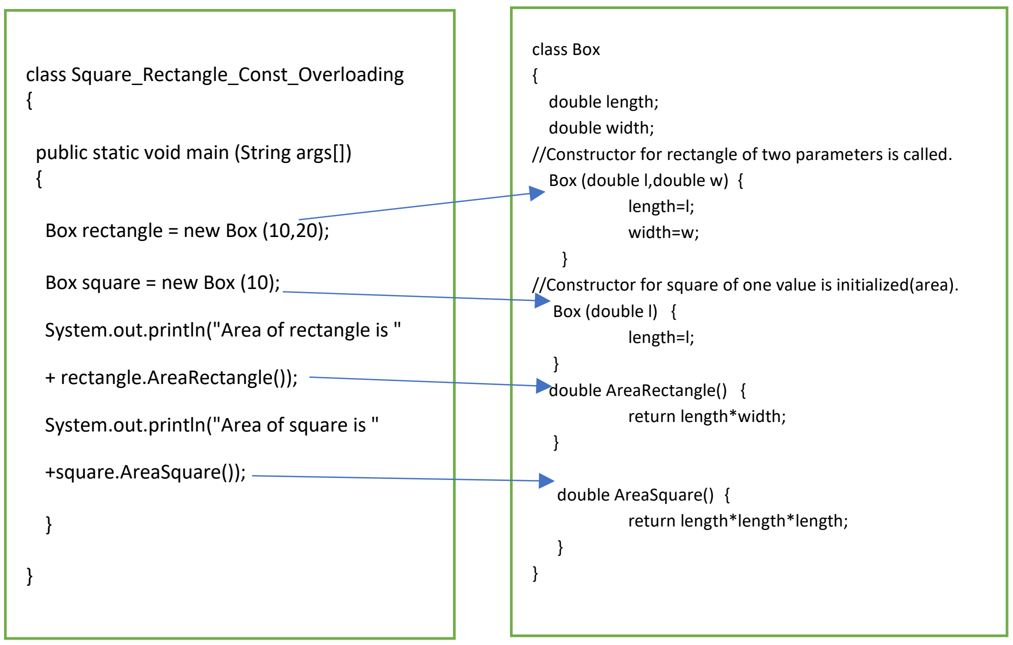
when you pass arguments, the constructor which matches number of arguments and type of arguments will be executed automatically.

Write a program to find area of square and rectangle, with help of constructor overloading.

```
class Box
{
        double length;
        double width;

        //Constructor for rectangle when two parameters are passed.
        Box (double l,double w)        {
                length=l;
                width=w;
        }
        //Constructor for square when one value is initialized(area).
        Box (double l)  {
                length=l;
        }
        double AreaRectangle() {
                return length*width;
        }
        double AreaSquare()     {
                return length*length*length;
        }
}
        //Main class from where program execution begins
class Square_Rectangle_Const_Overloading      {
        public static void main (String args[])       {
        Box rectangle = new Box (10,20);
        Box square = new Box (10);
        System.out.println("Area of rectangle is " + rectangle.AreaRectangle());
        System.out.println("Area of square is " +square.AreaSquare());
        }
}
```

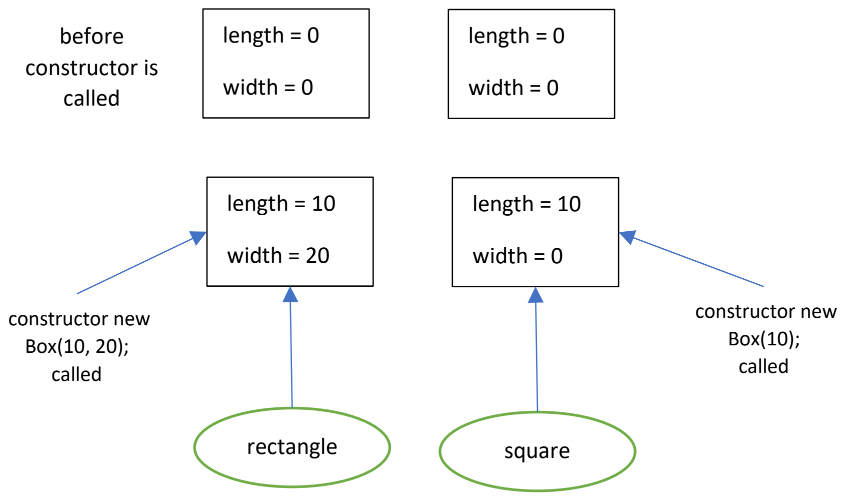
OutPut:

```
Area of rectangle is 200.0
Area of square is 1000.0
```



```
class Square_Rectangle_Const_Overloading
{

 public static void main (String args[])
 {

  Box rectangle = new Box (10,20);

  Box square = new Box (10);

  System.out.println("Area of rectangle is "

  + rectangle.AreaRectangle());

  System.out.println("Area of square is "

  +square.AreaSquare());

  }

}
```

```
class Box
{
   double length;
   double width;
//Constructor for rectangle of two parameters is called.
   Box (double l,double w)  {
                length=l;
                width=w;
       }
//Constructor for square of one value is initialized(area).
   Box (double l)   {
                length=l;
     }
   double AreaRectangle()   {
                return length*width;
     }

    double AreaSquare()  {
                return length*length*length;
     }
}
```



before constructor is called
length = 0
width = 0
length = 0
width = 0
length = 10
width = 20
length = 10
width = 0
constructor new Box(10, 20); called
constructor new Box(10); called
rectangle
square

Method Overloading:

Method Overloading (or) static polymorphism allows a class to have more than one method to have the same name, if their argument lists are different.

A method is said to be overloaded, if two methods with same name differ by any of these

1. Number of parameters
   example:
   ```
           add(int a, int b)          // function have two arguments
           {
                 :::::::
           }

           add(int, int, int)         // function have three arguments
           {
                  :::::::::
           }
   ```
2. Data type of parameters
   example:
   ```
           add(int a, int b)          // function have two int arguments
           {
                 :::::::
           }

           add(int a, float b)        // function have one int argument and one float argument
           {
                  :::::::::
           }
   ```
3. Order of parameters
   example:
   ```
           add(int a, float b, int c)          // function have two arguments
           {
                 :::::::
           }

           add(int a, int c, float b)          // function have three arguments
           {
                  :::::::::
           }
   ```

Write a program to print sum of given numbers, implement it through method overloading.

```
class MethodOverloading
{

        void sum (int a, int b)
        {
```

```
                int sum;
                sum = a+b;
                System.out.println("method sum declared with two integer arguments");
                System.out.println("sum is "+sum);
        }



        void sum (int a, int b, int c)
        {
                int sum;
                sum = a+b+c;
                System.out.println("this method differs by number of arguments");
                System.out.println("sum is "+sum);
        }

        void sum (int a, float b)
        {
                float sum;
                sum = a+b;
                System.out.println("this method differs by type of arguments");
                System.out.println("sum is "+sum);
        }

        void sum (float b, int a)
        {
                float sum;
                sum = a+b;
                System.out.println("this method differs by order of arguments");
                System.out.println("sum is "+sum);
        }
        public static void main (String args[])
        {
                MethodOverloading obj = new MethodOverloading();
                obj.sum(2, 4);
                obj.sum(2, 4, 6);
                obj.sum(2, 3.4f);
                obj.sum(3.4f, 2);
        }

}
```

OutPut:

```
method sum declared with two integer arguments
sum is 6
this method differs by number of arguments
sum is 12
this method differs by type of arguments
sum is 5.4
this method differs by order of arguments
sum is 5.4
```

Note:

A method will not be overloaded by return type of that method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading. It will be a compilation error “method has been already defined”.

Static keyword:

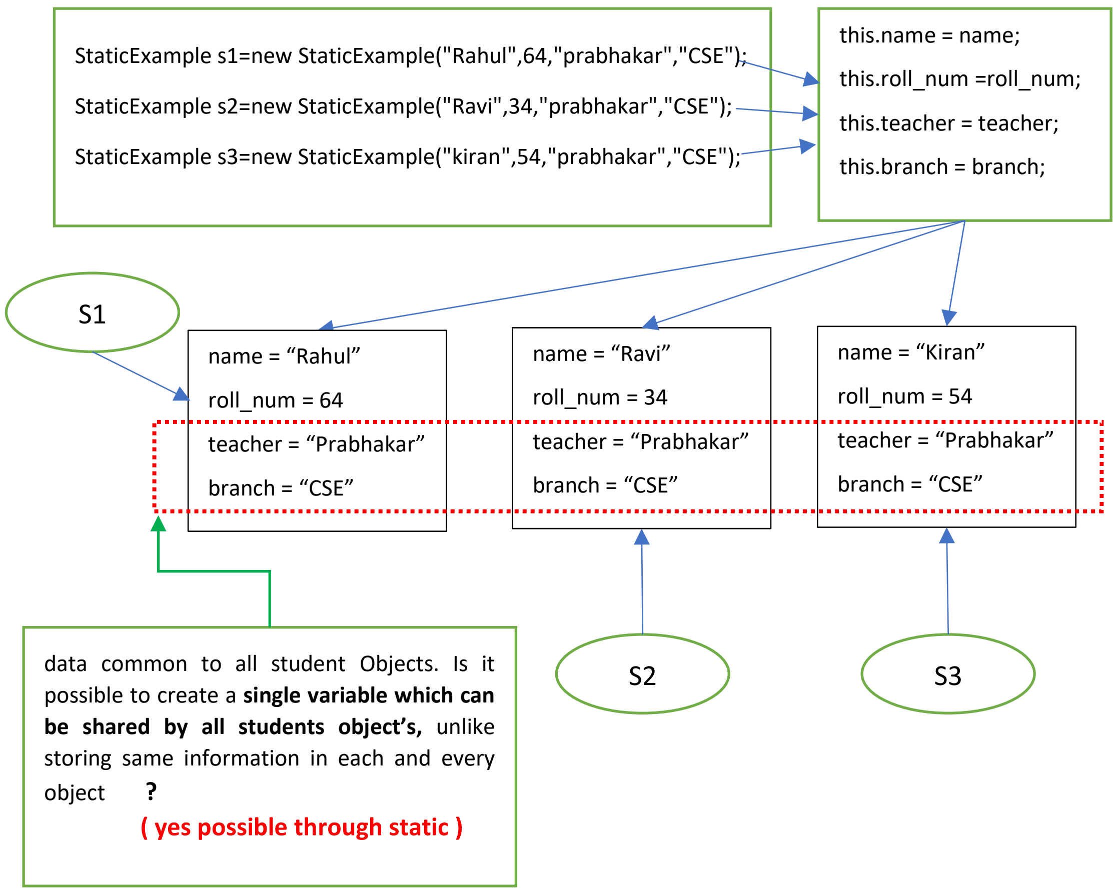
When a member is declared static, it can be accessed before any objects of its class are created. We can apply static keyword with variables, methods, blocks and nested class.

why static variables?

write a program to define student class with properties name, rollnum, teacher, branch. create 3 student objects with different properties initialized through a constructor. print all student details.

```
public class StaticExample
{

        String name;
        int roll_num;
        String teacher;
        String branch;

        StaticExample(String name, int roll_num, String teacher, String branch)
        {
                this.name = name;
                this.roll_num = roll_num;
                this.teacher = teacher;
                this.branch = branch;
        }

        void displayDetails()
        {
                System.out.println("name is "+name+" roll_num is "+roll_num+" teacher is "
                                        +teacher+" branch is "+branch);
        }

        public static void main(String args[])
        {
                StaticExample s1=new StaticExample("Rahul",64,"prabhakar","CSE");
                StaticExample s2=new StaticExample("Ravi",34,"prabhakar","CSE");
                StaticExample s3=new StaticExample("kiran",54,"prabhakar","CSE");


                s1.displayDetails();
                s2.displayDetails();
                s3.displayDetails();
        }
}
```

OutPut:
```
name is Rahul roll_num is 64 teacher is prabhakar branch is CSE
name is Ravi roll_num is 34 teacher is prabhakar branch is CSE
name is kiran roll_num is 54 teacher is prabhakar branch is CSE
```



data common to all student Objects. Is it possible to create a **single variable which can be shared by all students object’s,** unlike storing same information in each and every object **?**
(yes possible through static)

Like this, if I’m going to create total of 195 student objects. then lot memory will be misused to store redundant(repeated) information. As a good programmer we should always care about memory and execution time. Either it may be ‘program statements’ (or) ‘data’, we should never repeat.

static variables:

When a variable is declared as static, then a single copy of variable is created and shared among all objects.

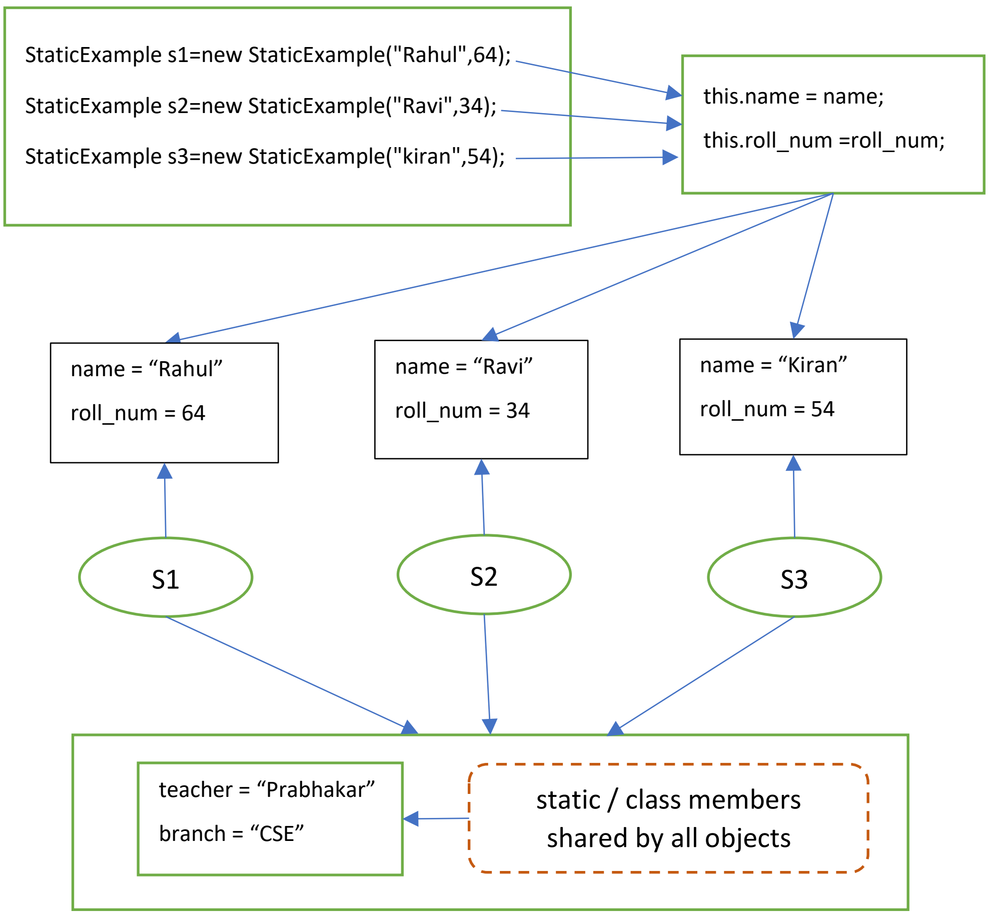
For the above program, to store student details declare the properties teacher and branch as static and implement the same functionality

```
public class StaticExample
{

        String name;
        int roll_num;
        static String teacher = "Mr. Prabhakar";
        static String branch = "CSE";
```

```
          StaticExample(String name, int roll_num)
          {
                    this.name = name;
                    this.roll_num = roll_num;
          }

          void displayDetails()
          {
                    System.out.println("name is "+name+" roll_num is "+roll_num+" teacher is
                              "+StaticExample.teacher+" branch is "+StaticExample.branch);
          }

          public static void main(String args[])
          {
                    StaticExample s1=new StaticExample("Rahul",64);
                    StaticExample s2=new StaticExample("Ravi",34);
                    StaticExample s3=new StaticExample("kiran",54);

                    s1.displayDetails();
                    s2.displayDetails();
                    s3.displayDetails();
          }
}
```

OutPut:

```
name is Rahul roll_num is 64 teacher is Mr. Prabhakar branch is CSE
name is Ravi roll_num is 34 teacher is Mr. Prabhakar branch is CSE
name is kiran roll_num is 54 teacher is Mr. Prabhakar branch is CSE
```



StaticExample s1=new StaticExample("Rahul",64);
StaticExample s2=new StaticExample("Ravi",34);
StaticExample s3=new StaticExample("kiran",54);
this.name = name;
this.roll_num =roll_num;
name = "Rahul"
roll_num = 64
name = "Ravi"
roll_num = 34
name = "Kiran"
roll_num = 54
S1
S2
S3
teacher = "Prabhakar"
branch = "CSE"
static / class members shared by all objects

umashankar.5544@gmail.com

In the above program, member data teacher and branch are declared static, so only one variable for each teacher and branch will be created at class level. This static data will be shared by all student objects. it is programmer responsibility to identify which data (or) methods will be common to all objects and mark them as static, whereas data specific for individual objects should be non-static. here rollnum and name of student cannot be shared they are individual for each student object called as instance (or) object data.

Note:

1. static variables can be accessed from static blocks and static methods only.

static members can be accessed

i) directly without an object ii) with class name iii) using object

2. static members cannot invoke or access non static members.

static methods:

A static method can be invoked without creating an instance of class, they can be accessed directly or by class name.

A static method can access static data members only.

write a program with static method display, it accepts an integer number and print the square value of it, call the method display from main function by passing value 5.

```
class StaticExample
{
	static void display(int n)
	{
		int res;
		res = n*n;
		System.out.println("square value of "+ n +" is "+res);
	}

	public static void main(String args[])
	{
		display(5);
	}

}
```

OutPut:

```
square value of 5 is 25
```

static block:

whenever a java class file is executed, static block and static methods will be loaded. From static blocks and static methods, static block will be executed first. Later among the static methods JVM will start executing main method with following signature
public static void main(String args[ ])

static blocks are used to initialize the static variables.

Write a program with any print statement in static block.

```
class StaticExample
{
        static
        {
          System.out.println("static block will be executed first");
        }
        public static void main(String args[])
        {
                System.out.println("main will be executed after static block");
        }
}
```

OutPut:

```
static block will be executed first.
main will be executed after static block.
```

this keyword:

'this' is a reference, that refers to the current class object.

write a program to initialize instance variables in constructor.

```
class Example
{
        int a;
        float b;

        Example(int j, float k)
        {
                a = j;
                b = k;
        }

        public static void main(String args[ ])
        {
                Example obj = new Example(5,5.4f);
                System.out.println(obj.a);
                System.out.println(obj.b);

        }
}
```

OutPut:

```
5
5.4
```

what if variables declared inside constructor j, k are named with same name as instance variables i.e., a, b. then how to initialize instance variables ?

```
class Example
{
	int a;
	float b;

	Example(int a, float b)
	{
		a = a;
		b = b;
	}

	public static void main(String args[ ])
	{
		Example obj = new Example(5,5.4f);
		System.out.println(obj.a);
		System.out.println(obj.b);

	}
}
```

OutPut:

```
0
0.0
```

though constructor is defined to initialize instance variables, it has no effect because,

inside constructor

```
a = a;
b = b;
```

both at lhs and rhs a,b refers to same local variables that are declared inside constructor.

we can refer to instance variables using this operator, that refers to current object

```
class Example
{
	int a;
	float b;

	Example(int a, float b)
	{
		this.a = a;
		this.b = b;
	}

	public static void main(String args[ ])
	{
		Example obj = new Example(5,5.4f);
		System.out.println(obj.a);
		System.out.println(obj.b);

	}
}
```

```
OutPut:
5
5.4
```

this can also be used

1. to invoke current class method.
2. to invoke current class constructor.
3. this can be passed as an argument in the method call.

invoke current class method using this.

```
class ThisMethod
{
	void display()
	{
		System.out.println("from method display");
	}
	void n()
	{
		System.out.println("calling method using this");
		this.display();
	}

}
class ThisExample
{
	public static void main(String args[])
	{
		ThisMethod a=new ThisMethod();
		a.n();
	}
}
```

OutPut:

```
calling method using this
from method display
```

invoke current class constructor

The this() constructor call can be used to invoke another constructor of same class.

Program to invoke no argument constructor from parameterized constructor using this()

```
class ThisConstructor
{
	ThisConstructor()
	{
		System.out.println("From Constructor with no arguments.");
	}
	ThisConstructor(int x)
	{
		this();
		System.out.println("controller back to Parameterized constructor");
		System.out.println("value passed to constructor is "+x);
	}
}
class ThisExample
{
	public static void main(String args[])
	{
		ThisConstructor a = new ThisConstructor(10);
	}
}
```

OutPut:

```
From Constructor with no arguments.
controller back to Parameterized constructor
value passed to constructor is 10
```

Program to invoke parameterized constructor from, no argument constructor using this(argument)

```
class ThisConstructor
{
	ThisConstructor()
	{
		this(20);
		System.out.println("controller back to no argument constructor");

	}
	ThisConstructor(int x)
	{
		System.out.println("From Constructor with arguments.");
		System.out.println("value received is "+x);
	}
}
class ThisExample
{
	public static void main(String args[])
	{
		ThisConstructor a = new ThisConstructor();
	}
}
```

OutPut:

```
From Constructor with arguments.
value received is 20
controller back to no argument constructor
```

passed as an argument in the method call

The this can also be passed as an argument in function call. It is mainly used in the event handling. Let's see the example:

```
class ThisExample
{
   int a;
   int b;

   ThisExample()
   {
      a = 10;
      b = 20;
   }

   void display(ThisExample obj)
   {
      System.out.println("a = " + obj.a + "  b = " + obj.b);
   }

   void get()
   {
      display(this);
   }

   public static void main(String[] args)
   {
      ThisExample object = new ThisExample();
      object.get();
   }
}
```

OutPut:

```
a = 10  b = 20
```

Arrays one Dimensional and multidimensional

Array: An array is a data structure that stores elements of same data type in contiguous memory location. Once array size is defined it cannot be increased (or) decreased. The first element of an array starts with the index 0.

why arrays?
when you want store student marks, we will declare a variable and store value

int studentMarks;   studentMarks = 92;

what if we need to store 90 students' marks, are we going to create 90 variables, oh it will be very difficult to program and maintain then. arrays will help us to store all these 90 student marks under same array name differed by array index (address location).

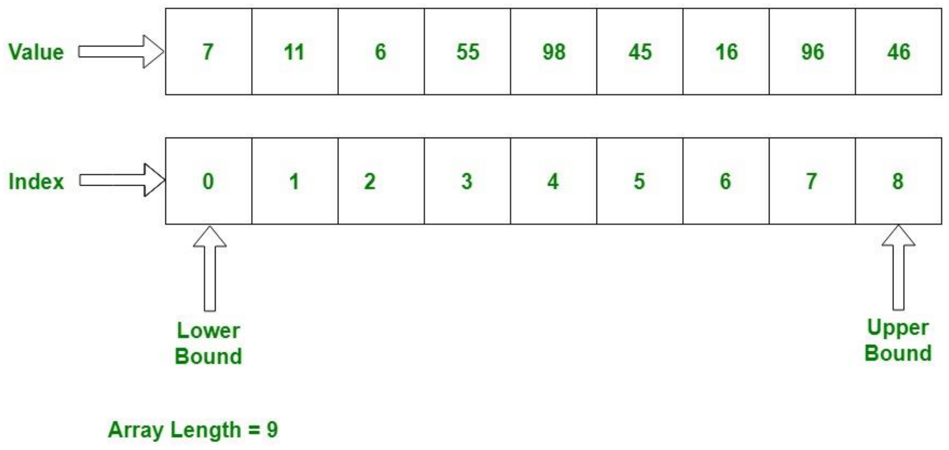
```
int studentMarks[ ] = new int[90];
```

studentMarks[0] = 80 // first element in array – marks of 1st student,

studentMarks[1] = 65 // second element in array – marks of 2nd student,

……………

studentMarks[89] = 90 // last element in array – marks of 90th student



Declaring an Array –

```
datatype[ ] arrayname = new datatype[size];
                    (or)
datatype arrayname[ ] = new datatype[size];
```

Initializing array -

```
arrayname[index] = value;
```

Example –

//To create an array with name myarray of size 3.

```
int myarray[ ] = new int[3];
```

Initializing

```
myarray[0] = 1;
myarray[1] = 2;
myarray[2] = 3;
```

Note: If the array elements are known at the time of array declaration, we can initialize array elements along with the declaration itself as

```
int myarray[ ] = {1,2,3,4,5};
```

an array, myarray will be created of size 5 with elements stored from index 0 to 4.

```
System.out.println(myarray[0])     // prints 1
System.out.println(myarray[4])     // prints 5
```

Write a program to declare an array with size n, then read n no of elements and store them in that array.

```
import java.util.Scanner;
class MyExample
```

```
{
        public static void main(String args[])
        {
                int n;
                Scanner sc = new Scanner(System.in);
        System.out.println("How many elements you want to store :")
                n = sc.nextInt();
                int ary[ ] = new int[n];
        // reads n elements from console, and stores in to array, ary
                System.out.println("Enter "+n+" no of elements");
                for(int i= 0; i<n;i++ )
                {
                        ary[i] = sc.nextInt();
                }
        // print each value stored in the array ary,
                System.out.println("elements stored in array are");
                for(int i = 0; i< n; i++)
                {
                        System.out.println(ary[i]);
                }
        }
}
```

OutPut:

```
How many elements you want to store : 4
Enter 4 no of elements: 2 4 6 8
elements stored in array are
2 4 6 8
```

Multidimensional arrays:

A multi dimensional array, is array for arrays. It has multiple levels. The simplest multi-dimensional array is the 2D array i.e., two-dimensional array

why multi-dimensional array?

when I want to store students' total marks, a one Dimensional array is enough. what if we want to store the 6 subject's marks of 90 students, we cannot accommodate them in one dimensional array, and we cannot declare 90*6 = 540 variables and memorize them.

In such a scenario, we can use 2-Dimensional array. one dimension to represent student number and other dimension to represent subject wise marks.

```
int studentSubjectWiseMarks[ ][ ] = new int[90][6];
```

A two-dimensional array can also be treated as rows and columns, in the above declaration studentSubjectWiseMarks we will have 90 rows and 6 columns for each row. where the row index and column index both starts from 0.

practice indexing,

let’s assign first student, six subject marks with 80, 60,56,70,86,90

```
studentSubjectWiseMarks[0][0] = 80;
studentSubjectWiseMarks[0][1] = 60;
studentSubjectWiseMarks[0][2] = 56;
studentSubjectWiseMarks[0][3] = 70;
studentSubjectWiseMarks[0][4] = 86;
studentSubjectWiseMarks[0][5] = 90;
```

second student, six subject marks with 70, 64,58,74,86,70

```
studentSubjectWiseMarks[1][0] = 70;
studentSubjectWiseMarks[1][1] = 64;
studentSubjectWiseMarks[1][2] = 58;
studentSubjectWiseMarks[1][3] = 74;
studentSubjectWiseMarks[1][4] = 86;
studentSubjectWiseMarks[1][5] = 70;
```

…………………………………………………………………

90th student, six subject marks with 90, 90,90,80,86,70

```
studentSubjectWiseMarks[89][0] = 90;
studentSubjectWiseMarks[89][1] = 90;
studentSubjectWiseMarks[89][2] = 90;
studentSubjectWiseMarks[89][3] = 80;
studentSubjectWiseMarks[89][4] = 86;
studentSubjectWiseMarks[89][5] = 70;
```

Note:

The important thing that we should be careful is with indexing of an element. If you understand how to identify an element with its index you can simply play with arrays.

The frequent error you will observe while dealing with arrays is Array Index Out of Bounds Exception.

int a[ ] = new int[5]
here lower bound index is 0, upper bound index is 4, total 5 elements.
if we try to access an element out of these bounds, we will get, Array Index Out of Bounds Exception.

a[0] = 1; a[1] = 2; a[2] = 3; a[3] = 4; a[4] = 5; These are perfectly valid.

a[-1] = 3; a[5] = 4; a[56] = 7; these will cause an Array Index Out of Bounds Exception.

Declaring 2 – Dimensional array:
int matrix[ ][ ] = new int[4][3];

Initializing 2-D array:
matrix[0][0] = 1;   matrix[0][1] = 2;   matrix[0][2] = 3;

matrix[1][0] = 4;   matrix[1][1] = 5;   matrix[1][2] = 6;

matrix[2][0] = 7;   matrix[2][1] = 8;   matrix[2][2] = 9;

matrix[3][0] = 10;   matrix[3][1] = 11;   matrix[3][2] = 12;

rows ↓	columns →	0	1	2
0		matrix[0][0]	matrix[0][1]	matrix[0][2]
1		matrix[1][0]	matrix[1][1]	matrix[1][2]
2		matrix[2][0]	matrix[2][1]	matrix[2][2]
3		matrix[2][0]	matrix[3][1]	matrix[3][2]

write a program to store elements from 1 to 12, using 2-D array of order 4*3
```
public class MatrixExample
{
        public static void main(String args[])
        {
                int matrix[ ][ ] = new int[4][3];
                int k=1;
                for(int i = 0; i<4; i++)                // represents rows
                {
                        for(int j = 0; j<3; j++)        // represents columns
                        {
                                matrix[i][j] = k;       // storing elements in array
                                k++;
                        }
                }
```

```
		// display all the elements in matrix array
		for(int i = 0; i<4; i++)
		{
			for(int j = 0; j<3; j++)
			{
				System.out.println(matrix[i][j]);
			}
		}
	}
}
```

OutPut :

1 2 3 4 5 6 7 8 9 10 11 12 (I know they will print line by line, don't want to make the document lengthy).

write a program that will ask the user for no of rows and no of columns, to declare 2-D array, and then ask user to enter elements for that matrix. finally print the elements in matrix format.

```
import java.util.*;
public class Example
{
	public static void main(String args[])
	{
		int rows, columns;
		Scanner sc = new Scanner(System.in);

		System.out.print("Enter no of rows: ");
		rows = sc.nextInt();			// reads no of rows

		System.out.print("Enter no of columns: ");
		columns = sc.nextInt();			// reads no of columns

	// Now declare 2-D array with specified rows and columns
		int matrix[][] = new int[rows][columns];
		System.out.print("Enter elements into matrix: ");
		for(int i = 0; i<rows; i++)		// represents rows
		{
			for(int j = 0; j<columns; j++)		// represents columns
			{
				matrix[i][j] = sc.nextInt();	// read element into the array
			}
		}

		// display all the elements in matrix array
		for(int i = 0; i<rows; i++)
		{
			for(int j = 0; j<columns; j++)
			{
```

```
                    System.out.print(matrix[i][j]+" ");
                }
                System.out.println();// line break, after printing each row elements
            }
        }
}
```

OutPut:

```
Enter no of rows: 3
Enter no of columns: 3
Enter elements into matrix: 1 2 3 4 5 6 7 8 9
1 2 3
4 5 6
7 8 9
```

*** Note : Finding length of an array, most important**
How to get no of elements/size of an array
arrayname.length will give us the size

example –
```
int a[] = new int[4];
System.out.println(a.length);
```
the output on console will be 4.

Program to print first and last elements of an array
```
public class ArrayExample
{
        public static void main(String args[])
        {
                int a[ ] = {1,2,3,4};
                display(a);
        }
        static void display(int b[])
        {
                int firstIndex = 0;
// b.length will give size 4, but last index is 3, as index starts from 0
                int lastIndex = b.length-1;
                System.out.println("First element is "+b[firstIndex]);
                System.out.println("Last Element is "+b[lastIndex]);
        }
}
```
OutPut :
```
First element is 1
Last Element is 4
```

Jagged Arrays:

Jagged array is a multidimensional array where member arrays are of different size. each row will have different no of elements in column .

Let say I want to store Student numbers of 3 sections, where each section have different number of students.

section 1 contains 64 students,
section 2 contains 65 students
section 3 contains 66 students

how to declare, first declare array with no of rows i.e., 3

```
int students[ ][ ] = new int[3][ ];
```

now declare size of each row, called member array.

```
students[0] = new int[64];
students[1] = new int[65];
students[2] = new int[66];
```

write a program that store elements if following manner
there should be 3 rows,
1st row stores elements 1,2
2nd row stores elements 3,4,5
3rd row stores elements 6,7,8,9

```
public class ArrayExample
{
        public static void main(String args[])
        {
                int rows = 3;
                int num = 1;

                int a[ ][ ] = new int[rows][ ];
                a[0] = new int[2];
                a[1] = new int[3];
                a[2] = new int[4];

                for(int i=0; i<a.length; i++)
                {
                        for(int j = 0;j<a[i].length; j++)
                        {
                                        a[i][j] = num;
                                        num++;
                        }
                }

                for(int i=0; i<a.length; i++)
                {
                        for(int j = 0;j<a[i].length; j++)
                        {
                                        System.out.print(a[i][j]+" ");
                        }
                        System.out.println();
                }

        }
}
```

OutPut:

```
1 2
3 4 5
6 7 8 9
```

Command line arguments:

Command Line Arguments is the information passed to program, at the time of execution. The passed information is stored as a string array in the main method. Command line arguments can be retrieved from String args[ ] array declared in main function.

Write a program to print given name and branch details, passed at run time using command line arguments.

```
class CommandLineArguments
{
        public static void main (String args[])
        {
                System.out.println("name is "+args[0]);
                System.out.println("branch is "+args[1]);
        }
}
```

OutPut:

```
C:\Users\UMA SHANKAR\Desktop>javac StaticExample.java

C:\Users\UMA SHANKAR\Desktop>java StaticExample Ravi ECE
name is Ravi
branch is ECE
```

Garbage Collection:

In C, C++ languages, it is the programmer's responsibility to free the memory allocated dynamically, using free() function. If a programmer knowingly or unknowingly doesn't free the memory he consumed, it effects the application performance because of inefficient memory utilization. Whereas, Java takes care of memory management. java itself is responsible to free the unused (un referenced) object data.

In the Java, memory will be allocated to objects dynamically using the **new** operator. Once an object is created, the memory remains allocated till there are references for this object.

Garbage Collector is a program in java, which is responsible for de-allocating memory occupied by un referenced objects. Garbage collector will be running periodically to de allocate memory of unused objects.

When a java object is being destroyed, Garbage Collector calls finalize() method on the object to perform clean-up activities(free the resources accessed by that object). Once finalize() method is executed, Garbage Collector will destroy that object.

when an object is said to be ready for garbage collection ?

1. object reference variable is made null
2. when object reference variable points to another object

'to be specific when there is not at least one reference to that object'

Write a program implementing finalize() method with any print statement, and make it executed from garbage collector

```
public class GarbageCollectionExample
{
        public void finalize()
        {
                System.out.println("object is garbage collected");
        }
        public static void main(String args[])
        {
                GarbageCollectionExample s1 = new GarbageCollectionExample();
                s1 = null;
                System.gc();
        }
}
```

OutPut:

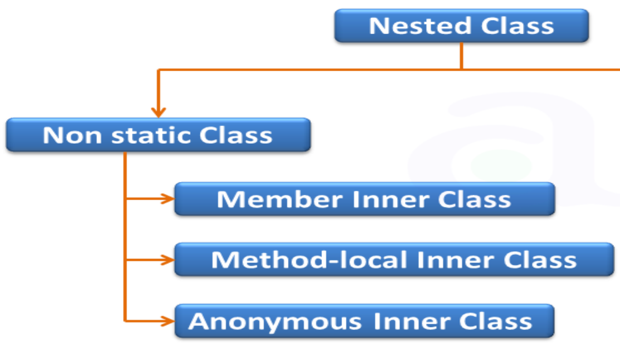
```
object is garbage collected.
```

Nested Classes

java allows us to define a class within another class, known as the **nested class**

```
class OuterClass {
   ...
   class NestedClass {
      ...
   }
}
```

Nested classes are divided into **static** and **non-static**. Nested classes that are declared as static are called ***static nested classes***. Non-static nested classes are called ***inner classes***.



```
class OuterClass {
      ...

      //static nested class
      static class StaticNestedClass {
          ...
      }

      //non-static nested class
      class InnerClass {
          ...
      }
}
```

Note:

Static nested classes do not have access to non-static members of the outer class.

As a member of the outer class, a nested class can be declared **private**, public, protected or can be left default.

Static Nested class :

An Nested class can be declared `static`, which means that you can access it without creating an object of the outer class:

```
class Outer
{
        static class StaticNestedClass
        {
                void innerClassMethod()
                {
                                System.out.println("from static Nested class");
                }
        }
        public static void main(String args[])
        {
                Outer.StaticNestedClass obj2 = new Outer.StaticNestedClass();
                obj2.innerClassMethod();
        }
}
```

Note : inner classes can access attributes and methods of the outer class

Inner Class:

A non-static class created within class and outside all methods

```
class Outer
{
	class InnerClass
	{
		void innerClassMethod()
		{
				System.out.println("from Inner class");
		}
	}

	public static void main(String args[])
	{
		Outer obj1	= new Outer();
		Outer.InnerClass obj2 = obj1.new InnerClass();
		obj2.innerClassMethod();
	}
}
```

Note: To create an object for inner class, We must create object for outer class first.

OuterClass.InnerClass ref= outerClassObject.new InnerClass();

Method Local Inner Class:

An inner class can be defined with in a method, just like local variables, the scope of the inner class is within the method. A method-local inner class can be instantiated only within the method.

```
class Outer
{
	void outerClassMethod()
	{
		class Inner
		{
			void innerClassMethod()
			{
				System.out.println("from method local inner class");
			}
		}
		Inner obj2 = new Inner();
		obj2.innerClassMethod();
	}
	public static void main(String args[])
	{
		Outer obj1 = new Outer();
		obj1.outerClassMethod();
	}
}
```

Anonymous InnerClass

Anonymous classes are declared at the time of instantiation/object creation. An inner class is declared without a class name that’s why known to be anonymous inner class. Anonymous classes are mostly used to override methods of interfaces.

```
class Outer
{
	void display()
	{
		System.out.println("from outer class method");
	}
	public static void main(String args[])
	{
		Outer obj1 = new Outer();
		obj1.display();

		Outer obj2 = new Outer()
		{
				void display()
				{
					System.out.println("from anonymous inner class");
				}
		};
		obj2.display();
	}
}
```

<u>Inheritance</u>: Access Control, Introduction to Inheritance, Types of Inheritance, Using super, Method Overriding and Dynamic Method Dispatch, Using final, Abstract Classes.

<u>Interfaces</u>: Defining and Implementing Interfaces.

<u>Packages</u>: Creating Packages, Importing Packages, Importance of CLASSPATH.

Exception Handling, Importance of try, catch, throw, throws and finally Block.

Inheritance:

Inheritance is the process of acquiring properties (data members) and functionalities(methods) of one class to another class. It is an important principle of object-oriented programming. Inheritance helps for code reusability.

'extends is the keyword in java to inherit one class functionality to another'

class B extends **A**

here all the properties and functions under class A will be inherited to class B. Along with the inherited, class B can have its own additional properties and functions.

here, class A is called parent class and class B is called child class.

Parent Class:
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

Child Class:
The class that extends the features of parent class is known as child class, sub class or derived class.

Define a class called calculator, given a two integer values it should perform

i) addition
ii) subtraction
iii) multiplication.

```
import java.util.Scanner;
class Caliculator
{
        int a, b;
        public void add(int i, int j)
        {
                int sum = i+j;
                System.out.println("addition is "+ sum);
        }
        public void substraction(int i, int j)
        {
                int difference = i-j;
                System.out.println("substraction is "+ difference);
        }
        public void multiply(int i, int j)
        {
                int multiple = i*j;
```

```
                    System.out.println("mulitplication is "+ multiple);
          }
          public static void main(String args[])
          {
                    Calculator obj = new Calculator();
                    Scanner sc = new Scanner(System.in);
                    System.out.print("Enter value for a: ");
                    obj.a =sc.nextInt();
                    System.out.print("Enter value for b: ");
                    obj.b =sc.nextInt();

                    // perform caliculator operations in these two values
                    obj.add(obj.a, obj.b);
                    obj.substraction(obj.a, obj.b);
                    obj.multiply(obj.a, obj.b);
          }
}
```

OuPut:

```
C:\Users\UMA SHANKAR\Desktop>javac Caliculator.java

C:\Users\UMA SHANKAR\Desktop>java Caliculator
Enter value for a: 8
Enter value for b: 6
addition is 14
substraction is 2
mulitplication is 48
```

Define a class called ScientificCalculator, given a two integer values it should perform

i) addition
ii) subtraction
iii) multiplication.
iv) power of
v) modulus
vi) division

we have already written first three operations in class Calculator why to write that code again in class ScientificCaliculator ? can't we reuse that code ?

'Yes' extend class Calculator in class ScientificCalculator by which it acquires instance variables a,b and three functionalities add, subtraction and multiply.

```
import java.util.*;
class ScientificCaliculator extends Caliculator
{
          public static void main(String args[])
          {
                    ScientificCaliculator ob = new ScientificCaliculator();
                    Scanner sc = new Scanner(System.in);
                    System.out.print("Enter value for a: ");
```

```
                ob.a = sc.nextInt();
                System.out.print("Enter value for a: ");
                ob.b = sc.nextInt();
        // perform all operations
                ob.add(ob.a, ob.b);
                ob.substraction(ob.a, ob.b);
                ob.multiply(ob.a, ob.b);
                ob.power(ob.a, ob.b);
                ob.modulus(ob.a, ob.b);
                ob.division(ob.a, ob.b);
        }
        public void power(int i, int j)
        {
                double sqr = Math.pow(i,j);
                System.out.println("power is "+ sqr);
        }
        public void modulus(int i, int j)
        {
                int mod = i%j;
                System.out.println("modulus is "+ mod);
        }
        public void division(int i, int j)
        {
                double div = i/j;
                System.out.println("division is "+div);
        }
}
```

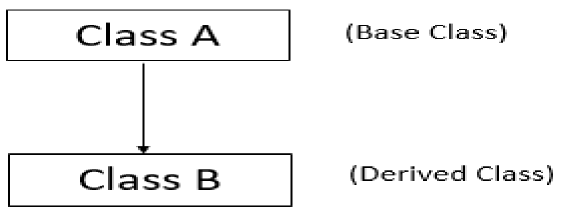
OutPut:

```
C:\Users\UMA SHANKAR\Desktop>java ScientificCaliculator
Enter value for a: 3
Enter value for a: 2
addition is 5
substraction is 1
mulitplication is 6
power is 9.0
modulus is 1
division is 1.0
```

Types of Inheritance:

Single Inheritance:

In Single Inheritance, there will be only two classes, and one class extends another class.



Example

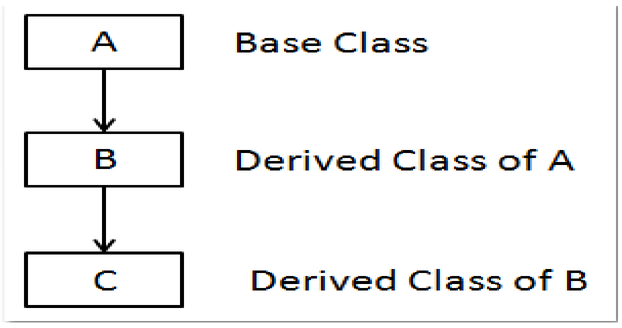
```
Class A
{
  public void methodA()
  {
   System.out.println("methodA of class A");
  }
}

Class B extends A
{
  public void methodB()
  {
   System.out.println("methodB of class B");
  }
  public static void main(String args[])
  {
   B obj = new B();
   obj.methodA();        //calling super class method
   obj.methodB();        //calling local method
 }
}
```

OutPut

```
methodA of class A
methodB of class B
```

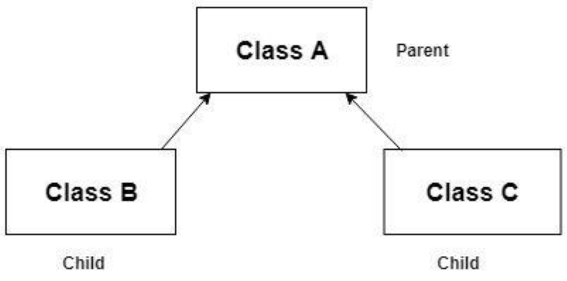
Multilevel inheritance:

In Multilevel Inheritance, a class inherits from a derived class. Hence, the derived class becomes the base class for the new class. refer the diagram.



Hierarchical Inheritance:

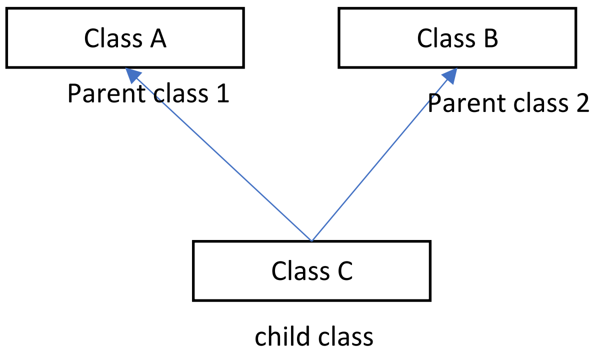
In Hierarchical Inheritance, one class is inherited by many sub classes.



*** Java doesn’t support following two types of inheritances**

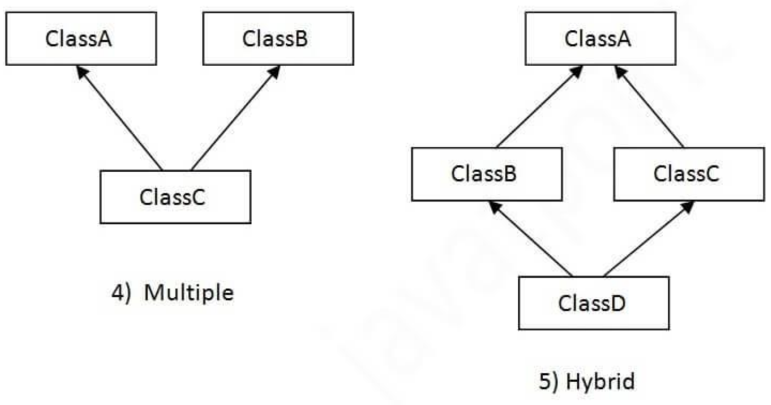
Multiple Inheritance:

In Multiple Inheritance, a class will extend more than one class. Java does not support multiple inheritance.



Hybrid Inheritance:

Hybrid inheritance is a combination of Hierarchical and Multiple inheritance.



To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call which method either of class A or B.

Java raises compile-time error if you try to inherit 2 classes. So, whether you have same method or different, there will be compile time error.

Method Overriding:

If subclass (child class) has the same method as declared in the parent class, then the function is said to be overridden in child class. When we create on object and try to invoke the method with same name, method from child class will be invoked. It is also called as dynamic polymorphism.

write a program to explain how method overriding takes place between parent and child class

```
class Parent
{
	void display()
	{
		System.out.println("method display of class Parent");
	}
}

public class Child extends Parent
{
	void display()
	{
		System.out.println("method display of class Child");
	}
	public static void main(String args[])
	{
		Child ch = new Child();
		ch.display(); // calls child display method
	}
}
```

OUTPUT:

```
method display of class Child
```

Note: When we want to access parent class overridden functions we can do, using super.

super Keyword:

The **super** keyword in java is a reference variable that is used to refer parent class objects. when a derived class and base class have members (either instance variables or functions) with same name. JVM will always pick child class member to execute. In such a scenario If we want to access parent class member's we can use super reference in child class to refer parent class members.

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1. Write program to explain accessing parent class instance overridden(variable with same name in parent and child class) variable from child class.

```
class Parent
{
//parent class instance variable
	int var=54;
}

public class Child extends Parent
{
//child class instance variable
	int var=74;
	public static void main(String[] args)
	{
		Child obj = new Child();
		obj.display();
	}
	void display()
	{
		System.out.println(var); //refers child class var
		System.out.println(super.var); // refers parent class var
	}

}
```

OUTPUT:

```
74
54
```

2. Write program to explain accessing parent class overridden method (method with same name in parent and child class) from child class.

```
class Parent
{
//parent class print method
	void print()
	{
		System.out.println("print method of parent class");
	}
}

public class Child extends Parent
{

	public static void main(String[] args)
	{
		Child obj = new Child();
		obj.display();
	}
	void display()
	{
		print(); // executes child class print method
		super.print(); // executed parent class print method
	}
//child class print method
	void print()
	{
		System.out.println("print method of child class");
	}

}
```

OUTPUT:

```
print method of child class
print method of parent class
```

3. Write a program to invoke parent class no argument constructor from child class using super

```
class Parent
{
//parent class constructor
	Parent()
	{
		System.out.println("no argument constructor of parent 
		class");
	}
}

public class Child extends Parent
{
//child class constructor
	Child()
	{
		super(); // calls parent class no argument constructor
			System.out.println("no argument constructor of child 
			class");
	}
	public static void main(String[] args)
	{
		Child obj = new Child(); // calls child class constructor
	}
}
```

OUTPUT:

```
no argument constructor of parent class
no argument constructor of child class
```

4. Write a program to invoke parent class constructor with arguments from child class using super

```
class Parent
{
//parent class constructor
	int a,b;
	Parent()
	{
		System.out.println("parent class constructor with out
					arguments");
	}
	Parent(int i, int j)
	{
		a = i;
		b = j;
		System.out.println("parent class instance variables are
					initialized");
		System.out.println("a = "+a+" b = "+b);
	}
}

public class Child extends Parent
{
//child class constructor
	Child()
	{
		super(5,4); //calls parent class constructor with arguments
		System.out.println("If we use super it should be first
				statement in constructor");
	}
	public static void main(String[] args)
	{
		Child obj = new Child(); // calls child class constructor
	}
}
```

OUTPUT:

```
parent class instance variables are initialized
a = 5 b = 4
If we use super it should be first statement in constructor.
```

Final keyword:

The **final keyword** in java is used to restrict the access to parent class variables and methods, and even to prevent the class from being inherited.

A Final can be applied for variables, methods, and class.

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Write a program using final variable, after initializing it with a value. try to update the variable again. mention the compile time error that occurs while modifying final variable.

```
class Parent
{
	final int i =65;
	void display()
	{
		i = 60;
		System.out.println(i);
		System.out.println("method display of class Parent");
	}
	public static void main(String args[])
	{
		Parent ob = new Parent();
		ob.display();
	}

}
```

Error :
java:6: error: cannot assign a value to final variable i
 i = 60;
Write a program, with method declared final and try to override that in child class. mention compile time error while you do that.

Note : we cannot override a final method in child class

```
class Parent
{
	final void display()
	{
		System.out.println("This method cannot be overriden in
                                child class");
	}
}

public class Child extends Parent
{
	int i = 80;
	void display()
	{
		System.out.println("this will not executed, as we cannot
		                    override final method");
	}
	public static void main(String args[])
	{
		Child ch = new Child();
		ch.display(); // calls child display method
	}
}
```

Error:

```
Child.java:13: error: display() in Child cannot override display() in Parent
        void display()
             ^
  overridden method is final
1 error
```

Write a program to extend a class that is final, mention the error while you obtain while implementing that

Note: We cannot extend a final class.

```
final class Parent
{
	final void display()
	{
		System.out.println("This method cannot be overriden in
		                    child class");
	}
}

public class Child extends Parent
{
	public static void main(String args[])
	{
		Child ch = new Child();
	}

}
```

```
Child.java:9: error: cannot inherit from final Parent
public class Child extends Parent
                           ^
1 error
```

Abstract classes

Abstarct method: A method which is declared as abstract and does not have implementation is known as an abstract method.

abstract void printStatus(); // no functionality will be defined when declared abstract.

Abstract class:

A class which is declared abstract, is known as an abstract class. It can have both abstract and non-abstract methods.

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. object cannot be created for abstract classes.
4. It can have constructors and static methods.
5. It can have final methods which will force the subclass not to change the body of the method.

NOTE : To use the abstract class we must **extend** the abstract class, and need to define the functionality for its abstract methods and we can use the inherited non abstract methods of it.

Write a program explaining how to use abstract and non-abstract methods of an abstract class.

```
abstract class Parent
{
      void method1()
      {
            System.out.println("this is a non abstract method class
                                    Parent");
      }
// abstract methods will not have any functionality
      abstract int method2(int a, int b);
}
class Child extends Parent
{
      public static void main(String args[])
      {
            Child ob = new Child();
            ob.method1();
            int sum = ob.method2(8,9);
            System.out.println("sum of two values is "+sum);
      }
      int method2(int i, int j)
      {
            return i+j;
      }

}
```

OUTPUT:

```
this is a non abstract method class Parent
sum of two values is 17
```

Note : If we extend an abstract class, we must implement all the abstract methods of the parent abstract class. otherwise we should declare implementing class also as abstract.

Interfaces:

An interface is a class declared by keyword interface, in interface all the methods will be abstract.

1. You cannot instantiate an interface.
2. An interface does not contain any constructors.
3. All of the methods in an interface are abstract.
4. An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
5. An interface is not extended by a class, it is implemented by a class.
6. An interface can extend multiple interfaces.

Interface declaration

```
interface interfaceName
{
        method declaration     // abstract, by default
}
```

Note:

1. An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
2. Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
3. Methods in an interface are implicitly public

Define an interface which specifies there should be functions called sum and multiply that accepts two integer numbers and returns sum and multiplication values respectively. also write a class to implement that interface.

```java
interface InterfaceOne
{
	int sum(int a, int b);
	int multiply(int i, int j);
}

class Impl implements InterfaceOne
{
	public int sum(int a1, int a2)
	{
		int sum;
		sum = a1+a2;
		return sum;
	}
	public int multiply(int b1, int b2)
	{
		int mul;
		mul = b1 * b2;
		return mul;
	}
	public static void main(String args[])
	{
		Impl obj = new Impl();
		int r1 = obj.sum(2,4);
		System.out.println("sum of values is "+r1);
		int r2 = obj.multiply(3, 4);
		System.out.println("multiplication of values is "+r2);
	}
}
```

OUTPUT:

```
sum of values is 6
multiplication of values is 12
```

NOTE:

1. A class can implement more than one interface at a time.
2. A class can extend only one class, but implement many interfaces.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods(abstract) of the parent interface.

Packages:

packages in Java are used to group related classes together. name conflicts can also be resolved using packages and helps in maintaining readability of code. Java uses file system directory for packages.

Types of packages:

1. **Built in packages**
2. **user defined packages**

Built in packages

classes which are a part of Java **API**, are called built in packages.

example: **java.lang, java.io, java.util, etc,.**

User-defined packages -These are the packages defined by the user.

creating a package:

First, we create a folder say, **myPackage** (name should be same as the name of the package we want create). Then create any class say, **MyClass** inside that directory. The first statement should be **package packagename**.

create a class A under package firstpackage, In class A write a print statement in display method as "first program on packages"

First create a directory called firstpackage (create a folder and name it as package name-firstpackage). The first statement in any java file under this package should be

package firstpackage;

```
package firstpackage;
class A
{
	public static void main(String args[])
	{
		display();
	}
	public void display()
	{
		System.out.println("first program on packages");
	}
}

compile
E:\MyApp\src>javac firstpackage/A.java

Note: compile the program from root folder
```

write a program with class B, in package secondpackage. import class A into B and execute display method of A.

```
package secondpackage; //class B belongs to second package

import firstpackage.A;  // importing class A from firstpackage
class B
{
	public static void main(String args[])
	{
		A obj = new A();
		obj.display();
	}

}
```

```
E:\MyApp\src>javac secondpackage/B.java
```

There are three ways to access the package from outside the package.

1. import package.*;

   example: import firstpackage.* (all classes inside this package will be imported)
2. import package.classname;

   import firstpackage.A;
3. fully qualified name.

   wherever we access class A, access it through firstpackage.A;

subpackages:

we can create packages inside a package known as sub packages.

```
package firstpackage.p1;
public class inner {

	public void display()
	{
		System.out.println("from method display inner class");
	}
}
```

```
E:\MyApp\src>javac firstpackage/p1/inner.java
```

here the first statement `package firstpackage.p1;`

represents class ‘inner’ belongs to package p1, where p1 is a sub package of firstpackage.

If we want to access this class inner in any other package, we need to import from root package. i.e., import package2.p1.inner observe the following program

```
package package3; //MyClass belongs to package3

import package2.p1.inner; //importing class inner

public class MyClass {

	public void display1()
	{
		System.out.println("display of MyClass in package3");
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		inner obj = new inner();
		obj.display();
	}

}
```

```
E:\MyApp\src>javac package3/MyClass.java
```

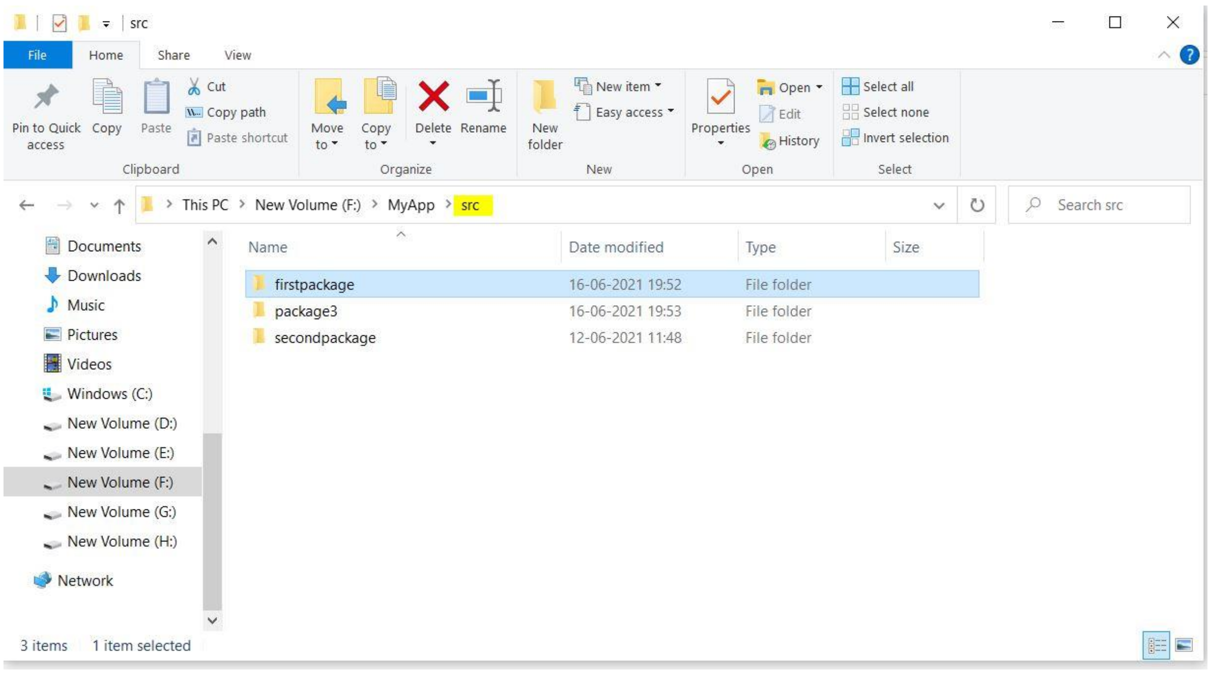
```
Executing MyClass

E:\MyApp\src>java package3.MyClass
```

OUTPUT:

```
from method display inner class
```

Here package package3; statement represents MyClass belongs to package3

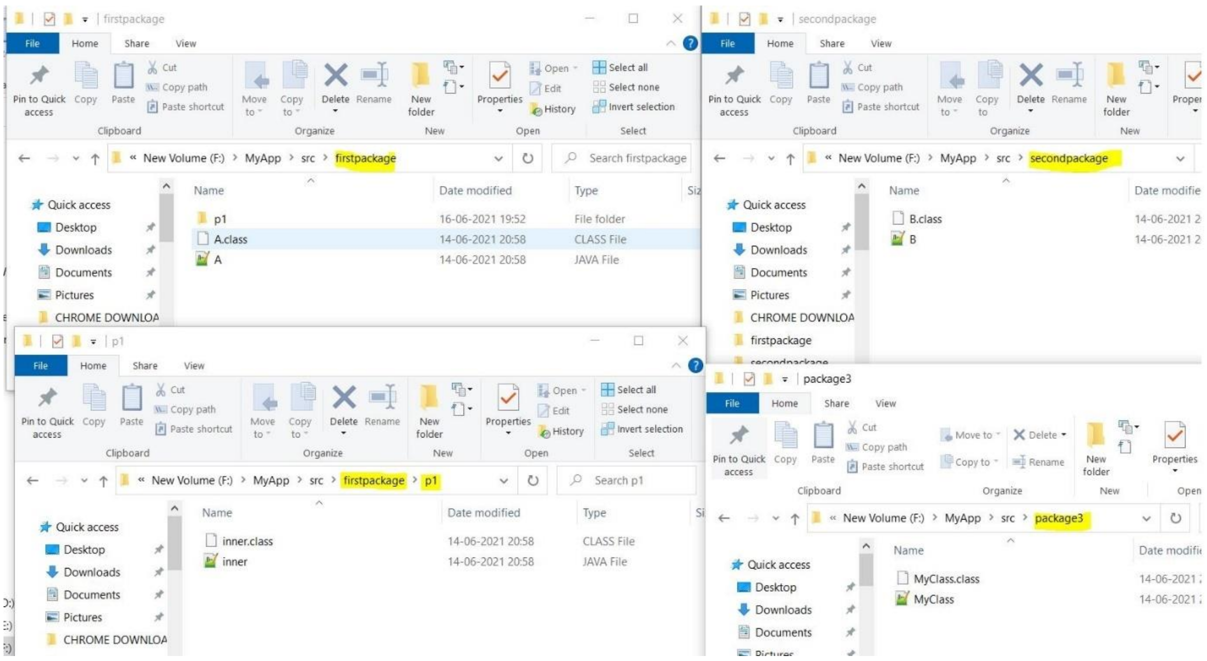
second statement import package2.p1.inner represents we import the class inner.



This PC › New Volume (F:) › MyApp › src

Name	Date modified	Type	Size
firstpackage	16-06-2021 19:52	File folder	
package3	16-06-2021 19:53	File folder	
secondpackage	12-06-2021 11:48	File folder	

3 items 1 item selected



« New Volume (F:) › MyApp › src › firstpackage

Name	Date modified	Type
p1	16-06-2021 19:52	File folder
A.class	14-06-2021 20:58	CLASS File
A	14-06-2021 20:58	JAVA File

« New Volume (F:) › MyApp › src › secondpackage

Name	Date modified
B.class	14-06-2021 2
B	14-06-2021 2

« New Volume (F:) › MyApp › src › firstpackage › p1

Name	Date modified	Type
inner.class	14-06-2021 20:58	CLASS File
inner	14-06-2021 20:58	JAVA File

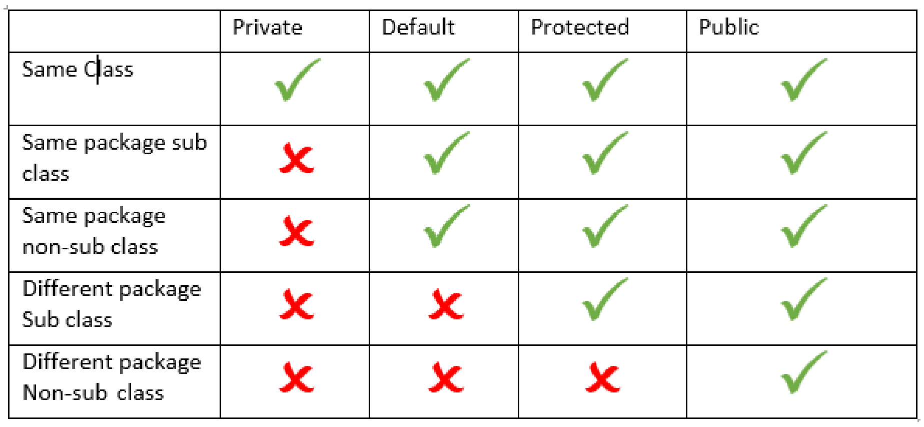
« New Volume (F:) › MyApp › src › package3

Name	Date modified
MyClass.class	14-06-2021
MyClass	14-06-2021

Access Control / Access modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.



	Private	Default	Protected	Public
Same Class	✓	✓	✓	✓
Same package sub class	✗	✓	✓	✓
Same package non-sub class	✗	✓	✓	✓
Different package Sub class	✗	✗	✓	✓
Different package Non-sub class	✗	✗	✗	✓

CLASSPATH:

CLASSPATH is an environment variable in Java, and tells Java applications and the Java Virtual Machine (JVM) where to find the classes that we use in the program.
setting class path from command prompt

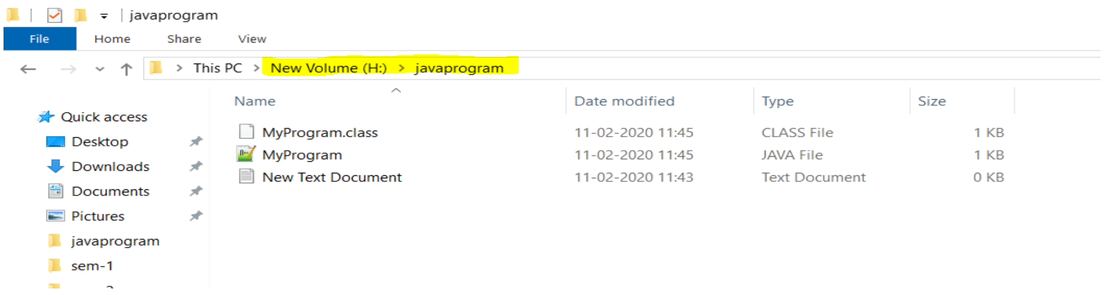
Example:
SET CLASSPATH=%CLASSPATH%;H:\javaprogram

Write a program to explain how classes existing in classpath location can be used directly without importing.

```
class MyProgram
{
	public void display()
	{
		System.out.println("from display method of class
		                              myprogram");
	}
}
```

This java file exists in the following path

H:\javaprogram



Name	Date modified	Type	Size
MyProgram.class	11-02-2020 11:45	CLASS File	1 KB
MyProgram	11-02-2020 11:45	JAVA File	1 KB
New Text Document	11-02-2020 11:43	Text Document	0 KB

If we access this class MyProgram from another directory, generally we need to import. but if the path of class MyProgram is given in classpath variable JVM can load the class by checking locations specified in classpath and can able to load and execute.

```java
public class ClassPathExample {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		MyProgram obj = new MyProgram();
		obj.display();
	}

}
```

the above ClassPathExample, uses MyProgram class and calling it's method display without importing. If classpath is not set with the location of MyProgram, JVM will not be able to fetch and load this MyProgram class and results in following error.

```
F:\example>javac ClassPathExample.java
ClassPathExample.java:6: error: cannot find symbol
                MyProgram obj = new MyProgram();
                ^
  symbol:   class MyProgram
  location: class ClassPathExample
```

after setting classpath using command prompt, as

```
F:\example>SET CLASSPATH=%CLASSPATH%;H:\javaprogram
```

(or)

setting classpath variable, with values as location of MyProgram class

JVM will not raise any error, now JVM can load the MyProgram class file by checking out the location specified in classpath variable

F:\example>javac ClassPathExample.java

F:\example>java ClassPathExample

OUTPUT:
```
from display method of class myprogram
```

Exceptions

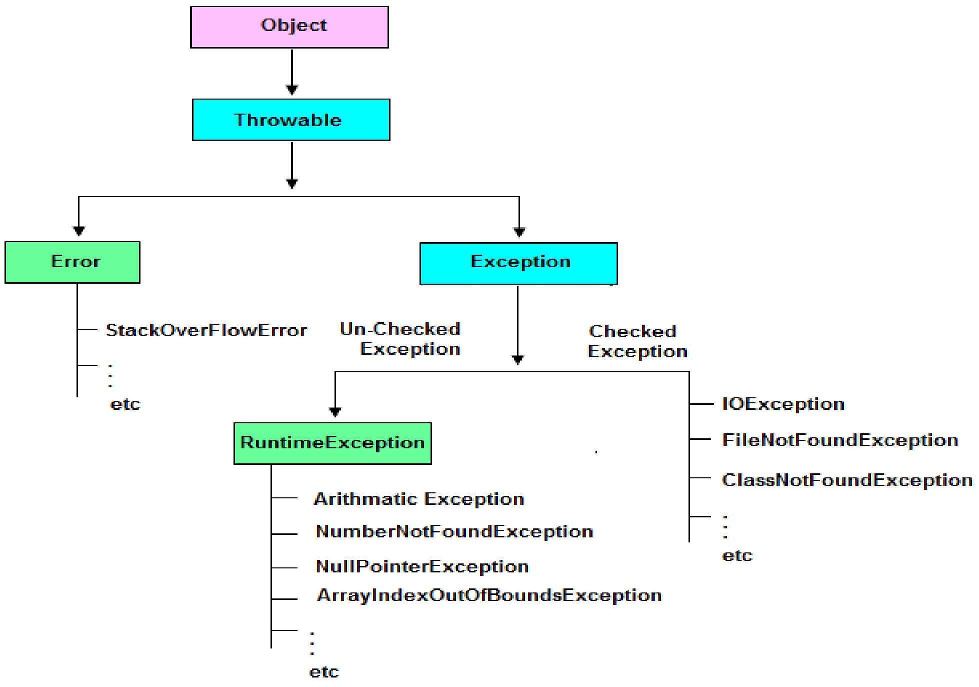
Exception in Java is an event that interrupts the execution of program-instructions and terminates the execution abnormally.

why we need exception handling

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has large no of statements and an exception occurs in middle after executing certain statements then the statements after the exception will not be executed and the program will terminate abruptly. By handling we make sure that all the statements execute and the flow of program doesn't break.

Exceptions are mainly categorized into

- Checked Exceptions
- unchecked Exceptions



Checked exceptions/Compile time exceptions – A checked exception is an compile time exception that will be checked by the compiler. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

Example - FileNotFoundException

Unchecked exceptions/Runtime Exceptions – An unchecked exception is an exception that occurs at the time of execution. Runtime exceptions will not be checked by the compiler. It is the programmer responsibility to handle these exceptions.

Example – ArithmeticException, ArrayIndexOutOfBoundsException

Note : Errors doesn’t come under the category of Exceptions

Errors can be categorized to

1. Compile time errors – Any syntax or semantic error in a program
   example –
   ```
   flt a; -> instead of float a;
   a = b+c , missing semicolon
   ```

2. Runtime errors

Erros / Runtime errors – Errors are the conditions from which application cannot get recovered by any handling techniques. It will cause termination of the program abnormally. Errors occur at runtime.
example:

```
StackOverflowError
UnSupportedClassVersionError
VirtualMachineError
```

Java provides exception handling with five keywords:
try, catch, throw, throws, and finally.

try block
Program statements that can raise exceptions should be written within a try block. If an exception occurs within the try block, it is thrown to corresponding catch block.

catch block
We can catch the exception raised in try and handle the flow of control with out letting it to terminate abruptly because of exception. We can provide any useful information to user regarding the exception.

A try block can be followed by multiple catch blocks.

throw
We can throw the exceptions manually either predefined or user defined, by using the keyword throw.

throws
When an exception causing method wants to return the Exception to its calling method it can be thrown by a throws clause and calling method should handle this exception.

finally
The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception. Using a finally block allows you to run any clean up statements that you want to execute like closing files, session or network connections

Example :
consider the following statement which leads to – ArithmeticException -divide by zero Exception

```
int res = a/b;          // There is chance of getting Exception
```

Handling the exception

```
try {
        int res = a/b ;
}
catch(ArithmeticException e)
{
        System.out.println("Exception caught : "+e);
}
finally
{
        System.out.println("statements under finally will always be executed even when
        exception is not raised");
}
```

user defined Exceptions/ custom exceptions

User Defined Exception or custom exception is creating your own exception class and raising that exception using 'throw' keyword. custom exception class can be created by extending the class Exception.

In the below example problem when amount required to with drawl exceeds the balance we are going to raise a user define exception called OutOfBalanceException

```
class Main
{
      public static void main(String args[])
      {
            Operations obj = new Operations();
            try {
            obj.withdrawl(20000);
            }
            catch(OutOfBalanceException e)
            {
                  System.out.println(e);
            }
      }
}
class OutOfBalanceException extends Exception
{
      public String toString()
      {
            return "insufficient funds can not perform withdrawl";
      }
}
class Operations
{
    static int balance = 10000;
    void withdrawl(int req)throws OutOfBalanceException
    {
        if(balance < req )
        {
            throw new OutOfBalanceException();
        }
        else
        {
            balance = balance - req;
        }
    }
}
```

To raise the user defined exception we an can use throw keyword

```
throw new OutOfBalanceException();
```

here the raised exception is transferred to the caller method using throws keyword, the exception should be handled there

```
try {
    obj.withdrawl(20000);
}


void withdrawl(int req)throws OutOfBalanceException
{
    throw new OutOfBalanceException();
}
```

Using parent Exception class reference for child class Exception

In this example, we generate ArithmeticException, but the catch block with corresponding exception type is not defined. In such case, the catch block containing the parent exception class **Exception** will executed.

```
public class CatchParentExceptionClass {

    public static void main(String[] args) {
            try{
                 int res = 24/0;
                System.out.println(res);
               }
               catch(NullPointerException e)
                  {
                   System.out.println("Arithmetic Exception
                                        occurs");
                  }
               catch(ArrayIndexOutOfBoundsException e)
                  {
                         System.out.println("ArrayIndexOutOfBounds
                                              Exception occurs");
                  }
               catch(Exception e)
                  {
                   System.out.println("Parent Exception class
                                        matched");
                  }
               System.out.println("rest of the code");
      }
  }

output : Parent Exception class matched
```

UNIT-IV

Multithreading and Files: Introduction, Thread Lifecycle, Creation of Threads, Thread Priorities, Thread Synchronization, Communication between Threads. Reading Data from Files and Writing Data to Files, Random Access Files

Multithreading is a process of executing multiple threads simultaneously.

A Thread is a light weight process, a smallest unit of processing. Multithreading helps to increase the throughput. In multithreading, the threads will share a common memory area so the context switching between the threads will take less time

Advantages of Java Multithreading

1) The users are not blocked because threads are independent, and we can perform multiple operations at same time

2) As such the threads are independent, the other threads won't get affected if one thread meets an exception.

Threads can be created in two ways:

1. Extending the Thread class
2. Implementing the Runnable Interface

Life cycle of a Thread (Thread States)

The life cycle of thread consists of 5 different states.

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

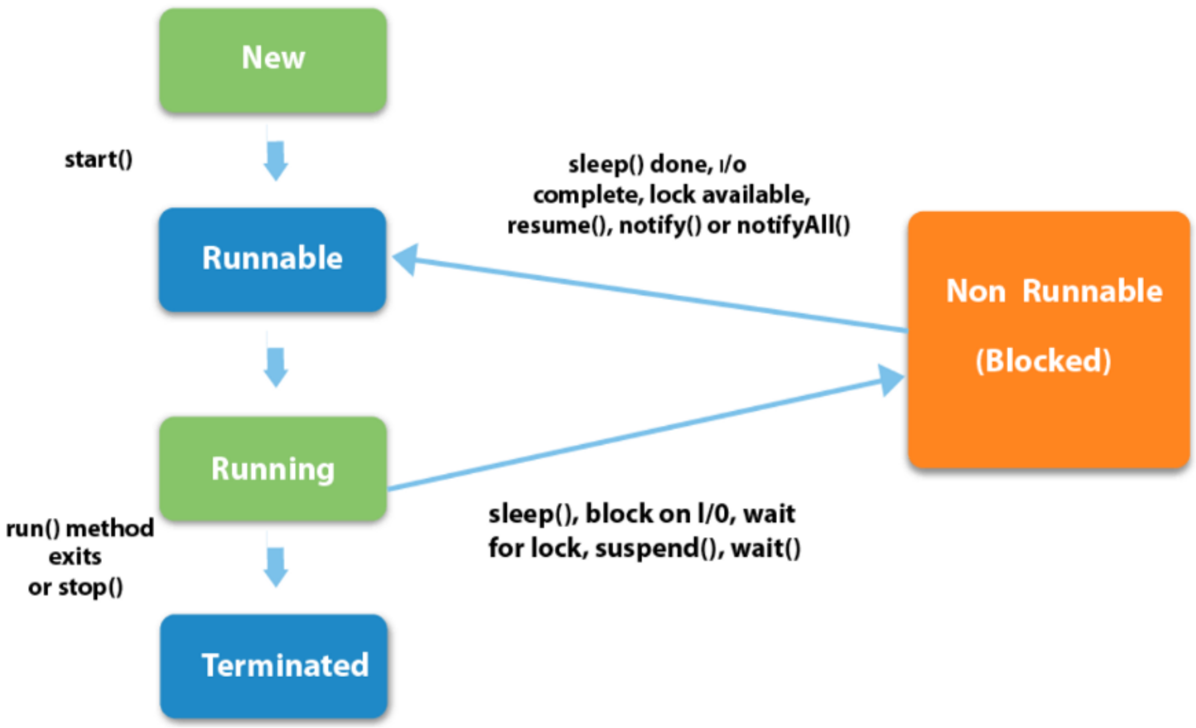
1. **New:** A thread is said to be in new state, if you create an instance of Thread class and start() method is *not yet invoked*.

2. **Runnable:** In this stage, the instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, when to run the thread.

3. **Running:** When the thread starts executing, then the state is changed to "running" state. The scheduler selects one thread from the thread pool, and it starts executing.

4. **Blocked/Non Runnable :** thread is considered to be in the blocked state when it is suspended, sleeping, or waiting for some time in order to satisfy some condition.

5. **Terminated:** This is the state when the thread is terminated. When the thread completes execution of its "run" method, it goes into the "terminated" state.


New
start()
Runnable
sleep() done, i/o complete, lock available, resume(), notify() or notifyAll()
Non Runnable (Blocked)
Running
sleep(), block on I/0, wait for lock, suspend(), wait()
run() method exits or stop()
Terminated

Creating Threads

1. Creating Thread by extending the Thread class

We should create a class by extending the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Example :

Java Thread Example by extending Thread class

```
class ThreadExample extends Thread
{
      public void run()
      {
                System.out.println("thread is running...");
      }
      public static void main(String args[])
      {
               ThreadExample t1=new ThreadExample ();
               t1.start();
      }
}
```

Constructors of Thread class:

1. Thread()
2. Thread(String name)
3. Thread(Runnable r)
4. Thread(Runnable r, String name)

methods of Thread class:

1. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
2. **public void run():** is used to perform action for a 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 milliseconds):** waits for a thread to die for the specified milliseconds.
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 boolean isDaemon():** tests if the thread is a daemon thread.
15. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
16. **public void interrupt():** interrupts the thread.
17. **public boolean isInterrupted():** tests if the thread has been interrupted.
18. **public void suspend():** is used to suspend the thread(depricated).
19. **public void resume():** is used to resume the suspended thread(depricated).
20. **public void stop():** is used to stop the thread(depricated).
21. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.

2. Creating Thread by implementing Runnable interface:

Threads can also be created by implementing the Runnable interface. Runnable interface have only one method named run(), with the help of Runnable interface the class can work as thread and also we can extend any other class

public void run(): is used to perform action for a thread.

following are the steps to create a new thread using the Runnable interface:

1. The first step is to create a Java class that implements the Runnable interface.
2. The second step is to override the run() method of the Runnable() interface in the class.
3. Now, pass the Runnable object as a parameter to the constructor of the Thread class
4. Finally, invoke the start method of the Thread object.

Example:

```
class RunnableThread implements Runnable
{
		public void run()
		{
				System.out.println("thread created using Runnable Interface");
		}
		public static void main(String args[])
		{
				RunnableThread m1=new RunnableThread ();
				RunnableThread t1 =new RunnableThread (m1);
				t1.start();
		}
}
```

Priority of a Thread (Thread Priority):

Each thread will have a priority. Priorities are represented by a number between 1, 5 and 10. In most cases, thread schedular 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.

constants defined in Thread class for Priorities:

1.	public static int MIN_PRIORITY	- 1
2.	public static int NORM_PRIORITY	- 5
3.	public static int MAX_PRIORITY	- 10

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example for priority of a Thread:

```
class ExamplePriority 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[])
    {
        ExamplePriority m1=new ExamplePriority ();
        ExamplePriority m2=new ExamplePriority ();
        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
```

Thread Scheduler in Java:

Thread scheduler in java is the part of the JVM that decides which thread should run. There is no guarantee that which runnable thread will be chosen to run by the thread scheduler. Only one thread at a time can run in a single process. The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Java Synchronization

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent inconsistency problem.

Thread Synchronization

thread synchronization can be done with the help of

Synchronized method.

Synchronized block.

Static synchronization.

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Understanding the problem without Synchronization

```
class Table{
void printTable(int n){//method not synchronized
 for(int i=1;i<=5;i++){
 System.out.println(n*i);
 try{
 Thread.sleep(400);
 }catch(Exception e){System.out.println(e);}
 }
}
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
} }
class TestSynchronization1{
public static void main(String args[]){
```

```
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Table{
synchronized void printTable(int n){//synchronized method
 for(int i=1;i<=5;i++){
 System.out.println(n*i);
 try{
 Thread.sleep(400);
 }catch(Exception e){System.out.println(e);}
 }
}
}
```

Synchronized Block in Java

Synchronized block can be used to perform synchronization on any specific resource of the method. Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block. If you put all the codes of the method in the synchronized block, it will work same as the synchronized method. Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

```
synchronized (object reference expression) {
//code block
}
```

```
class Table{
        void printTable(int n){
        synchronized(this){//synchronized block
        for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
        Thread.sleep(400);
        }catch(Exception e){System.out.println(e);}
        }
        }
        }//end of the method
}
```

Static Synchronization

If you make any static method as synchronized, the lock will be on the class not on object

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

```
class Table{
synchronized static void printTable(int n){
for(int i=1;i<=10;i++){
System.out.println(n*i);
 try{
 Thread.sleep(400);
 }catch(Exception e){}
 }
}
}
class MyThread1 extends Thread{
public void run(){
Table.printTable(1);
}
}

class MyThread2 extends Thread{
public void run(){
Table.printTable(10);
}
}

class MyThread3 extends Thread{
public void run(){
Table.printTable(100);
}
}
class MyThread4 extends Thread{
public void run(){
Table.printTable(1000);
}
}

public class TestSynchronization4{
```

```
public static void main(String t[]){
MyThread1 t1=new MyThread1();
MyThread2 t2=new MyThread2();
MyThread3 t3=new MyThread3();
MyThread4 t4=new MyThread4();
t1.start();
t2.start();
t3.start();
t4.start();
}
}
```

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.

Example of Deadlock in java

```
class TestDeadlockExample1 {
 public static void main(String[] args) {
 final String resource1 = "CSection";
 final String resource2 = "DSection";
 // 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

Inter-thread communication in Java

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:

- wait()
- notify()
- notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.
Method Description
public final void wait()throws InterruptedException waits until object is notified.
public final void wait(long timeout)throws InterruptedException waits for the specified amount of time

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax: public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:public final void notifyAll()

wait(), notify() and notifyAll() methods belongs to Thread class ?

No, they belong to Object class

Inter thread communication in java

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();
} }
```

```
Output: going to withdraw...
 Less balance; waiting for deposit...
 going to deposit...
 deposit completed...
 withdraw completed
```

Java I/O

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

OutputStream vs InputStream

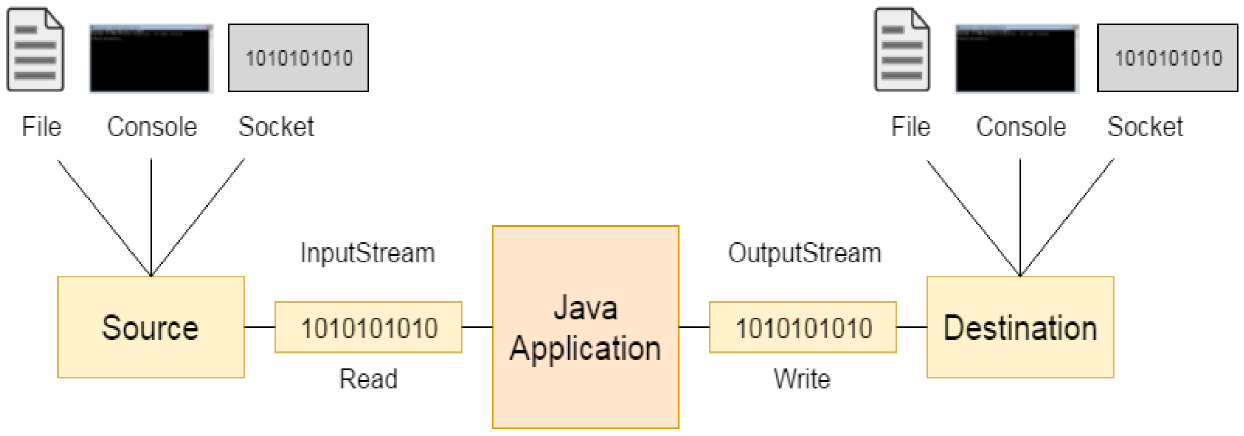
OutputStream

Java application uses an output stream to write data to a destination it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Useful methods of InputStream

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

InputStream Hierarchy

Java FileOutputStream Class

- Java FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class.
- You can write byte-oriented as well as character-oriented data through FileOutputStream class.
- But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

FileOutputStream class declaration

public class FileOutputStream extends OutputStream

FileOutputStream class methods

Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example 1: write byte

```
import java.io.FileOutputStream;

public class FileOutputStreamExample { public static
    void main(String args[]){
           try{
             FileOutputStream fout=new FileOutputStream("D:\\testout.txt"); fout.write(65);
             fout.close(); System.out.println("success...");
            }catch(Exception e){System.out.println(e);}
      }
}
```

Output:

```
Success...
```

The content of a text file testout.txt is set with the data A. A

Java FileOutputStream example 2: write string import

```
java.io.FileOutputStream;
public class FileOutputStreamExample { public static
    void main(String args[]){
           try{
             FileOutputStream fout=new FileOutputStream("D:\\testout.txt"); String s="Welcome
             to javaTpoint.";
             byte b[]=s.getBytes();//converting string into byte array fout.write(b);
             fout.close();
             System.out.println("success...");
            }catch(Exception e){System.out.println(e);}
      }
}
```

Output:

```
Success...
```

The content of a text file testout.txt is set with the data Welcome to java. testout.txt

Welcome to java.

Java FileInputStream Class

- Java FileInputStream class obtains input bytes from a file.
- It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc.
- You can also read character-stream data.
- But, for reading streams of characters, it is recommended to use FileReader class.

Java FileInputStream class declaration

public class FileInputStream extends InputStream

Java FileInputStream class methods:

Method	**Description**
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Java FileInputStream example 1: read single character

```
import java.io.FileInputStream; public class

DataStreamExample {

    public static void main(String args[]){ try{

            FileInputStream fin=new FileInputStream("D:\\testout.txt"); int i=fin.read();

            System.out.print((char)i);

            fin.close();

          }catch(Exception e){System.out.println(e);}

         }

        }
```

Note: Before running the code, a text file named as "testout.txt" is required to be created. In this file, we are having following content: Welcome to java

Welcome to java

After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character.

Output:

```
W
```

Java FileInputStream example 2: read all characters

```
import java.io.FileInputStream; public class

DataStreamExample {

    public static void main(String args[]){ try{

            FileInputStream fin=new FileInputStream("D:\\testout.txt"); int i=0;

            while((i=fin.read())!=-1){

             System.out.print((char)i);

            }

            fin.close();

          }catch(Exception e){System.out.println(e);}

         }

        }
```

Output: Welcome to java

Java File Class

- The File class is an abstract representation of file and directory pathname.
- A pathname can be either absolute or relative.
- The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Java File Example 1

```
import java.io.*;

public class FileDemo {

    public static void main(String[] args) {


        try {

            File file = new File("javaFile123.txt"); if

            (file.createNewFile()) {

                System.out.println("New File is created!");

            } else {

                System.out.println("File already exists.");

            }
        } catch (IOException e)

        {

            e.printStackTrace();

        }

    }

}
```

Output:

```
New File is created!
```

Java - RandomAccessFile

- This class is used for reading and writing to random access file.
- A random access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Method

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

Example

```
import java.io.IOException; import
java.io.RandomAccessFile;
public class RandomAccessFileExample
{
    static final String FILEPATH ="myFile.txt"; public
    static void main(String[] args)
  {  try {
            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
            writeToFile(FILEPATH, "I love my country and my people", 31);
             } catch (IOException e) { e.printStackTrace();
        }
    }
    private static byte[] readFromFile(String filePath, int position, int size) throws
            IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "r");
        file.seek(position);
        byte[] bytes = new byte[size];
        file.read(bytes);
        file.close();
        return bytes;
    }
    private static void writeToFile(String filePath, String data, int position) throws
            IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "rw");
        file.seek(position);
        file.write(data.getBytes());
        file.close();
    }
}
```

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contains

This class is used for reading I love my country and my people.

UNIT-V

S.No	Course Outcome	Intended Learning Outcomes (ILO)	Knowledge Level of ILO
1	CO 5	Explain about applet class	K2
2		Discuss about Applet Lifecycle	K2
		Discuss about AWT ,Components and Containers of AWT	K2
		Illustrate various AWT Controls like Button,label,Checkbox, RadioButton,List box, Menu and Scrollbar with example programs	K3
		Interpret different types of layout managers with examples	K3

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

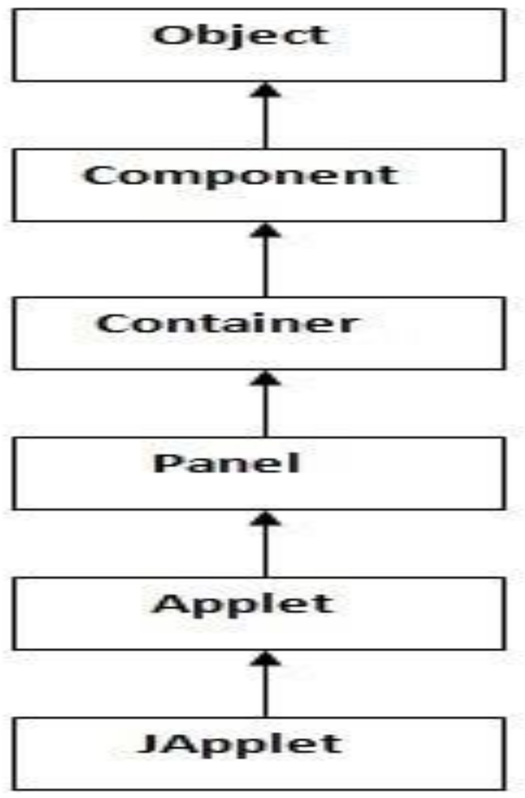
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods 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

1. public void init(): is used to initialized the Applet. It is invoked only once.
2. public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. 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.

1. 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.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
   public void paint(Graphics g)
  {
   g.drawString("welcome",150,150);
  }
}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
  public void paint(Graphics g)
    {
      g.drawString("welcome to applet",150,150);
    }
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

- **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
- **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
- **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.

- **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.

- **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{

   public void paint(Graphics g)
   {
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);

        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);

   }
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

public Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.

public URL getCodeBase(): is used to return the base URL.

Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
{
        Image picture;
        public void init()
        {
          picture = getImage(getDocumentBase(),"img1.jpg");
        }

        public void paint(Graphics g)
       {
          g.drawImage(picture, 30,30, this);
        }

  }
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{

  Image picture;

  public void init()
   {
      picture =getImage(getDocumentBase(),"bike_1.gif");
    }

  public void paint(Graphics g)
   {
     for(int i=0;i<500;i++)
     {
       g.drawImage(picture, i,30, this);

       try{ Thread.sleep(100); }  catch(Exception e){}
    }
  }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="AnimationExample.class" width="300" height="300">
</applet>
</body>
</html>
```

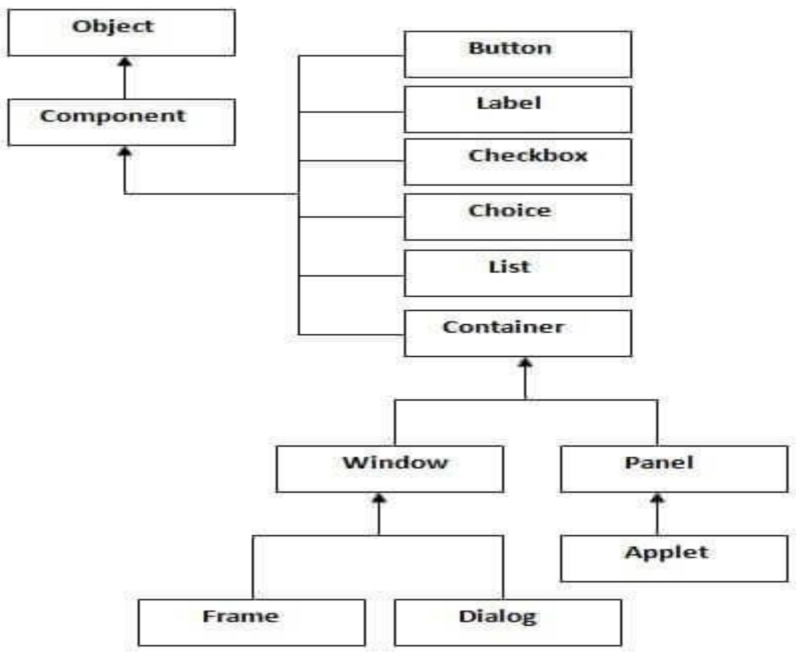
Java AWT

- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy



Container
The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window
The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel
The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame
The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class:

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example
To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame
     {
        First()
        {
             Button b=new Button("click me");
             b.setBounds(30,100,80,30);// setting button position
             add(b);//adding button into frame
             setSize(300,300);//frame size 300 width and 300 height
             setLayout(null);//no layout manager
             setVisible(true);//now frame will be visible, by default not visible
        }
        public static void main(String args[])
        {
             First f=new First();
        }
    }
```

The **setBounds(int xaxis, int yaxis, int width, int height)** method is used in the above example that sets the position of the awt button.

AWT Example by Association:

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2
{
   First2()
    {
      Frame f=new Frame();
      Button b=new Button("click me");
      b.setBounds(30,50,80,30);
      f.add(b);
      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
    }
   public static void main(String args[])
   {  First2 f=new First2();  }}
```

Java AWT Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

AWT Button Class declaration

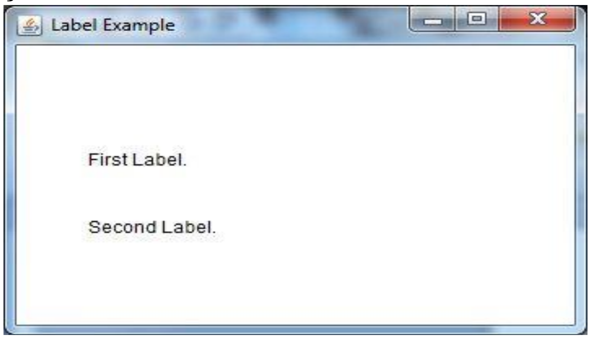
public class Button extends Component

AWT Label Class Declaration

public class Label extends Component

Java Label Example

```
import java.awt.*;
class LabelExample
{
 public static void main(String args[])
 {
    Frame f= new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
 }
}
```



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.

Components of Event Handling

Event handling has three main components,

- Events: An event is a change in state of an object.
- Events Source: An event source is an object that generates an event.
- Listeners: A listener is an object that listens to the event. A listener gets notified when an event occurs.

Important Event Classes and Interface

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exits a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener

Event Classes	Description	Listener Interface
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener

Event Classes	**Description**	**Listener Interface**
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

<u>Steps to perform Event Handling:</u>

- 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){}

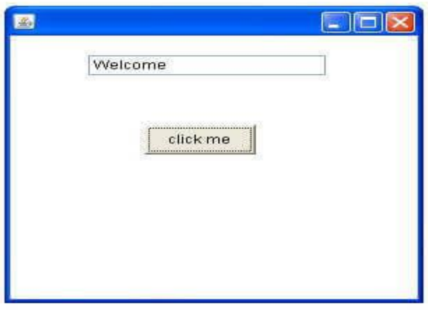
Java Event Handling Code

We can put the event handling code into one of the following places:

- Within class
- Other class
- Anonymous 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();
    }
}
```



2) Java event handling by outer class

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

3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame
{
      TextField tf;
      AEvent3()
      {
          tf=new TextField();
          tf.setBounds(60,50,170,20);
          Button b=new Button("click me");
          b.setBounds(50,120,80,30);

          b.addActionListener(new ActionListener(){
                 public void actionPerformed()
                 {
                 tf.setText("hello");
                 }
                 });
          add(b);add(tf);
          setSize(300,300);
          setLayout(null);
          setVisible(true);
      }
      public static void main(String args[])
      {
      new AEvent3();
      }
}
```

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.

Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

```
public abstract void windowActivated(WindowEvent e);
public abstract void windowClosed(WindowEvent e);
public abstract void windowClosing(WindowEvent e);
public abstract void windowDeactivated(WindowEvent e);
public abstract void windowDeiconified(WindowEvent e);
public abstract void windowIconified(WindowEvent e);
public abstract void windowOpened(WindowEvent e);
```

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");
}
}
```

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().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

```
public abstract void actionPerformed(ActionEvent e);
```

How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

```
    public class ActionListenerExample Implements ActionListener
```

2) Register the component with the Listener:

```
   component.addActionListener(instanceOfListenerclass);
```

3) Override the actionPerformed() method:

```
   public void actionPerformed(ActionEvent e)
   {
        //Write the code here
    }
```

Java MouseListener 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.

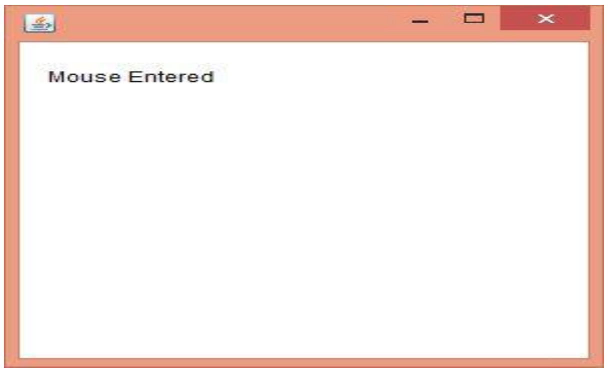
Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

```
public abstract void mouseClicked(MouseEvent e);
public abstract void mouseEntered(MouseEvent e);
public abstract void mouseExited(MouseEvent e);
public abstract void mousePressed(MouseEvent e);
public abstract void mouseReleased(MouseEvent e);
```

Java MouseListener Example1

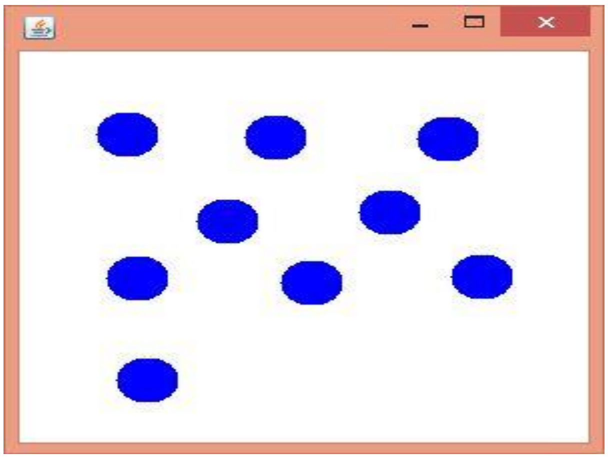
```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
   Label l;
   MouseListenerExample(){
      addMouseListener(this);

      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();
}
}
```



```
Mouse Entered
```

Java MouseListener Example 2

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample2 extends Frame implements MouseListener{
   MouseListenerExample2(){
      addMouseListener(this);

      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();
}
}
```



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.

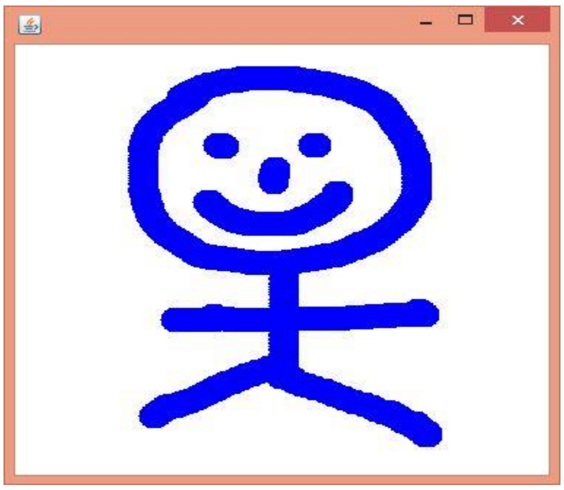
Methods of MouseMotionListener interface

The signature of 2 methods found in MouseMotionListener interface are given below:

```
public abstract void mouseDragged(MouseEvent e);
public abstract void mouseMoved(MouseEvent e);
```

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();
}
}
```



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 signature of 3 methods found in KeyListener interface are given below:

```
public abstract void keyPressed(KeyEvent e);
public abstract void keyReleased(KeyEvent e);
public abstract void keyTyped(KeyEvent e);
```

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();
    }
}
```

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

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
   Frame f;
   AdapterExample(){
      f=new Frame("Window Adapter");
      f.addWindowListener(new WindowAdapter(){
         public void windowClosing(WindowEvent e) {
            f.dispose();
         }
      });

      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
public static void main(String[] args) {
   new AdapterExample();
}  }
```

Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

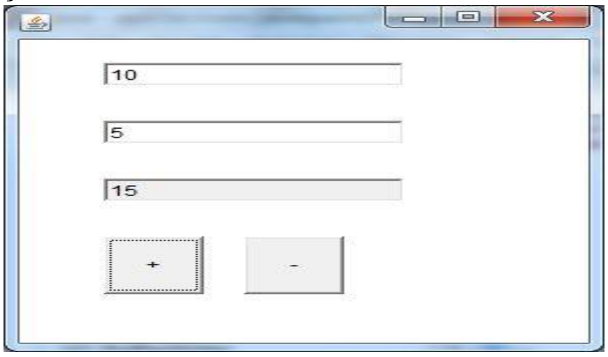
AWT TextField Class Declaration

```
public class TextField extends TextComponent
```

Java AWT TextField Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldExample extends Frame implements ActionListener
{
   TextField tf1,tf2,tf3;
   Button b1,b2;
   TextFieldExample()
   {
      tf1=new TextField();
      tf1.setBounds(50,50,150,20);
      tf2=new TextField();
      tf2.setBounds(50,100,150,20);
      tf3=new TextField();
      tf3.setBounds(50,150,150,20);
      tf3.setEditable(false);
      b1=new Button("+");
      b1.setBounds(50,200,50,50);
      b2=new Button("-");
      b2.setBounds(120,200,50,50);
      b1.addActionListener(this);
      b2.addActionListener(this);
      add(tf1);add(tf2);add(tf3);add(b1);add(b2);
      setSize(300,300);
      setLayout(null);
      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();
 }
}
```



Java AWT TextArea

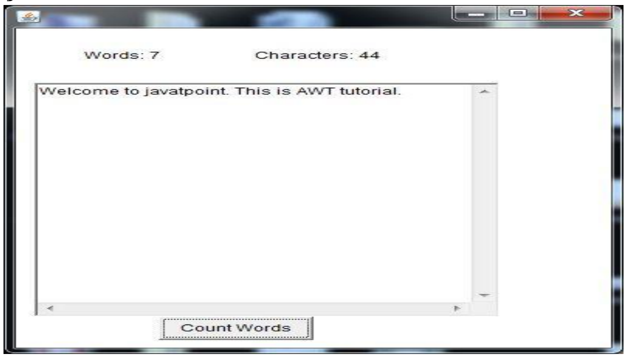
The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration

public class TextArea extends TextComponent

Java AWT TextArea Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class TextAreaExample extends Frame implements ActionListener
{
Label l1,l2;
TextArea area;
Button b;
TextAreaExample()
{
    l1=new Label();
    l1.setBounds(50,50,100,30);
    l2=new Label();
    l2.setBounds(160,50,100,30);
    area=new TextArea();
    area.setBounds(20,100,300,300);
    b=new Button("Count Words");
    b.setBounds(100,400,100,30);
    b.addActionListener(this);
    add(l1);add(l2);add(area);add(b);
    setSize(400,450);
    setLayout(null);
    setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args)
{
    new TextAreaExample();
}
}
```



```
Words: 7                Characters: 44

Welcome to javatpoint. This is AWT tutorial.
```

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".

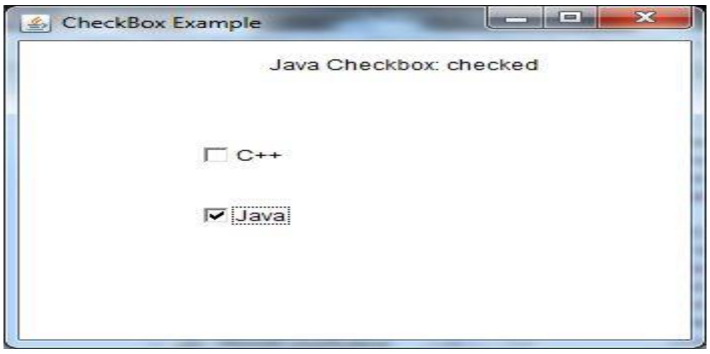
AWT Checkbox Class Declaration

public class Checkbox extends Component

Java AWT Checkbox Example with ItemListener

```
import java.awt.*;
import java.awt.event.*;
```

```
public class CheckboxExample
{
    CheckboxExample()
   {
       Frame f= new Frame("CheckBox Example");
       final Label label = new Label();
       label.setAlignment(Label.CENTER);
       label.setSize(400,100);
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100,100, 50,50);
       Checkbox checkbox2 = new Checkbox("Java");
       checkbox2.setBounds(100,150, 50,50);
       f.add(checkbox1); f.add(checkbox2); f.add(label);
       checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               label.setText("C++ Checkbox: "
               + (e.getStateChange()==1?"checked":"unchecked"));
            }
         });
       checkbox2.addItemListener(new ItemListener()
         {
            public void itemStateChanged(ItemEvent e)
            {
               label.setText("Java Checkbox: "
               + (e.getStateChange()==1?"checked":"unchecked"));
            }
         });
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxExample();
}
}
```



Java AWT CheckboxGroup

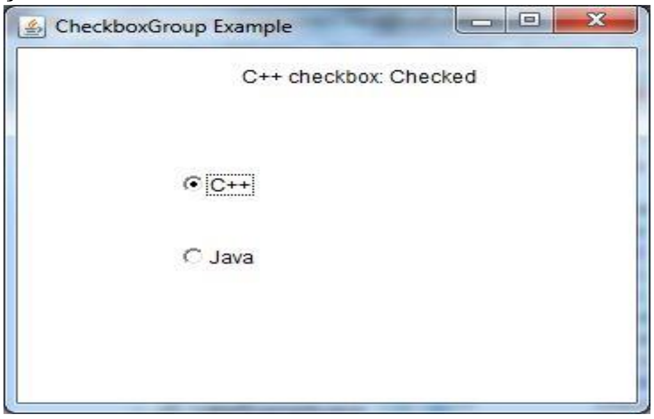
The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

AWT CheckboxGroup Class Declaration

public class CheckboxGroup extends Object

Java AWT CheckboxGroup Example with ItemListener

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxGroupExample
{
    CheckboxGroupExample(){
      Frame f= new Frame("CheckboxGroup Example");
      final Label label = new Label();
      label.setAlignment(Label.CENTER);
      label.setSize(400,100);
       CheckboxGroup cbg = new CheckboxGroup();
       Checkbox checkBox1 = new Checkbox("C++", cbg, false);
       checkBox1.setBounds(100,100, 50,50);
       Checkbox checkBox2 = new Checkbox("Java", cbg, false);
       checkBox2.setBounds(100,150, 50,50);
       f.add(checkBox1); f.add(checkBox2); f.add(label);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
       checkBox1.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
             label.setText("C++ checkbox: Checked");
          }
        });
       checkBox2.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("Java checkbox: Checked");
          }
        });
    }
public static void main(String args[])
{
   new CheckboxGroupExample();
}
}
```



C++ checkbox: Checked

Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

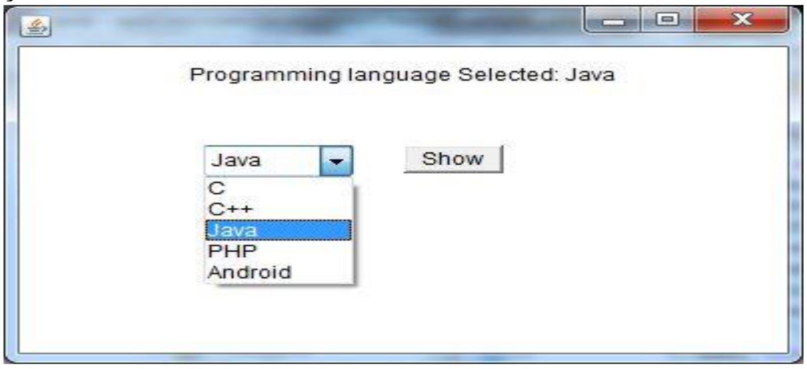
AWT Choice Class Declaration

```
public class Choice extends Component
```

Java AWT Choice Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ChoiceExample
{
      ChoiceExample(){
      Frame f= new Frame();
      final Label label = new Label();
      label.setAlignment(Label.CENTER);
      label.setSize(400,100);
      Button b=new Button("Show");
      b.setBounds(200,100,50,20);
      final Choice c=new Choice();
      c.setBounds(100,100, 75,75);
      c.add("C");
      c.add("C++");
```

```
        c.add("Java");
        c.add("PHP");
        c.add("Android");
        f.add(c);f.add(label); f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
         String data = "Programming language Selected: "+
c.getItem(c.getSelectedIndex());
         label.setText(data);
        }
        });
        }
public static void main(String args[])
{
  new ChoiceExample();
}
}
```


Programming language Selected: Java

Java AWT List

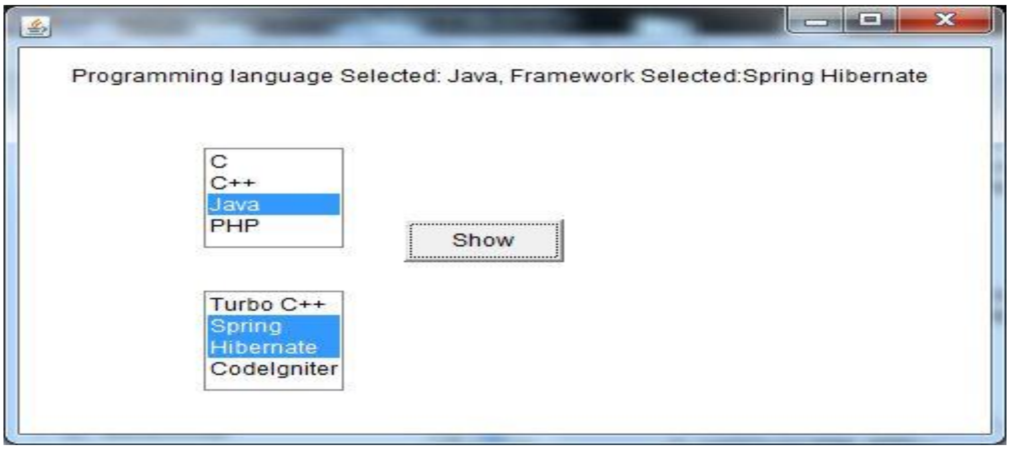
The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

AWT List class Declaration

```
public class List extends Component
```

Java AWT List Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ListExample
{
    ListExample(){
        Frame f= new Frame();
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(500,100);
        Button b=new Button("Show");
        b.setBounds(200,150,80,30);
        final List l1=new List(4, false);
        l1.setBounds(100,100, 70,70);
        l1.add("C");
        l1.add("C++");
        l1.add("Java");
        l1.add("PHP");
        final List l2=new List(4, true);
        l2.setBounds(100,200, 70,70);
        l2.add("Turbo C++");
        l2.add("Spring");
        l2.add("Hibernate");
        l2.add("CodeIgniter");
        f.add(l1); f.add(l2); f.add(label); f.add(b);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
          String data = "Programming language Selected: 
"+l1.getItem(l1.getSelectedIndex());
          data += ", Framework Selected:";
          for(String frame:l2.getSelectedItems()){
                 data += frame + " ";
          }
          label.setText(data);
          }
         });
}
public static void main(String args[])
{
   new ListExample();
}
}
```


Programming language Selected: Java, Framework Selected:Spring Hibernate

Java AWT Scrollbar

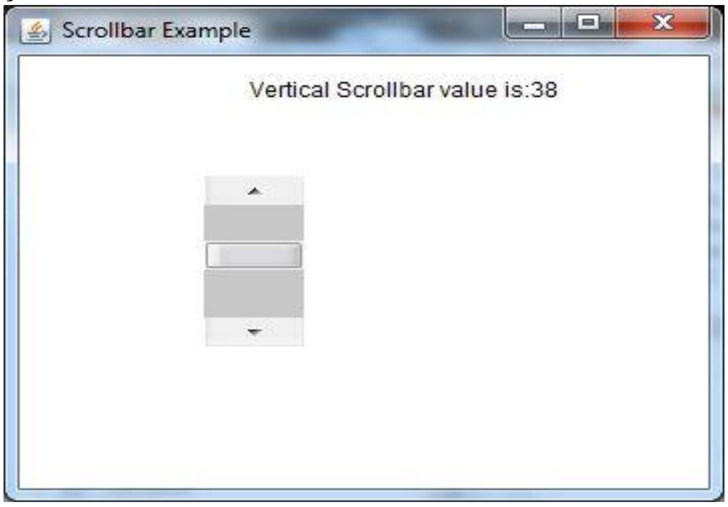
The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

AWT Scrollbar class declaration

public class Scrollbar extends Component

Java AWT Scrollbar Example with AdjustmentListener

```
import java.awt.*;
import java.awt.event.*;
class ScrollbarExample{
    ScrollbarExample(){
          Frame f= new Frame("Scrollbar Example");
          final Label label = new Label();
          label.setAlignment(Label.CENTER);
          label.setSize(400,100);
          final Scrollbar s=new Scrollbar();
          s.setBounds(100,100, 50,100);
          f.add(s);f.add(label);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
          s.addAdjustmentListener(new AdjustmentListener() {
             public void adjustmentValueChanged(AdjustmentEvent e) {
                label.setText("Vertical Scrollbar value is:"+ s.getValue());
             }
          });
        }
public static void main(String args[]){
new ScrollbarExample();
}
}
```



Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

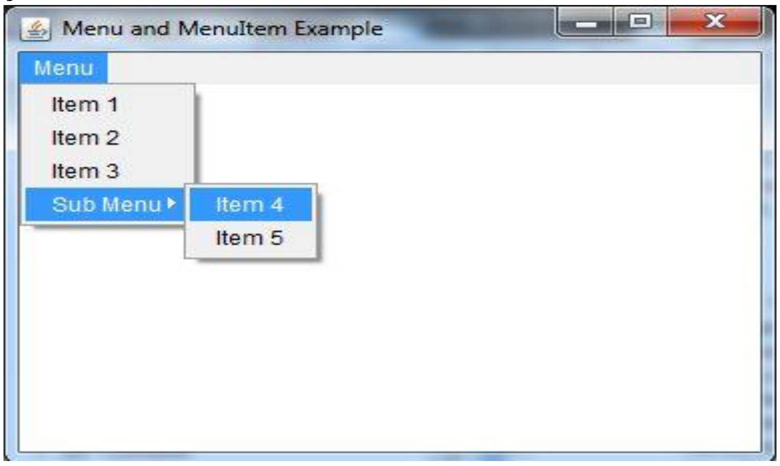
public class MenuItem extends MenuComponent

AWT Menu class declaration

public class Menu extends MenuItem

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```



Java 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:

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout etc.

Java 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:

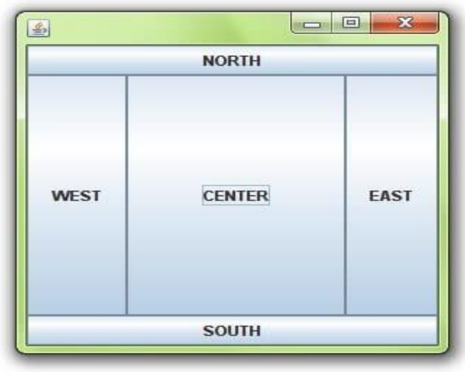
- public static final int NORTH
- public static final int SOUTH
- public static final int EAST
- public static final int WEST
- public static final int CENTER
-

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

BorderLayout Example:

```
import java.awt.*;
import java.awt.event.*;
public class Border {
Frame f;
Border()
{
   f=new Frame();
   Button b1=new Button("NORTH");
   Button b2=new Button("SOUTH");
   Button b3=new Button("EAST");
   Button b4=new Button("WEST");
   Button b5=new Button("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);
   f.addWindowListener(new WindowAdapter()
         {
              public void windowClosing(WindowEvent e)
              {
                    f.dispose();
                }
            });
}
public static void main(String[] args) {
   new Border();
}
}
```



Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

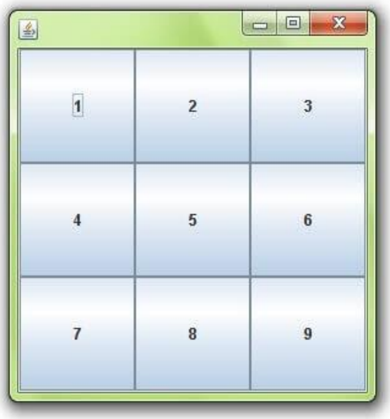
Constructors of GridLayout class:

- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
- GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class

```
import java.awt.*;
class MyGridLayout
{
 Frame f;
 MyGridLayout()
  {
    f=new Frame();
   Button b1=new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
```

```
    Button b4=new Button("4");
    Button b5=new Button("5");
    Button b6=new Button("6");
    Button b7=new Button("7");
    Button b8=new Button("8");
    Button b9=new Button("9");
    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);
    f.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}
```



Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

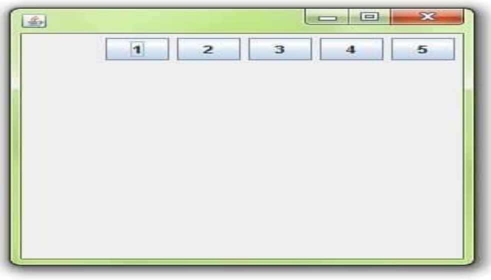
- public static final int LEFT
- public static final int RIGHT
- public static final int CENTER
- public static final int LEADING
- public static final int TRAILING

Constructors of FlowLayout class

- FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class:

```
import java.awt.*;
class MyFlowLayout
{
 Frame f;
 MyFlowLayout()
  {
    f=new Frame();
    Button b1=new Button("1");
    Button b2=new Button("2");
    Button b3=new Button("3");
    Button b4=new Button("4");
    Button b5=new Button("5");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment
    f.setSize(300,300);
    f.setVisible(true);
 }
 public static void main(String[] args)
 {
    new MyFlowLayout();
 }
}
```



Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

Fields of BoxLayout class

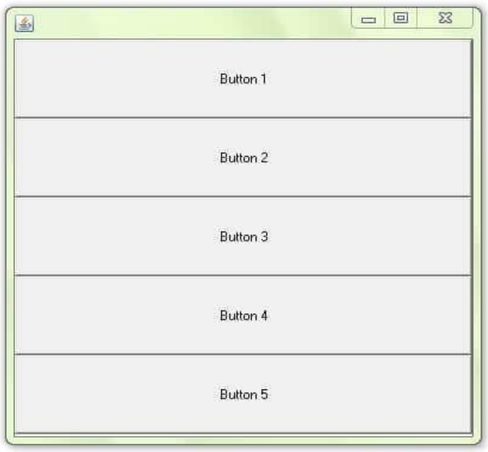
- public static final int X_AXIS
- public static final int Y_AXIS
- public static final int LINE_AXIS
- public static final int PAGE_AXIS

Constructor of BoxLayout class

- BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

Example of BoxLayout class with Y-AXIS:

```
import java.awt.*;
import javax.swing.*;
class BoxLayoutExample1 extends Frame
 {
   Button buttons[];
   BoxLayoutExample1 ()
  {
     buttons = new Button [5];
     for (int i = 0;i<5;i++)
     {
       buttons[i] = new Button ("Button " + (i + 1));
       add (buttons[i]);
     }
   setLayout (new BoxLayout (this,BoxLayout.Y_AXIS));
   setSize(400,400);
   setVisible(true);
}
public static void main(String args[])
{
  BoxLayoutExample1 b=new BoxLayoutExample1();
 }
}
```



Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

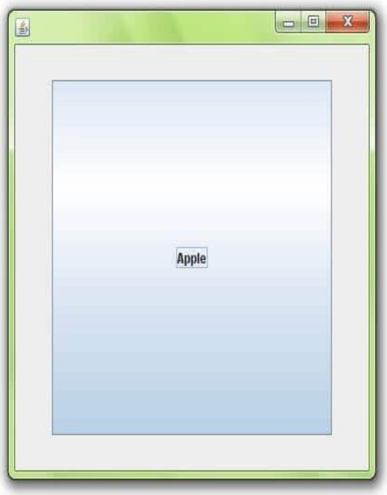
- CardLayout(): creates a card layout with zero horizontal and vertical gap.
- CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CardLayoutExample extends JFrame implements ActionListener
{
  CardLayout card;
  JButton b1,b2,b3;
  Container c;
  CardLayoutExample()
    {
      c=getContentPane();
      card=new CardLayout(40,30);
      //create CardLayout object with 40 hor space and 30 ver space
      c.setLayout(card);
      b1=new JButton("Apple");
      b2=new JButton("Boy");
      b3=new JButton("Cat");
      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);
      c.add("a",b1);c.add("b",b2);c.add("c",b3);

   }
   public void actionPerformed(ActionEvent e)
  {
   card.next(c);
   }

   public static void main(String[] args) {
      CardLayoutExample cl=new CardLayoutExample();
      cl.setSize(400,400);
      cl.setVisible(true);
      cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
   }
}
```



UNIT-VI

Describe Event Delegation Model

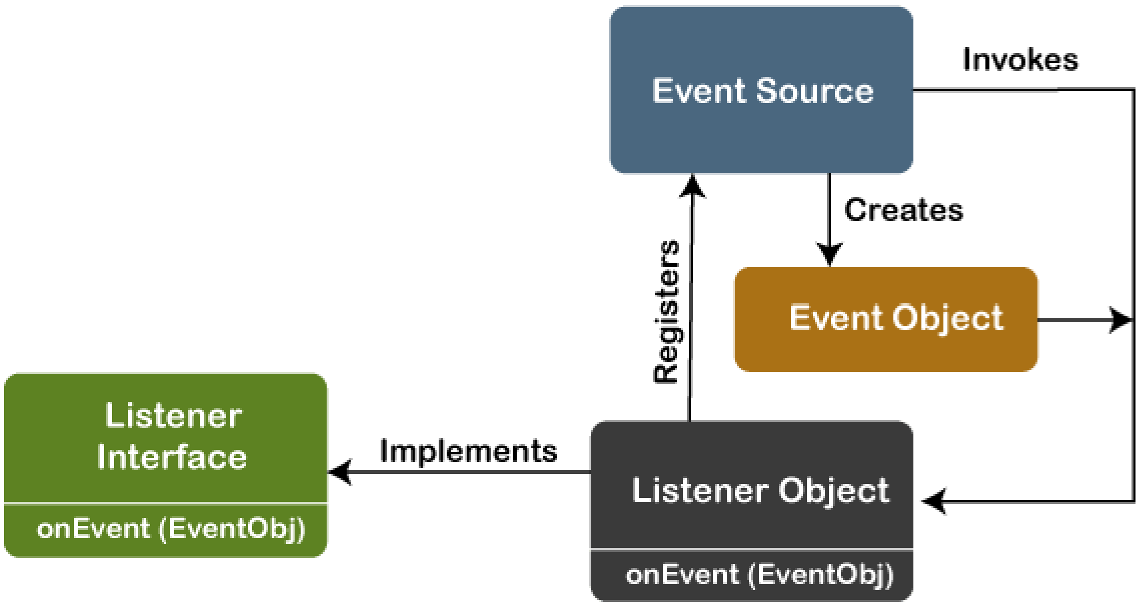
- The Delegation Event model is defined to handle events in GUI programming languages.

- The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

- The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

Event Processing in Java:

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods, below image demonstrates the event processing.



Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

- The Events are the objects that define state change in a source.
- An event can be generated as a reaction of a user while interacting with GUI elements.
- Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on.
- We can also consider many other user operations as events.
- The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc.
- We can define events for any of the applied actions.

Event Sources

- A source is an object that causes and generates an event.
- It generates an event when the internal state of the object is changed.
- The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method.

Below is an example:

public void addTypeListener (TypeListener e1)

- From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener.
- For example, for a keyboard event listener, the method will be called as addKeyListener().
- For the mouse event listener, the method will be called as addMouseMotionListener().
- When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object.
- This process is known as event multicasting. In few cases, the event notification will only be sent to listeners that register to receive them.

Event Listeners

- An event listener is an object that is invoked when an event triggers.
- The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events.
- Second, it must implement the methods to receive and process the received notifications.

Types of Events

The events are categories into the following two categories:

The Foreground Events:

- The foreground events are those events that require direct interaction of the user.
- These types of events are generated as a result of user interaction with the GUI component.
- For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

- The Background events are those events that result from the interaction of the end-user.
- For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

Adapter classes and inner classes with example programs

- 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**

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample{
    Frame f;
    AdapterExample(){
        f=new Frame("Window Adapter");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e) {
                f.dispose();
            }
        });

        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String[] args) {
    new AdapterExample();
}
}
```

Java MouseAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter{
   Frame f;
   MouseAdapterExample(){
      f=new Frame("Mouse Adapter");
      f.addMouseListener(this);

      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      Graphics g=f.getGraphics();
      g.setColor(Color.BLUE);
      g.fillOval(e.getX(),e.getY(),30,30);
   }

public static void main(String[] args) {
   new MouseAdapterExample();
}
}
```

Java KeyAdapter Example:

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter{
   Label l;
   TextArea area;
   Frame f;
   KeyAdapterExample(){
      f=new Frame("Key Adapter");
      l=new Label();
      l.setBounds(20,50,200,20);
      area=new TextArea();
      area.setBounds(20,80,300, 300);
      area.addKeyListener(this);

      f.add(l);f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
```

```
        f.setVisible(true);
    }
    public void keyReleased(KeyEvent e) {
        String text=area.getText();
        String words[]=text.split("\\s");
        l.setText("Words: "+words.length+" Characters:"+text.length());
    }

    public static void main(String[] args) {
        new KeyAdapterExample();
    }
}
```

Java Swings

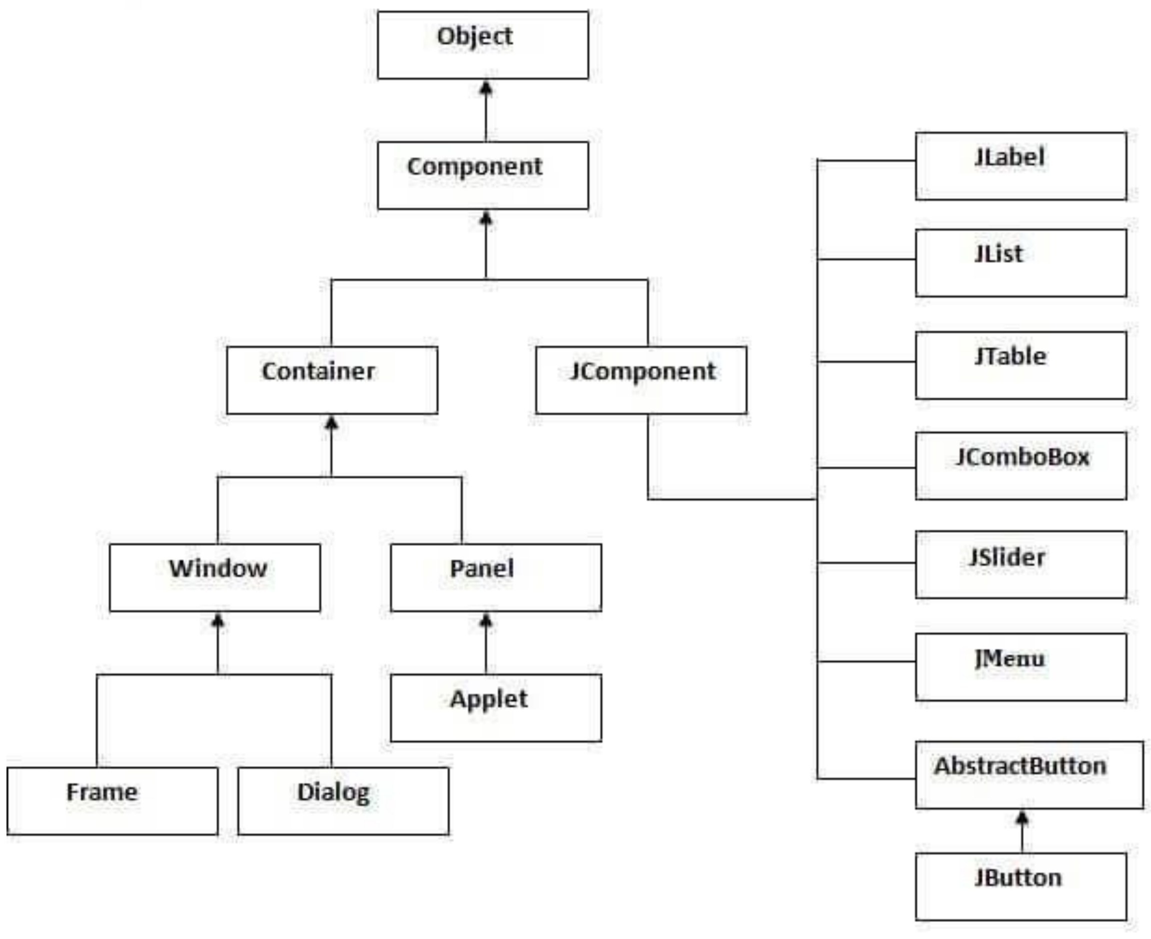
- Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing:

Java AWT	Java Swing
AWT components are platform-dependent**.**	**Java swing components are** platform-independent**.**
AWT components are heavyweight**.**	**Swing components are** lightweight**.**
AWT doesn't support pluggable look and feel**.**	**Swing** supports pluggable look and feel**.**
AWT provides less components **than Swing.**	**Swing provides** more powerful components **such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.**

AWT doesn't follows MVC(**Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.**	**Swing** follows MVC**.**

Hierarchy of Java Swing classes



Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

<u>Example of Swing by Association inside constructor:</u>

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}

public static void main(String[] args) {
new Simple();
}
}
```

<u>Simple example of Swing by inheritance</u>

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

public class **JButton** extends **AbstractButton**

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(booleanb)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.

void addActionListener(ActionListener a)	**It is used to add the action listener to this object.**

Java JButton Example 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("Welcome to Java.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Example of displaying image on the button:

```
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();
}
}
```

Java JLabel:

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

public class JLabel **extends** JComponent

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void	It sets the alignment of the label's contents along the

setHorizontalAlignment(int alignment)	X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

public class JTextField **extends** JTextComponent

Commonly used Constructors:

Constructor	**Description**
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

JTextArea class declaration

public class JTextArea **extends** JTextComponent

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.

JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
JLabel l1,l2;
JTextArea area;
JButton b;
TextAreaExample() {
    JFrame f= new JFrame();
    l1=new JLabel();
    l1.setBounds(50,25,100,30);
    l2=new JLabel();
    l2.setBounds(160,25,100,30);
    area=new JTextArea();
    area.setBounds(20,75,250,200);
    b=new JButton("Count Words");
    b.setBounds(100,300,120,30);
    b.addActionListener(this);
    f.add(l1);f.add(l2);f.add(area);f.add(b);
    f.setSize(450,450);
    f.setLayout(null);
    f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}
```

Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

public class JPasswordField **extends** JTextField

Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.

JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

Java JPasswordField Example

Java JPasswordField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample {
    public static void main(String[] args) {
    JFrame f=new JFrame("Password Field Example");
     final JLabel label = new JLabel();
     label.setBounds(20,150, 200,50);
     final JPasswordField value = new JPasswordField();
     value.setBounds(100,75,100,30);
     JLabel l1=new JLabel("Username:");
        l1.setBounds(20,20, 80,30);
        JLabel l2=new JLabel("Password:");
        l2.setBounds(20,75, 80,30);
        JButton b = new JButton("Login");
        b.setBounds(100,120, 80,30);
        final JTextField text = new JTextField();
        text.setBounds(100,20, 100,30);
                f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
                f.setSize(300,300);
                f.setLayout(null);
                f.setVisible(true);
                b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                   String data = "Username " + text.getText();
                   data += ", Password: "
                   + new String(value.getPassword());
                   label.setText(data);
                }
             });
}
}
```

Java JCheckBox

The JCheckBox 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 ".It inherits JToggleButton class.

JCheckBox class declaration

public class JCheckBox **extends** JToggleButton

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

public class JRadioButton **extends** JToggleButton

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example 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();
}}
```

Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

public class JComboBox **extends** JComponent

Commonly used Constructors:

Constructor	**Description**
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

Methods	**Description**
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
    f=new JFrame("ComboBox Example");
    final JLabel label = new JLabel();
    label.setHorizontalAlignment(JLabel.CENTER);
    label.setSize(400,100);
    JButton b=new JButton("Show");
    b.setBounds(200,100,75,20);
    String languages[]={"C","C++","C#","Java","PHP"};
    final JComboBox cb=new JComboBox(languages);
    cb.setBounds(50, 100,90,20);
    f.add(cb); f.add(label); f.add(b);
    f.setLayout(null);
    f.setSize(350,350);
    f.setVisible(true);
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
String data = "Programming language Selected: "
   + cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                          {"102","Jai","780000"},
                          {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```

Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

```
1. public class JList extends JComponent
```

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, mode

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified e time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a lis items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an a of objects.

Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

public class JTree **extends** JComponent

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object [] value)	Creates a JTree with every element of the specified array as the child o new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displ the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
JFrame f;
TreeExample(){
    f=new JFrame();
    DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
    DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
    DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
    style.add(color);
    style.add(font);
    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");

    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");

    color.add(red); color.add(blue); color.add(black); color.add(green);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
  }
  public static void main(String[] args) {
    new TreeExample();
  }}
```

Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane class declaration

```
1. public class JOptionPane extends JComponent implements Accessible
```

Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	**It is used to create a JOptionPane with a test message.**
JOptionPane(Object message)	**It is used to create an instance of JOptionPane to display a message.**
JOptionPane(Object message, int messageType	**It is used to create an instance of JOptionPane to display a message specified message type and default options.**

Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parent JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-mess dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with gi title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dia requesting input from the user parented parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selec or input by the user.

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOpti
onPane.WARNING_MESSAGE);
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Java JOptionPane Example: showInputDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

Java JOptionPane Example: showConfirmDialog()

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        f.addWindowListener(this);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        f.setVisible(true);
    }
    public void windowClosing(WindowEvent e) {
        int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
    if(a==JOptionPane.YES_OPTION){
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
    }
```

Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

```
public class JTabbedPane extends JComponent
```

Commonly used Constructors:

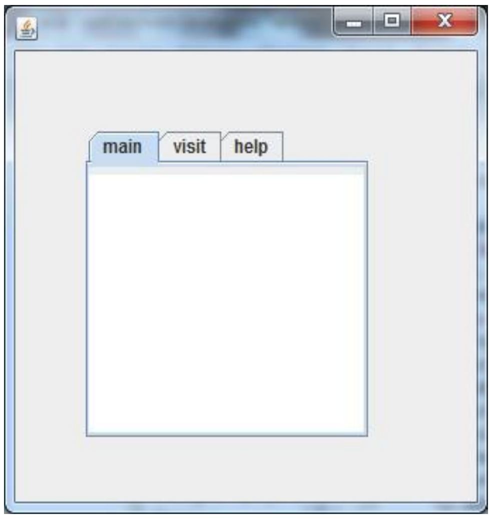
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.

JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
   f=new JFrame();
   JTextArea ta=new JTextArea(200,200);
   JPanel p1=new JPanel();
   p1.add(ta);
   JPanel p2=new JPanel();
   JPanel p3=new JPanel();
   JTabbedPane tp=new JTabbedPane();
   tp.setBounds(50,50,200,200);
   tp.add("main",p1);
   tp.add("visit",p2);
   tp.add("help",p3);
   f.add(tp);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new TabbedPaneExample();
}}
```

Output:



Java JLayeredPane

The JLayeredPane class is used to add depth to swing container. It is used to provide a third dimension for positioning component and divide the depth-range into several different layers.

JLayeredPane class declaration

```
public class JLayeredPane extends JComponent
```

Commonly used Constructors:

Constructor	Description
JLayeredPane	It is used to create a new JLayeredPane

Commonly used Methods:

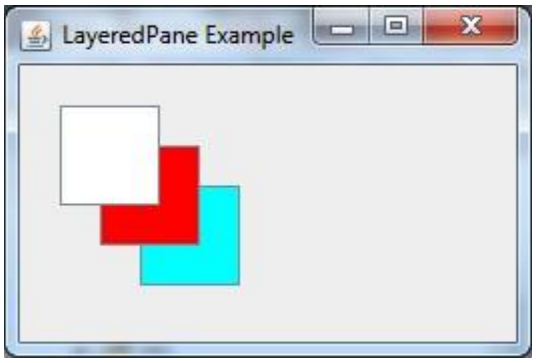
Method	Description
int getIndexOf(Component c)	It is used to return the index of the specified Component.

int getLayer(Component c)	It is used to return the layer attribute for the specified Component.
int getPosition(Component c)	It is used to return the relative position of the component within its layer.

Java JLayeredPane Example

```
import javax.swing.*;
import java.awt.*;
public class LayeredPaneExample extends JFrame {
  public LayeredPaneExample() {
  super("LayeredPane Example");
    setSize(200, 200);
    JLayeredPane pane = getLayeredPane();
    //creating buttons
    JButton top = new JButton();
    top.setBackground(Color.white);
    top.setBounds(20, 20, 50, 50);
    JButton middle = new JButton();
    middle.setBackground(Color.red);
    middle.setBounds(40, 40, 50, 50);
    JButton bottom = new JButton();
    bottom.setBackground(Color.cyan);
    bottom.setBounds(60, 60, 50, 50);
    //adding buttons on pane
    pane.add(bottom, new Integer(1));
    pane.add(middle, new Integer(2));
    pane.add(top, new Integer(3));
  }
  public static void main(String[] args) {
      LayeredPaneExample panel = new LayeredPaneExample();
      panel.setVisible(true);
  }
}
```

Output:



Java JPanel

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

JPanel class declaration

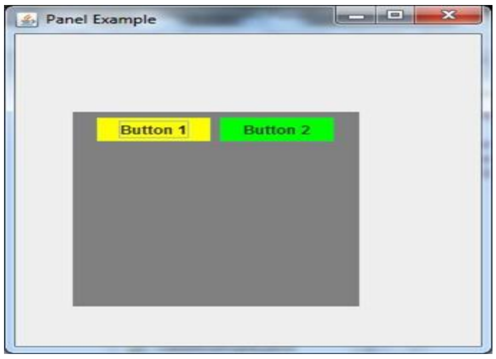
```
public class JPanel extends JComponent
```

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

Java JPanel Example

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
      {
      JFrame f= new JFrame("Panel Example");
      JPanel panel=new JPanel();
      panel.setBounds(40,80,200,200);
      panel.setBackground(Color.gray);
      JButton b1=new JButton("Button 1");
      b1.setBounds(50,100,80,30);
      b1.setBackground(Color.yellow);
      JButton b2=new JButton("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:
```



Panel Example
Button 1
Button 2