JAVA

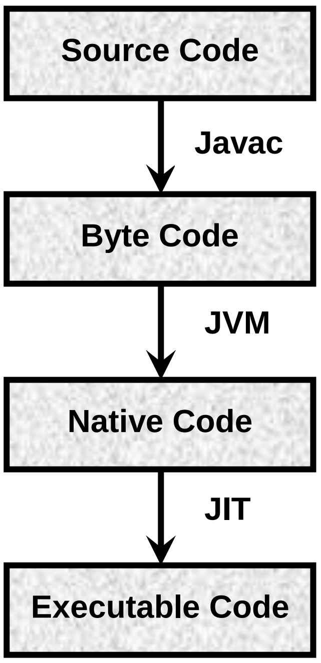
History of Java

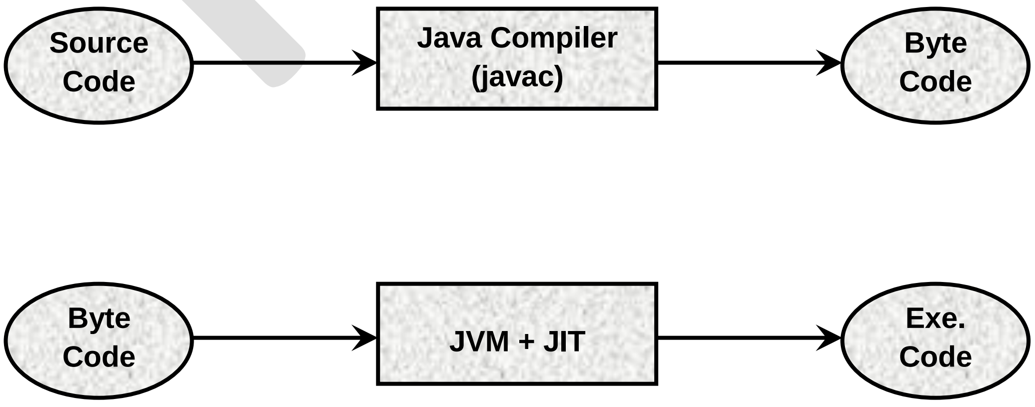
- Java is **Platform Independent (Portable)** Programming Language.
- Java is **Object Oriented** Language.
- Java was developed in **1991** at Sun Micro system Inc.
- It took **18** months to develop.
- It was developed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan
- Its original name was "**OAK**" but in 1995 it was renamed as "**JAVA**"

Characteristics of Java(5 marks)

1.) Simple: - Easy to learn and can be used effectively.
2.) Secure: - Java helps you to download the data from internet without the fear of virus attack.
3.) Portable: - The programs made in Java can run at any system.
4.) Object Oriented: - it supports all features of object oriented language.
5.) Robust: - Ability to provide reliable program development.
6.) Multithreaded: - More than two processes can be run simultaneously.
7.) Architecture Neutral: - Write once, run anywhere, anytime forever.
8.) Distributed: - Java can be used across the network.
9.) Dynamic: - It provides Verification of many things at run time.
10.) Memory management: - It provides some mechanism like garbage collection.
11.) Exception Handling: - You can also handle error effectively in Java.

Why Java is Portable Language? (3 marks)





- ► The program written by user is called Source code.
- ► This Source code with the help of java compiler converted to Byte code.
- ► This Byte code is converted to appropriate Native code with the help of **JVM (Java Virtual Machine).**
- ► Native code is ready to execute code
- ► JVM is the compiler which is different in all different operating system.
- ► JVM of windows will not work on Linux.
- ► The Native code with the help of **JIT (Just In Time)** converted to Executable code.

$$\begin{tikzcd}
 & \textbf{Java} \arrow[d] & \\
 & \textbf{Byte Code} \arrow[d] & \\
 & \textbf{JVM (JIT)} \arrow[dl] \arrow[d] \arrow[dr] & \\
\textbf{Windows} & \textbf{Linux} & \textbf{Macintosh}
\end{tikzcd}$$

JIT (Just In Time)

If any code is compiled two times than it might run slow as expected. But that's not the thing with Java. After initial released of Java version the Sun Micro System started providing the HotSpot technology.

This technology provides JIT compiler to native code. The selected portion of native code is compiled to executable code piece by piece, as demanded.

It is not possible to compile whole Java program all at once because Java does some rum time checking. JIT compiles the code as needed and converts it to executable code.

JVM + JIT is called Application Launcher

Basics Of Java

- ➢ Extension of Java program is **.java**
- ➢ Java program must consists **at least one class**
- ➢ The **no. of class** consists in the program that much **no. of .class files** will be generated.
- ➢ The class name which consist main method and the program file name must be same.
- ➢ The byte code will be generated with **.class** extension.
- ➢ Java is **case sensitive** language.
- ➢ To compile java program use **Javac** command (javac filename.java)
- ➢ To execute java program use **Java** command (java filename)

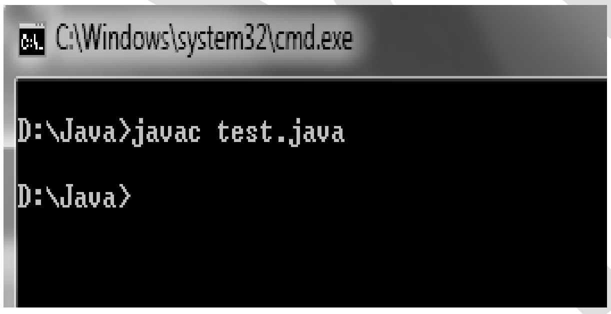
JDK and its Components

If you are using Java Development environment than you may need to follow different procedure for compiling and executing Java programs. Necessary objects required to execute a Java program is available in **JDK (Java Development Kit).** JDK is the installation package of Java. It has a collection of components that are used in development of Java program.

JDK Tools	Meaning	Use
javac	Java Compiler	Compiles source code to byte code
java	Java interpreter	Interprets byte code and generates the output
jdb	Java debugger	Used to debug your program
appletviewer	Applet Viewer	Used to execute Java applet programs
javadoc	Java Documentation	Used to create documentation for Java code.

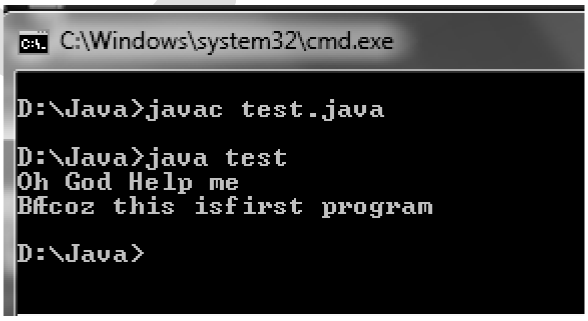
First Program (test.java)

```
class test
{
        public static void main(String args[ ] )
        {
                System.out.println("Oh God Help me");
                System.out.print("B'coz this is ");
                System.out.println("first program");
        }
}
```



```
C:\Windows\system32\cmd.exe

D:\Java>javac test.java

D:\Java>
```

Compiling



```
C:\Windows\system32\cmd.exe

D:\Java>javac test.java

D:\Java>java test
Oh God Help me
B‰coz this is first program

D:\Java>
```

Executing

public: - It is access specifier which allows to call method from outside the class.

static: - The static keyword allows you to call main method without the object by using class name.

void : - The main method does not return any value.

main( ):- The program execution always starts from main method.

String args[ ]: - it gets the parameter from command line argument and store it in array of string.

System.out.println( ): - Here System is the default class which is predefined. 'Out' is the output stream and 'println' is the method used to display data on console.

Language Building Blocks Or Lexical Issues (3 marks)

The things we use to create java program is called building blocks. It includes the following things

1) **Comments**: - There are three types of comments available in Java Single line(**//**), Multi line (**/*….*/**) and Documentation (**/**…..*/**).

2) **Identifiers** : - It is name given to the variables which can store some values. It has some rules to follow
   a) It may consist of alphabets, number, underscore and dollar sign
   b) It must not start with number.
   c) No white space is allowed
   d) No keyword can be used as identifier.
   e) Some valid names are **mark1, first_value, $price**
   f) Some invalid names are **1mark, #name, first name**

3) **Literals**: - Any constant value is called literals. Ex 100, 38.95, 'A', "hello world" etc

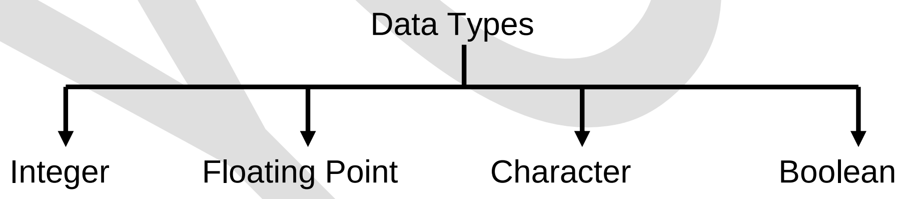
4) **White Space**: - White space can be single space, new line or tab

5) **Separators**: - It is used to separate two statements. The most commonly used is semicolon (**;**). Other are comma(**,**), colon (**:**), Braces (**{ }**) etc

6) **Keywords**: - It is also called reserved words. There are 49 keywords available in Java. Keywords cannot be used as variable name.

7) **Operators**: - Used for different operations. In java we have arithmetic operators, logical operator, relational operators etc.

Data Types available in Java (3 – 5 marks)



Integer: - It is used to store whole number only. All are signed so that they can store negative as well as positive values.

Data type	Size (byte)	Range	Default Value
byte	1	-128 to 127	0
short	2	-32768 to 32767	0
int	4	-2147483648 to 2147483647	0
long	8	--	0L

Floating Point: - It is used to store fractional numbers.

Data type	Size (byte)	Range	Default Value
float	4	1.4e – 045 to 3.4e +038	0.0f
double	8	4.9e -324 to 1.8e +308	0.0d

Character: - It is used to store single character.

Data type	Size (byte)	Range	Default Value
char	2	0 to 65535	'\0'

Boolean: - It is 8 bit data type used to store logical value i.e. true/false.

Data type	Size (byte)	Range	Default Value
Boolean	1	true/false	false

Operators in Java (3 – 5 marks)

Operators	Symbols
Arithmetic	+ ,- , * , / , %
Relational	<, >, <=, >=, !=, ==
Logical	&&, ||, !
Bitwise	&, |, ^, <<, >>
Assignment	=, +=, -=, *=, /=, %=
Conditional	? :

Operator Precedence: - It means in what order the expression should be evaluated so that it gives correct result.

/, *, %	**1st Priority**
+, -	**2nd Priority**
=	**3rd Priority**

Type Casting (2 – 3 marks)

In many programs we generally assign the value of one variable to other variable. In Java, before assigning values to a variable, if their types are not same then type conversion required. Type casting means converting one data type to another data type.

There are two type of type casting

1) Implicit (Automatic) type casting
2) Explicit type casting.

Implicit (Automatic) casting: - This is automatically done by Java compiler when the types are compatible and destination type is larger than the source type. For example if we assign int value to long type than it will be converted automatically.

From	To
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long,	float, double
float	double

Explicit casting: - The expression of destination type is not larger than the source type then the conversion will not be done automatically, we have to convert it explicitly. For example if we want to store long value to int then it must be done explicitly.

```
long a = 9999999;
int  i = (int) a;

class Conversion
 {
        public static void main(String args[])
        {
                byte b;
                int  i = 257;
                long L;
                double d = 323.142;
                L = i    // Implicit Conversion
                System.out.println("\nConversion of int to byte.");
                b = (byte) i;    // Explicit Conversion
                System.out.println("i and b " + i + " " + b);
                System.out.println("\nConversion of double to int.");
                i = (int) d;       // Explicit Conversion
                System.out.println("d and i " + d + " " + i);
                System.out.println("\nConversion of double to byte.");
                b = (byte) d;  // Explicit Conversion
                System.out.println("d and b " + d + " " + b);
        }
}
```

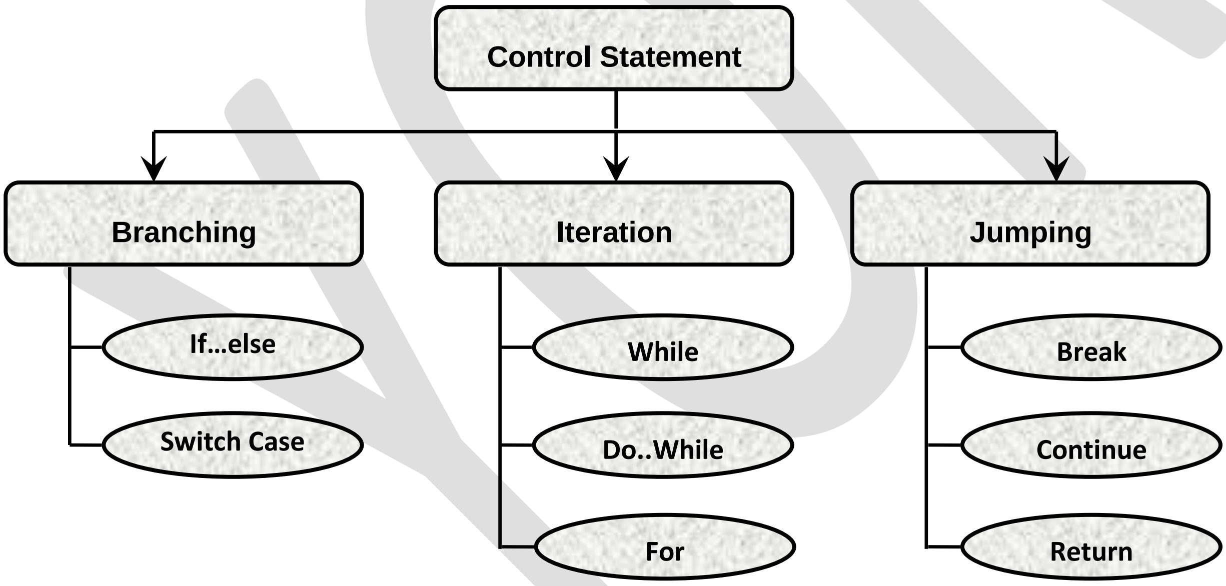
<u>Scope and Life time of a Variable (3 marks)</u>

When we declare a variable, it has a specific scope up to where it can be accessed and can be used. When any variable is declared within block then scope of that variable will be within that block only. If you create any variable within a method then scope of that variable will be within that method.

The region of program in which a variable is available to access is called scope of variable. The region between two curly brackets is called block. A block defines a scope. Each time you start a new block, you are creating new scope.

```
class scope
{
      public static void main(String args[ ])
      {  <──────────────────────────── Starts scope1
            int x = 100;
            for (int i = 1; i<=3; i++)
            {  <──────────────────────── Starts scope2
                    int x = 500;            // Re-declaration error
                    int y = 200;            // known only to this block
                  System.out.println ("x = " + x);
                  System.out.println ("y = " + y);

            } // scope2 over

            System.out.println ("x = " + x);
            System.out.println ("y = " + y);        // Error Out of scope

            int y = 1000;
            System.out.println ("y = " + y);
      } // scope1 over
}
```

Control Statement(3-5 marks)



Branching or Decision making Statement

Java supports two branching statements: **if and switch**. These statements allow you to control the flow of your program's execution based upon conditions known only during run time.
if….else statement : It is a simplest branching statement we use. Most often, the expression used to control the if statement will involve the relational operators.

```
Syntax                  if (condition)
                        {
                                ……………..
                        }
                        else
                        {
                                ………….
                        }
```

Nested ifs

A nested if is an if statement that is the target of another if or else. Nested ifs are very common in programming. When you nest ifs, the main thing to remember is that an else statement always refers to the nearest if statement that is within the same block as the else and that is not already associated with an else.

Here is an example:

```
if(i == 10)
{
        if(j < 20)
           a = b;
                if(k > 100)     // this if is
                        c = d;
                else            // associated with this else
                        a = c;
}
else
        a = d;          // this else refers to if(i == 10)
```

As the comments indicate, the final else is not associated with if(j<20), because it is not in the same block (even though it is the nearest if without an else). Rather, the final else is associated with if(i==10). The inner else refers to if(k>100), because it is the closest if within the same block.

The if-else-if Ladder

A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder. It looks like this:

```
if(condition)
        statement;
else if(condition)
        statement;
else if(condition)
        statement;
.
.
.
else
        statement;
```

The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. The final else acts as a default condition; that is, if all other conditional tests fail, then the last else statement is performed. If there is no final else and all other conditions are false, then no action will take place.

Switch Case

The switch statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of if-else-if statements.

Here is the general form of a switch statement:

```
switch (expression)
 {
        case value1:
                // statement sequence
                break;
        case value2:
                // statement sequence
                break;
                .
                .
                .
        case valueN:
                // statement sequence
                break;
        default:
                // default statement sequence
}
```

The expression must be of type byte, short, int, or char; each of the values specified in the case statements must be of a type compatible with the expression. Each case value must be a unique literal (that is, it must be a constant, not a variable). Duplicate case values are not allowed.

The switch statement works like this: The value of the expression is compared with each of the literal values in the case statements. If a match is found, the code sequence following that case statement is executed. If none of the constants matches the value of the expression, then the default statement is executed. However, the default statement is optional. If no case matches and no default is present, then no further action is taken.

The break statement is used inside the switch to terminate a statement sequence. When a break statement is encountered, execution branches to the first line of code that follows the entire switch statement. This has the effect of "jumping out" of the switch.

Looping Statement

Looping statements are used to execute some statement repeatedly. There are three types of looping statements.

1) **For loop**
2) **While loop**
3) **Do..while loop**

1) **For Loop**: - It is mostly used loop. It is entry controlled loop.

Syntax

```
        for(Exp1; Exp2; Exp3)
        {
                - - - - - - - - - - -
                 - - - - - - - - - - -
                - - - - - - - - - - -
        }
```

Exp1 $\longrightarrow$ Initialisation
Exp2 $\longrightarrow$ Condition
Exp3 $\longrightarrow$ Increment/Decrement

You can also use a loop within another loop that is called nesting of loop.

2) **While loop**: - It is used when you don’t know how many number of iteration you want to execute.

Syntax

```
Initialisation
        While(condition)
        {
                - - - - - - - - - - -
                 - - - - - - - - - - -
                - - - - - - - - - - -
                Increment/Decrement
        }
```

3) **Do..While loop**: - It is exit controlled loop. In do..while the statements are executed first and then the condition is checked. If the condition is true the block is executed again. Thus the statements are executed at least once even if the condition is false.

Syntax

```
Initialisation
Do
{
        - - - - - - - - - - -
         - - - - - - - - - - -
        - - - - - - - - - - -
        Increment/Decrement
} While(condition);
```

Jumping Statement (3 marks)

These statements are generally used to jump to another location of program. There are three types of jumping statements.

1) **Break**
2) **Continue**
3) **Return**

1) **Break**: - When we want to jump out of the loop without waiting to get back for condition check. Then we can use break statement. Break I used to terminate the loop. A break statement is generally associated with if statement. The break statement transfers the control at the end of the loop.

```
for(            )
{
        if(condition)
                break;
        - - - - - - - - - - -
         - - - - - - - - - - -
}
```

Labelled Break: - To break a particular loop we can give a label to the loop and use break statement before that label

```
Label:
for(    )
{
        if(condition)
                break Label;
        - - - - - - - - - - -
}
```

class BreakLoop4

```
{
        public static void main(String args[ ])
         {
                outer:
                for(int i=0; i<3; i++)
                {
                        System.out.print("Pass " + i + ": ");
                        inner:
                        for(int j=0; j<100; j++)
                        {
                                if(j == 10)
                                   break outer; // exit both loops
                                System.out.print(j + " ");
                        }
                        System.out.println("This will not print");
                }
        System.out.println("Loops complete.");
        }
}
```

```
class BreakErr
{
        public static void main(String args[])
        {
                one:
                for(int i=0; i<3; i++)
                 {
                        System.out.print("Pass " + i + ": ");
                }
                for(int j=0; j<100; j++)
                {
                        if(j == 10)
                          break one; // WRONG
                        System.out.print(j + " ");
                }
        }
}
```

2) **Continue**: - Continue statement skip the current iteration and continues the loop to the next iteration. So when we want to skip some statements based on condition the continue can be used. It passes control at the beginning of the loop. It is usually associated with if statement.

```
        for(            ) <------------+
        {                              |
                if(condition)          |
                        continue; -----+
                - - - - - - - - - - -
                 - - - - - - - - - - -
        }
```

Labelled Continue: - To continue a particular loop we can use labelled continue.

```
                outer:
                for(int i=0; i<3; i++)
```

```
{
        inner:
        for(int j=0; j<100; j++)
        {
                System.out.print(j + " ");
                if(j == 10)
                   continue outer;
                else
                   continue inner
        }
}
```

3) **Return**: - It returns control to the calling function. It returns value to the calling function. Only one value is returned to calling function.

Array (3 marks)

Array is the collection of variable with similar data type that shares a common name. Individual value of array is called element. We cannot have array of 10 numbers in which 5 are of integer and 5 are of float data type. We can access the elements of array by its index. Index always starts with zero.

Single Dimension Array: - It is collection of array with one row and multiple columns.

Syntax	data type array_name[ ] ;
Example	**int marks[ ]; // declaration**

Here marks is said to be array of integers. The array is jus created but not ready to use. To use this array we must allocate memory space to it using new keyword.

Syntax	array_name = new data type[size];
Example	**marks = new int [5]; // creation**

marks[ ] = {10,20,30,40,50}   // initialization

Multi Dimension Array: - It represents a variable which has values in tabular form i.e. rows and columns.

Syntax	data type array_name[ ] [ ] = new data type[row] [col]
Example	int A[ ][ ] = new int [3] [3];

A [ ] [ ] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};

Class

A class can be declared with the use of class keyword. It defines a new data type. Once it defined, it can be used to create object of the type.

Syntax

```
        class classname
        {
                Type variable 1;
                Type variable 2;
                :
                :
                Type variable N;

                Type method 1( parameter )
                {
                }
                Type method 2 ( parameter )
                {
                }
                :
                :
                Type method N( parameter )
                {
                }
        };
```

The variables defined within a class are called **instance variables**. The coding is written within methods. The methods and variables defined within a class are called members of the class. The variables are accessed by the methods defined in that class. So method determines how a class data can be used.

Each instance of class contains its **own copy** of these variables. The data for one object is separate and unique from the data for another object.

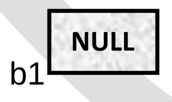
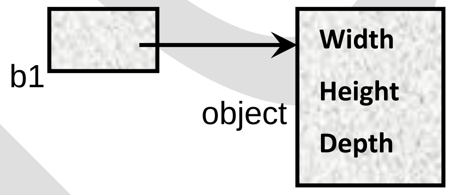
```
// This program declares two Box objects.
class Box
{
        double width;
        double height;
        double depth;
}
class BoxDemo2
{
        public static void main(String args[ ])
        {
                Box mybox1 = new Box( );
                Box mybox2 = new Box( );
                double vol;
                // assign values to mybox1's instance variables
                mybox1.width = 10;
                mybox1.height = 20;
                mybox1.depth = 15;
                /* assign different values to mybox2's
                instance variables */
                mybox2.width = 3;
                mybox2.height = 6;
                mybox2.depth = 9;

                // compute volume of first box
                vol = mybox1.width * mybox1.height * mybox1.depth;
                System.out.println("Volume is " + vol);

                // compute volume of second box
                vol = mybox2.width * mybox2.height * mybox2.depth;
                System.out.println("Volume is " + vol);
        }
}
```

New Operator (2 – 3 marks)

Creating object is two step process. First you must declare a variable of the class type. Second you must allocate memory to that object. You must acquire an actual copy of the object and assign it to that variable. This can be done dynamically by using new operator. The new operator allocates memory for an object.

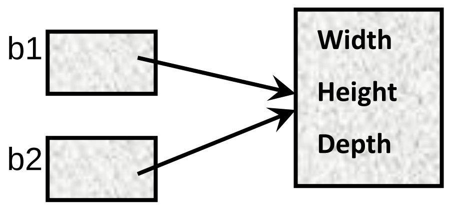
```
Box b1;          // declares reference
b1 = new Box( );      // allocates memory
```

The new keyword is used to allocate memory equal to size of instance variables of class. Before using new keyword the object cannot be used. The object gets physical location only after memory allocation by new keyword.

Statement	Effect
Box b1	
b1 = new Box( );	

Reference Variable

```
Box b1 = new Box( );
Box b2 = b1;
```



Here both b1 and b2 pointing to the same object. Here b1 does not allocate any memory to b2. It simply makes b2 refer to the same object as b1. So nay changes made to the object with b2 will affect the object to which b1 is referring

```
class Box
{
        double width;
        double height;
        double depth;
        // display volume of a box
        void volume( )
        {
                System.out.print("Volume is ");
                System.out.println(width * height * depth);
        }
}
class BoxDemo3
{
        public static void main(String args[ ])
        {
                Box mybox1 = new Box( );
                Box mybox2 = new Box( );

                // assign values to mybox1's instance variables
                mybox1.width = 10;
                mybox1.height = 20;
                mybox1.depth = 15;

                // assign values to mybox2's instance variables
                mybox2.width = 3;
                mybox2.height = 6;
                mybox2.depth = 9;

                // display volume of first box
                mybox1.volume( );

                // display volume of second box
                mybox2.volume( );
        }
}
```

➔ Write program for returning value and passing the parameter to method.

Constructor (3 marks)

- It is special method which has same name as its class.
- It is automatically invoked when object is created.
- It does not have any return type not even void.
- It is used to initialize variables of class when object is created.
- It can have parameters.
- It can be overloaded.

```
class Box
{
        double width;
        double height;
        double depth;
        // This is the constructor for Box.
        Box( )
         {
                System.out.println("Constructing Box");
                width = 10;
                height = 10;
                depth = 10;
        }
        // compute and return volume
        double volume()
         {
                return width * height * depth;
        }
}
class BoxDemo6
 {
        public static void main(String args[])
        {
                // declare, allocate, and initialize Box object
                Box mybox1 = new Box();
                Box mybox2 = new Box();
                double vol;
                // get volume of first box
                vol = mybox1.volume();
                System.out.println("Volume is " + vol);
                // get volume of second box
                vol = mybox2.volume();
                System.out.println("Volume is " + vol);
        }
}
```

➔ Write same program for parameterised constructor

“this” keyword (2 marks)

Sometimes a method will need to refer to the object that invoked it. Java defined “this” keyword for this. “this” can be used inside any method to refer current object.

Sometimes we have same names of instance variables and other local variables. “this” keyword can be used to distinguish them.

```
        Box(double width, double height, double depth)
        {
                this.width = width;
                this.height = height;
                this.depth = depth;
        }
```

Garbage Collection (2 marks)

Since the object memory is dynamically allocated by using new keyword. But the de-allocation of memory is also required. In C++ we have concept of delete operator to release memory manually. But in java de-allocation is done automatically. The technique is called Garbage collection. When no reference to an object exists then that object is no longer needed and the memory occupied by that object can be released.

Finalize ( ) method (2 marks)

Before object is destroyed it must perform some action such as free the resources hold by that object. For this java provides finalize( ) method.

With the help of finalize( ) method you can define specific action when object is just about to destroyed by garbage collector. Inside finalize( ) method you must specify those actions that must be performed before object is destroyed.

Method Overloading (3 – 5 marks)

It is one of the ways that Java supports polymorphism. If there is more than one method in class with same name but different number or types of parameter then method is called overloaded and this process is called method overloading. The number and type of parameter is called signature of method.

When method is invoked Java tries to find exact match of the method. If no exact type is found then java tries to convert the type.

```
class OverloadDemo
{
        void test()
        {
                System.out.println("No parameters");
        }
        // Overload test for one integer parameter.
        void test(int a)
        {
                System.out.println("a: " + a);
        }
        // Overload test for two integer parameters.
        void test(int a, int b)
        {
                System.out.println("a and b: " + a + " " + b);
        }
        // overload test for a double parameter
        double test(double a)
        {
                System.out.println("double a: " + a);
                return a*a;
        }
}
class Overload
{
        public static void main(String args[])
        {
                OverloadDemo ob = new OverloadDemo();
                double result;
                // call all versions of test()
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.25);
                System.out.println("Result of ob.test(123.25): " + result);
        }
}
```

Constructor Overloading

When we create more than one constructor with different parameter it is called constructor overloading. Sometimes it is needed to create object with different parameters then we can use constructor with different parameters.

```
class Box
{
        double width;
        double height;
        double depth;
        // constructor used when all dimensions specified
        Box(double w, double h, double d)
        {
                width = w;
                height = h;
                depth = d;
        }
        // constructor used when no dimensions specified
        Box()
        {
                width = -1;  // use -1 to indicate
                height = -1; // an uninitialized
                depth = -1;  // box
        }
        // constructor used when cube is created
        Box(double len)
        {
                width = height = depth = len;
        }
        // compute and return volume
        double volume()
        {
                return width * height * depth;
        }
}
class OverloadCons
{
        public static void main(String args[ ])
        {
                // create boxes using the various constructors
                Box mybox1 = new Box(10, 20, 15);
                Box mybox2 = new Box( );
                Box mycube = new Box(7);
                double vol;
                // get volume of first box
                vol = mybox1.volume();
                System.out.println("Volume of mybox1 is " + vol);

                // get volume of second box
                vol = mybox2.volume();
                System.out.println("Volume of mybox2 is " + vol);

                // get volume of cube
                vol = mycube.volume();
                System.out.println("Volume of mycube is " + vol);
        }
}
```

Static Members (3 marks)

We can access the member variables and method by the object of its class. The static members are initialized and static methods are executed before any object of its class are created. The main( ) method is static because it must be executed before any object of that class is created. You can also define a block as static. This static block will be executed when class is loaded in memory.

- The static members can call only static methods and access only static variables.
- They cannot refer to "this" or "super" keyword.

```
class UseStatic
{
        static int a = 3;
        static int b;
        static void meth(int x)
        {
                System.out.println("x = " + x);
                System.out.println("a = " + a);
                System.out.println("b = " + b);
        }
        static
         {
                System.out.println("Static block initialized.");
                b = a * 4;
        }
        public static void main(String args[ ])
         {
                meth(42);
        }
}
```

Access Specifier (3 marks)

It is the attribute of encapsulation. With access control you can manage what part of a program can be accessed. There are four type of Access control available in Java

1) **Public**: - The members defined by public specifier can be accessed from anywhere i.e. by its class member and also by other class members.

2) **Private**: - The private members can only be accessed within class. The member of other class cannot access private members.

3) **Protected**: - This specifier is used in inheritance only. The protected member can be accessed by the member of other package but only to the subclass of the class.

4) **Default**: - When you do not specify any access specifier, the default specifier is applied to that member. The default members can be accessed from the member of any class but in same package.

Final Keyword (3 marks)

There are **three uses** of final keyword. The final keyword can be applied to variables, method and classes.

- Declaring a **variable** as final makes it **constant**. So the content of the final variable cannot be modified by any statement. Thus final variable can be used as read only.
- Second use of final keyword is to **prevent method over riding**. The method which is declared as final is cannot be over ridden. Once the method is declared as final you cannot over ride that method to its child class.
- Third use of final keyword is to **prevent inheritance**. The class declared as final is cannot be inherited. Once you declared a class as final than it cannot become parent class or derived class. Means that class cannot be inherited.

YOR COMPUTER CLASSES

C Language	Oracle
C++	Java
HTML	Adv. Java
Data Structure	C# .NET
COA	ASP.NET
Visual Basic 6.0	SQL Server
PHP	MS-Office

- ✓ Practical Lab with latest technology
- ✓ Library with all kinds of reference books & materials
- ✓ Personal attention on each student
- ✓ Best material will be provided

YOR COMPUTER CLASSES

Vishal Sir(MCA)	**Naman Sir(MCA)**
9427732231	**9408526428**

101 Shiromani Complex, Nr. Balaji Mandir,
Karansinghji Highschool Chowk,
Above YOR Restaurant, Rajkot - 360001

Inheritance

A class can acquire the properties of another class is called inheritance. It provides the facility of reusability. The elements are created once and can be reuse again at many places with the help of inheritance.

The class which is being inherited is called super class or parent class and the class that inherits the properties of super class is called subclass or child class. A subclass is specialized version of a super class. It inherits all of the instance variable and methods defined by super class.

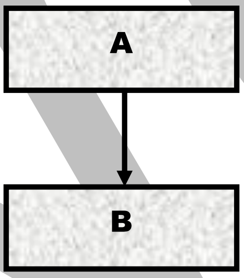
To inherit a class we use “extends” keyword. The subclass uses extends keyword after subclass name.

Syntax:

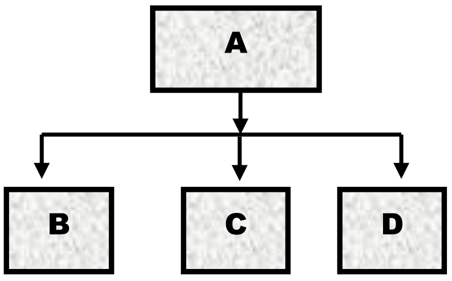
```
class super
{
        - - - - - - - -
        - - - - - - - -
}
class sub extends super
{
        - - - - - - - -
        - - - - - - - -
}
```

Types of inheritance

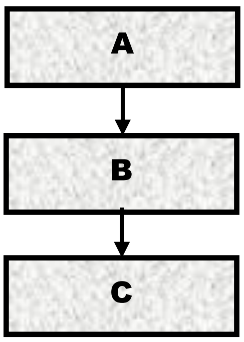
1) **Single Inheritance**: - If there is only one super class and one subclass then that inheritance is single inheritance.

	class A
	{
	        -------- 	
	}
	class B extends A
	{
	        ---------- 	
	}

2) **Hierarchical Inheritance**: - If there is one super class and more than one subclass then that inheritance is called hierarchical inheritance.

	class A
	{
	        -------- 	
	}
	class B extends A
	{
	        ---------- 	
	}
	class C extends A
	{
	        ---------- 	
	}

3) **Multilevel Inheritance**: - If one class extends is super class and another class extends its subclass then the inheritance would be multilevel inheritance.



```
class A
{
        --------
}
class B extends A
{
        ----------
}
class C extends B
{
        ----------
}
```

➔**Imp Note**: - In Java multiple inheritance is not possible. In multiple inheritance there is one subclass and more than one super class.

```
class A
{
        int i, j;
        void showij()
        {
                System.out.println("i and j: " + i + " " + j);
        }
}
// Create a subclass by extending class A.
class B extends A
{
        int k;
        void showk()
        {
                System.out.println("k: " + k);
        }
        void sum()
        {
                System.out.println("i+j+k: " + (i+j+k));
        }
}
class SimpleInheritance
{
        public static void main(String args[])
        {
                A superOb = new A();
                B subOb = new B();

                superOb.i = 10;
                superOb.j = 20;
                System.out.println("Contents of superOb: ");
                superOb.showij();
```

```
		subOb.i = 7;
		subOb.j = 8;
		subOb.k = 9;
		System.out.println("\nContents of subOb: ");
		subOb.showij();
		subOb.showk();
		System.out.println();
		System.out.println("Sum of i, j and k in subOb:");
		subOb.sum();
	}
}
```

Super Keyword (3 marks)

Whenever a sub class needs to refer to its immediate super class we can use keyword **super**. Super keyword has two uses (i) To call the constructor of super class. (ii) To access the members of the super class that has been hidden by subclass.

Characteristics

- Used to invoke the **constructor of parent** class which has parameters.
- Super must be **first** statement in body of sub class constructor.
- Only the **immediate parent class** data and methods can be accessed.
- If only default constructor is used then no need of super keyword.
- You can call overridden method of parent class using super keyword.

```
class check1
{
		int i,j;
		check1( )
		{
			i = j = 0;
		}
		check1(int a )
		{
			i = a;
			j = 0;
		}
		check1(int a, int b )
		{
			i = a;
			j = b;
		}
		void pudata1()
		{
			System.out.println(“check1”);
			System.out.println(“i = “ + i);
			System.out.println(“j = “ + j);
		}
}
class check2 extends check1
{
	int k;
```

```
        check2( )
        {
                super( );
                k = 0;
        }
        check2(int a )
        {
                super(a);
                k = 0;
        }
        check2(int a, int b )
        {
                super(a, b);
                k = 0;
        }
        check2(int a, int b, int c )
        {
                super(a, b);
                k = c;
        }
        void pudata2()
        {
                putdata1();
                System.out.println(“check2”);
                System.out.println(“k = “ + k);
        }
}
class superchk
{
        public static void main(String args[ ])
        {
                check1 obj1 = new check1(100,200);
                check2 ob1 = new check2( );
                check2 ob2 = new check2(10);
                check2 ob3 = new check2(11,12);
                check2 ob4 = new check2(13,14,15);
                obj1.putdata1();
                ob1.putdata2();
                ob2.putdata2();
                ob3.putdata2();
                ob4.putdata2();
        }
}
```

Program 2

```
class A
{
        int i;
}
// Create a subclass by extending class A.
class B extends A
{
        int i; // this i hides the i in A
        B(int a, int b)
        {
                super.i = a; // i in A
                i = b; // i in B
        }
        void show()
        {
                System.out.println("i in superclass: " + super.i);
                System.out.println("i in subclass: " + i);
        }
}
class UseSuper
{
        public static void main(String args[])
        {
                B subOb = new B(1, 2);
                subOb.show();
        }
}
```

Method Overriding (3 marks)

If you have one method in subclass with same name and same type of signature as in super class then this method is said to be overridden and this process is called method overriding. When an overridden method is called by subclass object the method of sub class is invoked and method defined in super class will be hidden.

```
class A
{
        int i, j;
        A(int a, int b)
        {
                i = a;
                j = b;
        }
        void show()
        {
                System.out.println("i and j: " + i + " " + j);
        }
}
class B extends A
{
        int k;
        B(int a, int b, int c)
        {
                super(a, b);
                k = c;
        }
        void show()
        {
                super.show();
                System.out.println("k: " + k);
        }
}
```

```
class Override
{
        public static void main(String args[])
        {
                B subOb = new B(1, 2, 3);
                subOb.show(); // this calls show() in B
        }
}
```

Abstract Class and Method (3 marks)

In some situation the methods of super class do not defined but they must be define in subclass properly. The super class defines just the structure or general form of method and these methods must be defined in subclass. This can be done by using abstract methods.

Characteristics

- When method is declared as abstract then sub class must override this method.
- The method can be declared as abstract by using abstract keyword.
- The class which has abstract method must be declared as abstract class.

Rules

- The abstract method must be in abstract class.
- The abstract methods do not have body.
- No objects can be created of abstract class.
- Only reference of abstract class can be created.
- The abstract class must be inherited by at least one subclass.
- The subclass of abstract class must override all abstract methods of super class.

➔**Note**: - The methods which are not abstract are called **concrete** methods

```
abstract class A
{
        abstract void callme( );
        void callmetoo( )     //Concrete Methods are also allowed in abstract class.
        {
                System.out.println("This is a concrete method.");
        }
}
class B extends A
{
        void callme()
        {
                System.out.println("B's implementation of callme.");
        }
}
class AbstractDemo
{
        public static void main(String args[ ])
        {
                B b = new B();
                b.callme();
                b.callmetoo();
        }
}
```

Command Line Arguments (3 marks)

You can pass arguments to the main( ) method. This can be done at run time. You can pass information into a program when you run it. The main( ) method takes arguments as string and it stores the arguments in array of string.

```
class cmdLineEx
{
	public static void main(String args[ ])
	{
		for(int i = 0; i< args.length; i++)
			System.out.println("Argument" + i + " : " + args[i]);
	}
}
```

Execute program D:\java cmdLineEx 123 ABC God is Great

Output

```
Argument 0 : 123
Argument 1 : ABC
Argument 2 : God
Argument 3 : is
Argument 4 : Great
```

Dynamic Method Dispatch (3 marks)

Method overriding fulfils the concept of polymorphism. But this is compile time polymorphism not runtime. The call to a method should be resolved at runtime. The dynamic method dispatch is a mechanism by which the call to overridden method is resolved at runtime.

This is done by super class reference. The reference used to call overridden method. The method will be called to which the reference is pointing at runtime.

```
class A
{
	void callme()
	{
		System.out.println("Inside A's callme method");
	}
}
class B extends A
{
	// override callme()
	void callme()
	{
		System.out.println("Inside B's callme method");
	}
}
class C extends A
{
	// override callme()
	void callme()
	{
		System.out.println("Inside C's callme method");
	}
}
```

```
class Dispatch
{
        public static void main(String args[])
        {
                A a = new A(); // object of type A
                B b = new B(); // object of type B
                C c = new C(); // object of type C

                A r; // obtain a reference of type A
                r = a; // r refers to an A object
                r.callme(); // calls A's version of callme

                r = b; // r refers to a B object
                r.callme(); // calls B's version of callme

                r = c; // r refers to a C object
                r.callme(); // calls C's version of callme
        }
}
```

Native Methods (2 marks): - Native methods are those methods which are written in some other language like C or C++. You can use these methods. Native keyword is used to declare a method as native. But native methods are not defined in java program.

Most native methods are written in C language. The mechanism used to integrate C code with a java program is called Java Native Interface (JNI).

Volatile (2 marks): - If variable is modified by volatile keyword, it can be changed unexpectedly by other part of program such as threads. In multithreading there may be more than one threads which share a same variable. Each thread can keep its own private copy of such shared variable. The real (master) copy of the variable is updated various times. In such case the thing only matters is master copy of that variable. So by volatile keyword compiler always use master copy of such variable.

Transient (2 marks): - When instance variable is specified as transient then java runtime system will not write its content to permanent storage area when instance is saved.

Object Class

There is one special class **Object**, defined by Java. All other classes are sub classes of Object. That is, Object is a **super class** of all other classes. This means that a reference variable of type Object can refer to an object of any other class.

Method	Description
void finalize( )	Called before an unused object is recycled.
void notify( )	Resumes execution of a thread waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( )	Waits on another thread of execution.
Class getClass( )	Obtains the class of an object at run time.

Interface

Java does not support multiple inheritance, but it is possible with the help of interface. More than one interface can be implemented in a class.

In interface we have methods but all methods will be abstract. We cannot define method within interface. So the class which implements the interface must override all methods declared in interface.

Syntax:
```
access specifier interface interface_name
{
        Type variable1 = value;
        Type variable2 = value;
        - - - - - -
        Type variableN = value;
        Return_type method1(parameter);
        Return_type method2(parameter);
        - - - - - -
        Return_type methodN(parameter);
}
```

- The **interface** keyword is used to define interface
- We are not allowed to create instance variable inside interface.
- We can only declare the method in the interface but we cannot define method in interface.
- We are not allowed to create instance of an interface.
- But we are allowed to create its reference.
- The access specifier can be public or default
- The variables of interface are implicitly **final & static**
- The variables must be initialized in interface
- All methods are **implicitly abstract** in interface

How to implement Interface?

We can use "implements" keyword to implement interface in our class.

Syntax: **class class_name [extends class] implements interface1, interface2……..**

- A class can extend another class and also implements interface.
- The extends must come before implements keyword
- More than one interface can be separated by comma (,)
- A class must implements all methods of interface and must be declared as public
- If you want only interface in file then file name and interface name must be same

```
interface testme
{
                void test1(int t);
}
class test implements testme
{
                public void test1(int t)
                {
                                System.out.println("Test interface class");
                }
}
class inftest
{
        public static void main(String args[ ] )
        {
                test  t1 = new test( );
                t1.test1(100);
        }
}
```

Interface implementing Interface

As we can inherit one class to another class same way we can inherits one interface to another interface. We can do this by using extends keyword while creating interface.

Syntax: **interface interface_name extends interface1**

```
                {
                        ----------
                }
```

```
interface A
{
        void method1();
        void method2();
}
interface B extends A
{
        void method3();
}
class C implements B
{
        public void method1()
        {
                System.out.println("Method1");
        }
        public void method2()
        {
                System.out.println("Method2");
        }
        public void method3()
        {
                System.out.println("Method3");
        }
}
```

```
class impinterface
{
        public static void main(String args[ ])
        {
                C obj1 = new C ();
                obj1.method1();
                obj2.method2();
                obj3.method3();
        }
}
```

Variables within interface

```
interface constchk
{
        int no = 1;
        int ans = 2;
        int ques = 3;
}
class testvar implements constchk
{
        void printdata()
        {
                System.out.println("no = " + no);
                System.out.println("ans = " + ans);
                //ques = ques + 10; ERROR
                System.out.println("ques  = " + ques);
        }
}
class chkvar
{
        public static void main(String args[ ])
        {
                testvar obj1 = new testvar();
                obj1.printdata( );
        }
}
```

YOR COMPUTER CLASSES

❖ C Language	❖ Oracle
❖ C++	❖ Java
❖ HTML	❖ Adv. Java
❖ Data Structure	❖ C# .NET
❖ COA	❖ ASP.NET
❖ Visual Basic 6.0	❖ SQL Server
❖ PHP	❖ MS-Office

✓ Practical Lab with latest technology

✓ Library with all kinds of reference books & materials

✓ Personal attention on each student

✓ Best material will be provided

YOR COMPUTER CLASSES

Vishal Sir(MCA) **9427732231** **Naman Sir(MCA)** **9408526428**

101 Shiromani Complex, Nr. Balaji Mandir,
Karansinghji Highschool Chowk,
Above YOR Restaurant, Rajkot - 360001

Package

A package is just a **container** or **collection of classes**. It is a directory or folder which contains the source files and its class files. We have to take care about duplicate class name. But using package you can have more than one class of same name but in different packages.

To create a package the "**package**" keyword is used as the first executable statement of your program. After package keyword you have to give package name i.e. directory in which your class files and source files will be stored.

Syntax : **package package_name;**

```
package mypack;
class player
{
        String name;
        int score;
        player(String name, int score)
        {
                this.name = name;
                this.score = score;
        }
        void display()
        {
                System.out.println("Player name " + name);
                System.out.println("Player score " + score);
        }
}
class PackageTest
{
        public static void main(String args[ ])
        {
                player p1 = new player("Sachin", 186);
                player p2 = new player("Dravid", 153);

                p1.display();
                p2.display();
        }
}
```

To compile **D:\java\mypack\javac *.java**

To run **D:\java\java mypack.PackageTest**

Access Control in package

	Private	Default	Protected	Public
Same class	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Same package sub class	x NO	✓ Yes	✓ Yes	✓ Yes
Same package non sub class	x NO	✓ Yes	✓ Yes	✓ Yes
Different Package subclass	x NO	x NO	✓ Yes	✓ Yes
Different Package non subclass	x NO	x NO	x NO	✓ Yes

Import Statement

After creating package you can use its classes into other java files. To do this we have to import that package in which class belongs to. Importing package is similar to including header files in C or C++. Similarly we can import a package to use classes and methods of that package. To import package "import" keyword is used.

Import must be the first executable statement in program but if package and import both are used then package must be the first statement. No executable statement is allowed between package and import.

Syntax : **import package_name;**

Program name:- player.java in package mypack

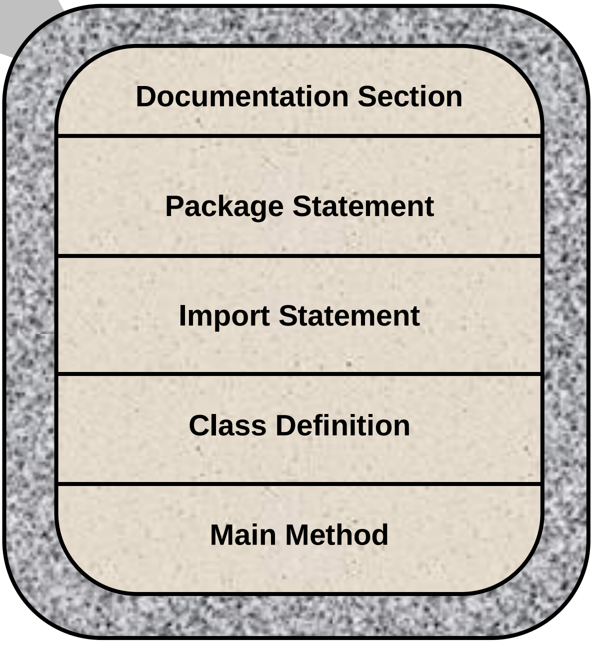
```
package mypack;
public class player
{
        String name;
        int score;
        player(String name, int score)
        {
                this.name = name;
                this.score = score;
        }
        void display()
        {
                System.out.println("Player name " + name);
                System.out.println("Player score " + score);
        }
}
```

Program name:- ImportEx.java in your regular folder.

```
import mypack.*     // OR mypack.player
class ImportEx
{
        public static void main(String args[ ])
        {
                player p1 = new player("Saurav", 180);
                player p2 = new player("Sehwag", 219);

                p1.display();
                p1.display();
        }
}
```

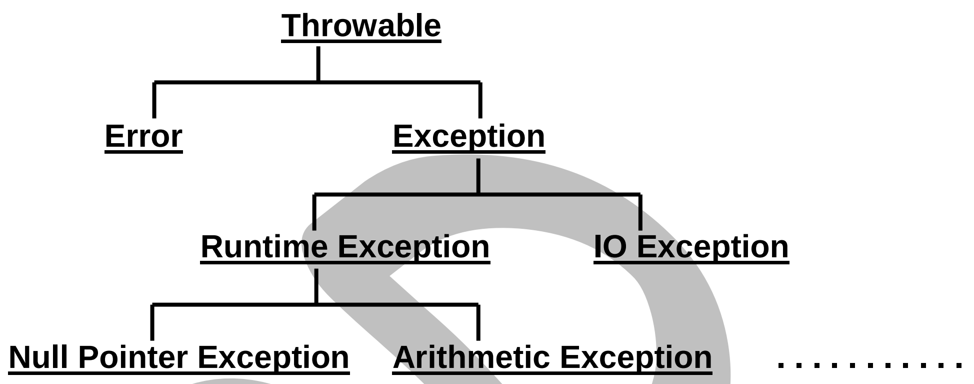
Java API Packages	Includes
java.applet	This package contains classes to implement applet
java.awt	This is abstract window toolkit package. Used for GUI
java.lang	This is language package. It includes String, System, Math class
java.net	It is used for network. Its network package
java.util	This is utility package. It includes Date, Calendar etc
java.io	It is used for Input Output of data. Used for file handling.

Java Program Structure



Exception Handling

Our program can have some errors. The errors are either compile time or run time errors. The compile time errors are syntax errors and can be detected by compiler. But the run time errors are unnatural and abnormal exception. When exception occurs the program terminates suddenly. To handle this exception java supports **Exception Handling**.



➔ In java exception handling is done by using five keywords try, catch, finally, throws and throws.

Try: - Try is a block. The statements that can generate an exception are placed in try block.

Catch: - If an exception is generated in try block then its object is created and thrown. This object is caught in catch block and is handled in catch block.

Finally: - It is a block. The statement written in finally block are must executed whether the exception is generated or not.

- ► A try block is created with at least one catch block.
- ► No executable statement is allowed between try and catch and also between catch and catch.
- ► More than one catch is allowed with single try.

Syntax:

```
try
{
}
catch1( )
{
}
catch2( )
{
}
finally( )
{
}
```

```
class Ex1
{
        public static void main(String args[])
        {
                int d = 0;
                int a = 42 / d;
        }
}
```

Program with Try and Catch

```
class Ex2
{
	public static void main(String args[])
	{
		int d, a;
		try
		{
			d = 0;
			a = 42 / d;
			System.out.println("This will not be printed.");
		}
		catch (ArithmeticException e)
		{
			System.out.println("Division by zero.");    // Error is + e
		}
		System.out.println("After catch statement.");
	}
}
```

Program for Multiple catch

```
class MultiCatch
{
	public static void main(String args[ ])
	{
		try
		{
			int a = args.length;
			System.out.println("a = " + a);
			int b = 42 / a;
			int c[ ] = { 1 };
			c[42] = 99;
		}
		catch(ArithmeticException e)
		{
			System.out.println("Divide by 0: " + e);
		}
		catch(ArrayIndexOutOfBoundsException e)
		{
			System.out.println("Array index out: " + e);
		}
		System.out.println("After try/catch blocks.");
	}
}
```

Program for Nested try

```
class NestTry
{
        public static void main(String args[ ])
        {
                try
                {
                        int a = args.length;
                        int b = 42 / a;
                        System.out.println("a = " + a);
                        try
                        {
                                if(a==1)
                                  a = a/(a-a); // division by zero
                                if(a==2)
                                {
                                        int c[ ] = { 1 };
                                        c[42] = 99; // generate an out-of-bounds exception
                                }
                        }
                        catch(ArrayIndexOutOfBoundsException e)
                        {
                                System.out.println("Array index out-of-bounds: " + e);
                        }
                }
                catch(ArithmeticException e)
                {
                        System.out.println("Divide by 0: " + e);
                }
        }
}
```

Throw: - We have only catching the exceptions which are thrown by Java run-time system. It is also possible to **throw** an exception manually using throw statement.

throw Throwable_Instance

Here Throwable_instance must be an object of type Throwable or of its subclass. Primitive types such as int or char as well as non Throwable classes such as String and Object cannot be used as exceptions.

Inside the standard package **java.lang**, Java defines several exception classes. In java two types of exceptions **Unchecked Exceptions** and **Checked Exception**.

➔In Unchecked exception the compiler does not check to see if a method handles or throws these exceptions.

➔If a method generate an exception that it cannot handled by itself then that type of error is called Checked exception.

Program for throw.

```
class ThrowDemo
{
	static void demoproc()
	{
		try
		{
			throw new NullPointerException("demo");
		}
		catch(NullPointerException e)
		{
			System.out.println("Caught inside demoproc.");
			throw e; // rethrow the exception
		}
	}
	public static void main(String args[])
	{
		try
		{
			demoproc();
		}
		catch(NullPointerException e)
		{
			System.out.println("Recaught: " + e);
		}
	}
}
```

Program for finally

```
class FinallyDemo
{
	static void procA()
	{
		try
		{
			System.out.println("inside procA");
			throw new RuntimeException("demo");
		}
		finally
		{
			System.out.println("procA's finally");
		}
	}
	static void procB()
	{
		try
		{
			System.out.println("inside procB");
			return;
		}
		finally
		{
			System.out.println("procB's finally");
```

```
                }
        }
        public static void main(String args[ ])
        {
                try
                {
                        procA();
                }
                catch (Exception e)
                {
                        System.out.println("Exception caught" + e);
                }
                procB();
        }
}
```

Throws: - If a method is capable of causing an exception that is does not handle, then it must specify this behavior to calling method. So that they can guard themselves against that exception. It can be done by throws keyword. Throws keyword is used with Checked Exceptions.

```
        type method-name (parameter list) throws exception-list
        {
                // Body of method
        }
```

Unchecked Exception	Meaning
ArithmeticException	Arithmetic Error such as divide by zero
NullPointerException	Invalid use of null pointer
NumberFormatException	Invalid conversion from string to number
ArrayIndexOutOfBoundException	Array index is out of bound.

Checked Exception	Meaning
ClassNotFoundException	Class not found
IllegalAccessException	Access to class is denied
NoSuchFieldException	A requested field does not exist
NoSuchMethodException	A requested method does not exist

```
class ThrowsDemo
{
        static void throwOne() throws IllegalAccessException
        {
                System.out.println("Inside throwOne.");
                throw new IllegalAccessException("demo");
        }
        public static void main(String args[ ])
        {
                try
                {
                        throwOne();
                }
                catch (IllegalAccessException e)
                {
                        System.out.println("Caught " + e);
                }
        }
}
```

Throw	**Throws**
throw keyword is used to throw Exception from any method or static block in Java	**throws keyword, used in method declaration, denoted which Exception can possible be thrown by this method.**
throw keyword can be used in switch case in Java	**throws keyword cannot be used anywhere except on method declaration line.**
throw transfers control to caller	**throws is suggest for information and compiler checking.**
It is used for both checked and unchecked exception types	**It is used only for checked exception types**

Multithreaded Programming

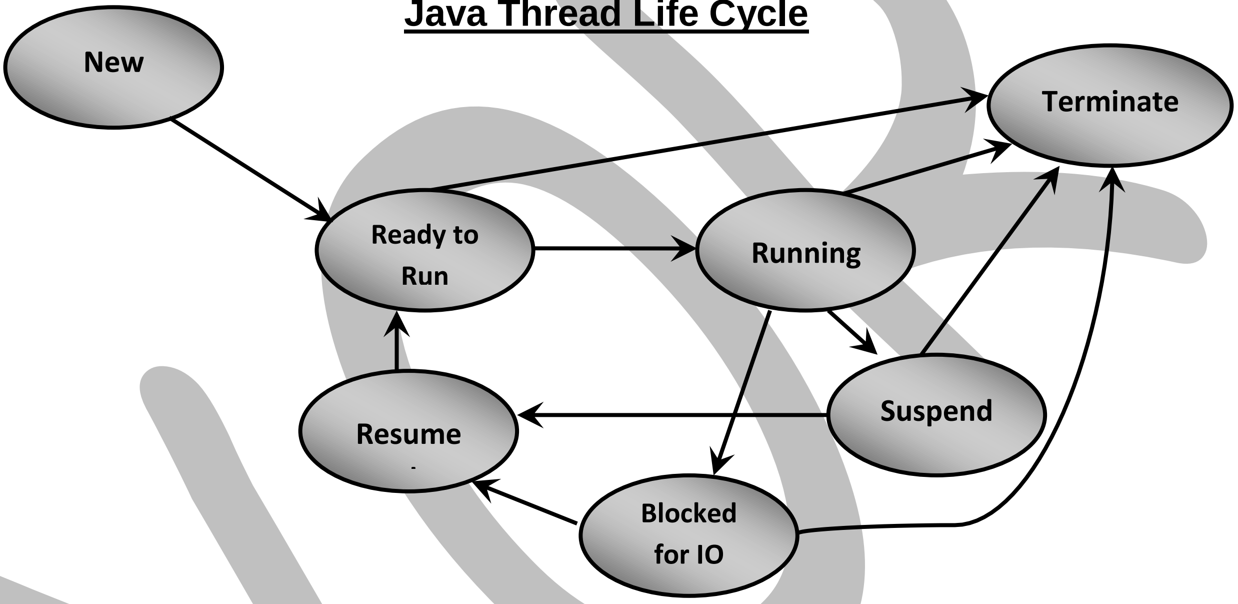
A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a **thread**, and each thread defines a separate path of execution. A thread is simply light weight process or says it is sub process or child process. *Multithreading* is specialized form of *Multitasking*.

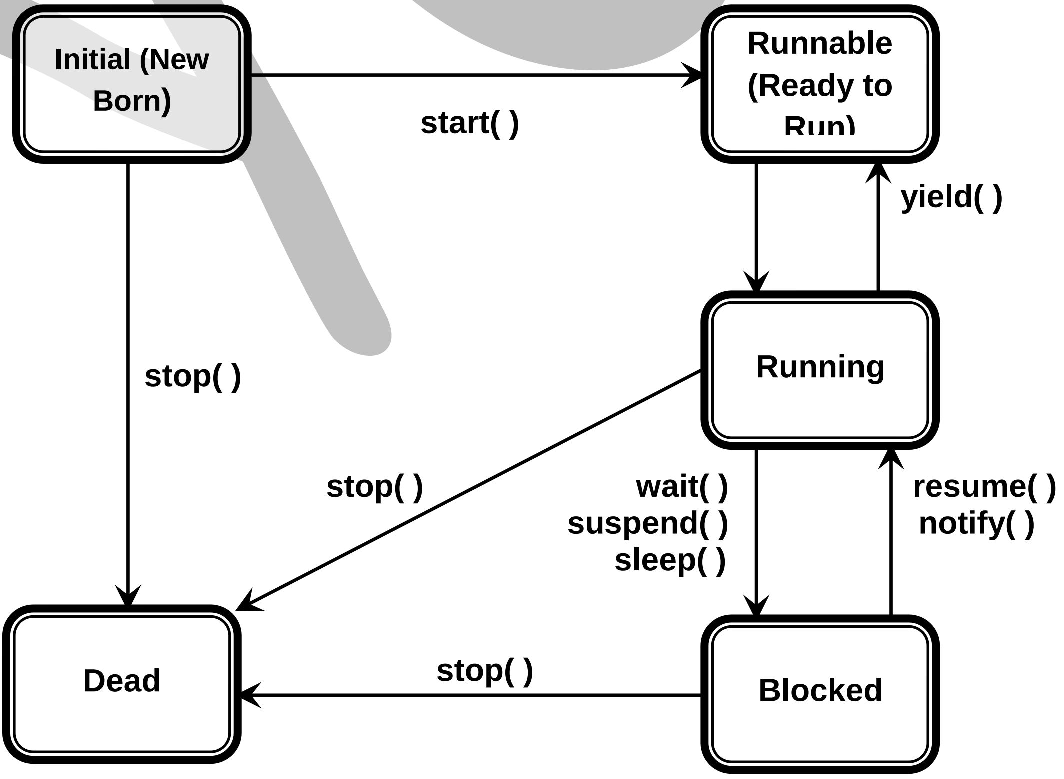
Now a day's most of the operating system is working on multitasking mechanism. There are two types of multitasking: *process based* and *thread based.*

In process based mechanism two or more program runs concurrently on your computer.

In thread based mechanism the single program is divided into units and all units runs concurrently. This single unit is called **thread**. Means a single program can perform two or more task simultaneously.

Java Thread Life Cycle





Java Thread Model

Java supports multiple threads to run concurrently which is known as multithreading. This has more advantage over single threaded system. In single thread system if thread is running it fires some events and waits for signal. So between two events it stops for some time. At that time lots many CPU cycles are wasted and CPU remains free. But with multiple threads if one thread waits for signal mean while other threads runs at CPU. So CPU cannot remain free.

```
class ThreadDemo
{
        public static void main(String args[ ])
        {
                Thread t = Thread.currentThread();
                System.out.println("Current thread: " + t);
                System.out.println("Name of thread is : " + t.getName());
                t.setName("My Thread");
                System.out.println("After name change: " + t);
        }
}
```

Implementing Thread

There are two ways to create thread in Java

1) You can implement **Thread class**
2) You can extend **Runnable interface**

Runnable Interface: The easiest way to create a thread is to create a class that implement the Runnable interface. To implement Runnable, a class need only implement a single method called **run().**

Inside run( ), you will define the code that constitutes the new thread. After you create a class that implements Runnable, you will have to create an object of type Thread from within that class. Now use the constructor of thread class

Thread (Runnable object, String name)

To run the thread you have to use **start( )** method which is declared within Thread. Start( ) executes a call to run( ) method.

Thread Class: Second way to create a thread is to extend thread class. The extending thread class must override the run( ) method which is the entry point for the new thread. It must also call start( ) method to begin execution of the new thread.

Constructors of Thread Class

1) Thread( )
2) Thread(String name)
3) Thread(Runnable object)
4) Thread(Runnable object, String name)

Methods of Thread Class

Method	Meaning
final String getName( )	Obtains thread's name
final int getPriority( )	Obtains thread's priority
final boolean isAlive( )	Determines if thread is still running
final void join( )	Wait for thread to terminate
public void run( )	Entry point for the thread
static void sleep( )	Suspend thread for period of time
void start( )	Start a thread by calling its run method.
final void setName( )	Used to set name of particular thread.
final void setPriority(int)	Used to set priority of particular thread.

Program By extending Thread class

```
class NewThread extends Thread
{
        NewThread()
        {
                super("Demo Thread");
                System.out.println("Child thread: " + this);
        }
        public void run()
        {
                try
                {
                        for(int i = 5; i > 0; i--)
                        {
                                System.out.println("Child Thread: " + i);
                                Thread.sleep(500);
                        }
                }
                catch (InterruptedException e)
                {
                        System.out.println("Child interrupted.");
                }
                System.out.println("Exiting child thread.");
        }
}
class ExtendThread
{
        public static void main(String args[ ])
        {
                NewThread  t = new NewThread(); // create a new thread
                t.start();
                try
                {
```

```
				for(int i = 5; i > 0; i--)
				{
					System.out.println("Main Thread: " + i);
					Thread.sleep(2000);
				}
			}
			catch (InterruptedException e)
			{
				System.out.println("Main thread interrupted.");
			}
			System.out.println("Main thread exiting.");
		}
}
```

Program By implementing Runnable Interface

```
class MyThread implements Runnable
{
		MyThread()
		{
			Thread t = new Thread (this, “My Thread”);
			System.out.println("Child thread: " + this);
			t.start();
		}
		public void run()
		{
			try
			{
				for(int i = 5; i > 0; i--)
				{
					System.out.println("Child Thread: " + i);
					Thread.sleep(500);
				}
			}
			catch (InterruptedException e)
			{
				System.out.println("Child interrupted.");
			}
			System.out.println("Exiting child thread.");
		}
}
class ImpRun
{
		public static void main(String args[ ])
		{
			MyThread  t = new MyThread(); // create a new thread
			try
			{
				for(int i = 5; i > 0; i--)
				{
					System.out.println("Main Thread: " + i);
					Thread.sleep(2000);
				}
```

```
                }
                catch (InterruptedException e)
                {
                        System.out.println("Main thread interrupted.");
                }
                System.out.println("Main thread exiting.");
        }
}
```

Creating Multiple Threads

```
class NewThread implements Runnable
{
        String name; // name of thread
        Thread t;
        NewThread(String threadname)
        {
                this.name = threadname;
                t = new Thread(this, name);
                System.out.println("New thread created is: " + t);
                t.start(); // Start the thread
        }

        public void run()
        {
                try
                {
                        for(int i = 5; i > 0; i--)
                        {
                                System.out.println("Thread" + name + ": " + i);
                                Thread.sleep(1000);
                        }
                }
                catch (InterruptedException e)
                {
                        System.out.println(name + "Interrupted");
                }
                System.out.println("Thread" + name + " exiting.");
        }
}
class MultiThreadDemo
{
        public static void main(String args[ ])
        {
                new NewThread("One"); // start threads
                new NewThread("Two");
                new NewThread("Three");
                try
                {
                        Thread.sleep(10000);
                }
        catch (InterruptedException e)
        {
            System.out.println("Main thread Interrupted");
        }
        System.out.println("Main thread exiting.");
    }
}
```

isAlive( ) and Join( ) methods

The execution of main thread must be terminated after all child threads are terminated. So we must know which thread is executing and which is not. To do this Java supports two ways to determine whether thread has finished or not i.e. **isAlive( ) and join( )**

isAlive( ) : This method is used to know whether a thread is running or not. It returns true if the thread is still running else it returns false.

Join( ) : This method waits until the thread on which it is called terminates. It will not finish its execution until the thread on which it is called completes its execution.

Thread Priorities

Each thread has its priority which is used by the processor for scheduling. This priority defines precedence over other threads. Priority decides which thread would run first and which thread will run after.

There are two methods associated with priority

1) **setPriority( )** : This method is used to set the priority of a thread. The parameter priority is an int value from 1 to 10. 1 is the lowest priority and 10 is the highest priority.
2) **getPriority( )** : This method returns the priority of the invoking thread object.

The thread class has following int constants for thread priorities. These variables are static and final.

NORM_PRIORITY : This is default value which is set to **5**.
MIN_PRIORITY : The minimum priority value is **1.**
MAX_PRIORITY : The maximum priority value is **10.**

```
class MyThread implements Runnable
{
    Thread t;
    String name;
    MyThread(String name)
    {
        this.name = name;
        t = new Thread(this, name);
        t.start( );
    }

    public void run( )
    {
        for(int i = 1; i<=5; i++)
        {
            try
            {
                Thread.sleep(300);
                System.out.println("Thread " + name + ":" + i);
            }
            catch(InterruptedException e)
            {
                System.out.println("Thread " + name + "interrupted");
            }
        } // for loop over
    } // run method over
} // class over

class ThreadPrio
{
    public static void main(String args[ ] )
    {
        MyThread t1 = new MyThread("Slow");
        MyThread t2 = new MyThread("Medium ");
        MyThread t3 = new MyThread("Fast ");

        System.out.println("Default Thread Priorities are " );
        System.out.println("Priority of Thread  " + t1.t.getName +"="+t1.t.getPriority());
        System.out.println("Priority of Thread  " + t2.t.getName +"="+t2.t.getPriority());
        System.out.println("Priority of Thread  " + t3.t.getName +"="+t3.t.getPriority());

        t1.t.setPriority(Thread.MIN_PRIORITY);
        t2.t.setPriority(Thread.NORM_PRIORITY);
        t3.t.setPriority(Thread.MAX_PRIORITY);

        System.out.println("After changing Priority now Thread Priorities are " );
        System.out.println("Priority of Thread  " + t1.t.getName +"="+t1.t.getPriority());
        System.out.println("Priority of Thread  " + t2.t.getName +"="+t2.t.getPriority());
        System.out.println("Priority of Thread  " + t3.t.getName +"="+t3.t.getPriority());
    }
}
```

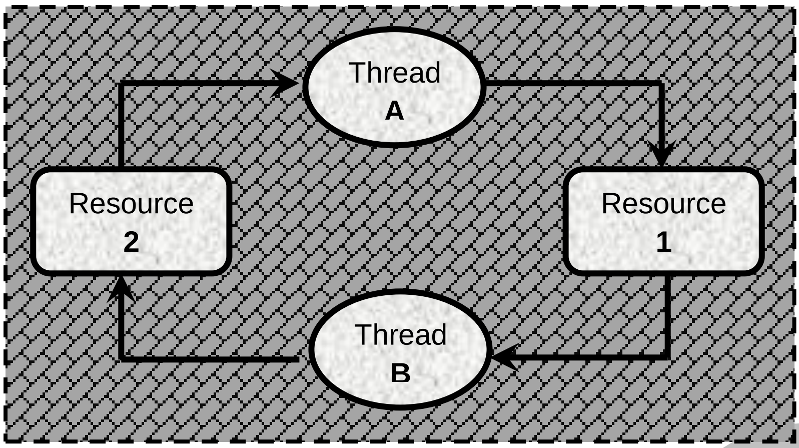
Synchronization

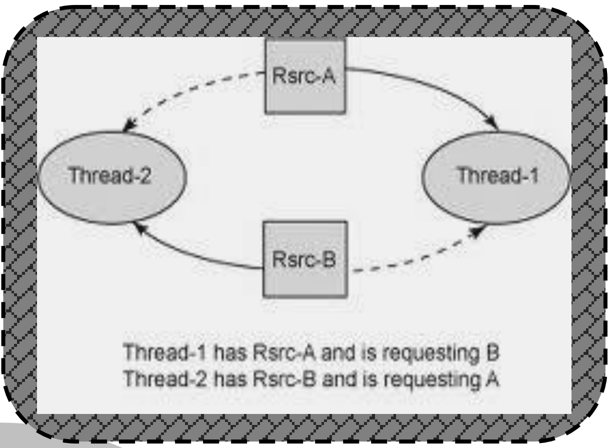
When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only **one thread at a time**. The process by which this is achieved is called **synchronization**. Key to synchronization is the concept of the **monitor** (also called a semaphore).

A **monitor** is an object that is used as a **mutually exclusive lock, or mutex**. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other threads are said to be **waiting for the monitor**. A thread that owns a monitor can re-enter the same monitor if it so desires.

Deadlock

When two threads have circular dependency on objects a major problem arises. This type of error is known as deadlock. In deadlock, one thread is waiting for other thread and other thread is waiting for the first thread. Thus both threads go in waiting (blocked) state and cannot continue their execution.




Rsrc-A
Thread-2
Thread-1
Rsrc-B
Thread-1 has Rsrc-A and is requesting B
Thread-2 has Rsrc-B and is requesting A

Inter-Thread Communication

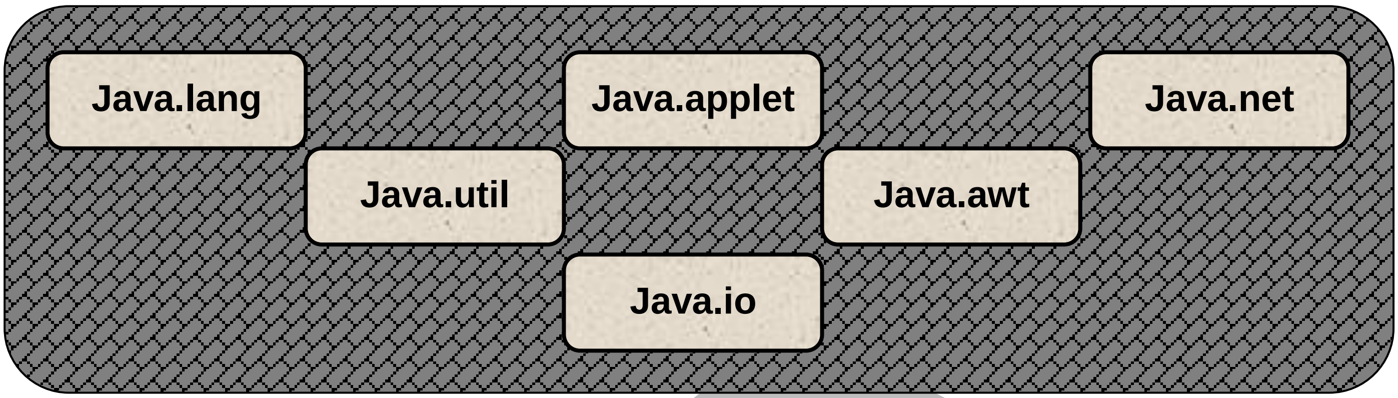
Java includes an elegant inter process communication mechanism via the **wait( ), notify( ),** and **notifyAll( )** methods. These methods are implemented as final methods in **Object class**, so all classes have them.

final void wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

final void notify( ) wakes up the first thread that called wait( ) on the same object.

final void notifyAll( ) wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

Packages Available in Java



Java.lang

This is language package. This is default package in Java. You can directly use all its classes without importing this package. It contains the following classes.

Object Class	Math Class	String Class	StringBuilder Class
Thread Class	Throwable Class	Wrapper Class	

It also contains interfaces. It contains **Runnable** interface

1) Object Class

There is one special class **Object**, defined by Java. All other classes are sub classes of Object. That is, Object is a **super class** of all other classes. This means that a reference variable of type Object can refer to an object of any other class.

Method	Description
void finalize( )	Called before an unused object is recycled.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( )	Waits on another thread of execution.
Class getClass( )	Obtains the class of an object at run time.
boolean equals(Object object)	Returns true if the invoking object is equivalent to object.

2) Math Class

This class contains all methods related to mathematical operations.

Method	Description
float abs(float f) **int abs(int i)**	Returns the absolute value of the argument.
double ceil(double d)	Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
double floor(double d)	Returns the largest integer that is less than or equal to the argument. Returned as a double.
long round(double d) **int round(float f)**	Returns the closest long or int, as indicated by the method's return type, to the argument.
float min(float arg1, float arg2) **int min(int arg1, int arg2)**	Returns the smaller of the two arguments.
float max(float arg1, float arg2) **int max(int arg1, int arg2)**	Returns the larger of the two arguments.
double exp(double d)	Returns the base of the natural logarithms, e, to the power of the argument.(e = 2.71 appr)
double pow(double base, double exponent)	Returns the value of the first argument raised to the power of the second argument.
double sqrt(double d)	Returns the square root of the argument

3) String Class

Java implements strings as objects of type String. When you are creating String object you are creating a string that cannot be changed. Once a String object has been created you cannot change the characters that comprise that string. Each time you need to change the String, a new string object is created that contains the modifications. The original string left unchanged.

If you wish to change the original string then Java provides two other String classes: **StringBuffer** and **StringBuilder**.

Constructor of String Class

1) **String( )**
2) **String(char charArray[ ])**
3) **String(char charArray[ ], int start, int length)**

e.g.

```
char name[ ]  = {'J','a','v','a'};
String s1 = new String(name);
```

String s2 = "Hello"   String s3 = "World";

Method	Description	Example
int length( )	Returns the length of this string.	int i = s1.length( ); **// O/p 3**
char charAt(int index)	Returns the char value at the specified index.	ch = s1.charAt(2); **// O/p 'v'**
void getChars(int Start, int End, char target[ ], int targetStart)	Copies characters from this string into the destination character array	s2.getChars(2,4,temp,0) **// O/p llo**
boolean equals(Object str)	It is used to compare two strings for equality and returns Boolean value	s2.equals(s3) **// O/p false**
boolean equalsIgnoreCase(String str)	It is used to compare two strings for equality ignoring case.	s2.equalsIgnoreCase("HELLO"); **// O/p true**
boolean startsWith(String str)	The startsWith( ) method determines whether a given String begins with a specified string.	"Foobar".startsWith("Foo") **// O/p true**
boolean endsWith(String str)	endsWith( ) determines whether the String in question ends with a specified string	"Foobar".endsWith("bar") **// O/p true**
int compareTo(String str)	It compares two strings and returns zero if both are same return greater than zero if first string is big and return less than zero if first string is small	s2.compareTo("Hi") **// O/p -4**
String concat(String str)	Concatenates the specified string to the end of this string	String s4 = s2.concat(s3); **//O/p Hello World**
String replace(char original, char replace)	The replace( ) method replaces all occurrences of one character in the invoking string with another character.	String s = s2.replace('l', 'w'); **//O/p Hewwo**
String trim( )	The trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.	String s = " Hello World ".trim(); **// O/p "Hello World"**
String substring (int Index)	Returns a new string that is a substring of this string. The substring begins with the character at the specified index	"unhappy".substring(2) **// O/p "happy"** "smiles".substring(1, 4) **// O/p "mile"**

4) StringBuffer Class

StringBuffer is a peer class of String that provides much of the functionality of strings. With StringBuffer class you can create writable and growable strings. With StringBuffer class you can enter substring in middle or append at end of the string. The StringBuffer class allocates room of 16 characters when no buffer size is explicitly allocted..

Method	Description	Example
void setLength(int len)	Used to set the length of the string.	s1.setLength(5)
void setCharAt(int index,char ch)	Used to set character at specified index	s2.setCharAt(1, 'i'); **//O/p Hillo**
append(String str)	It is used to append a string at the end of calling string.	s2.append("World") **//O/p "Hello World"**

insert(int index, String str)	It is used to insert a string or character at particular index position.	S1 = "I Java"
		S1.insert(2,"like ")
		//O/p I like Java

reverse( )	It is used to reverse the characters of string.	S1 = "Java" S1.reverse( ) **// O/p avaJ**
delete(int start, int end)	This method deletes sequence of characters from a string.	S1 = "This is Java" S1.delete(4,6) **// O/p This Java**

5) Wrapper Class

There is a wrapper class for every primitive data type in Java. There are two main reasons for wrapper classes (i) To provide a mechanism to "wrap" primitive values in an object so that the primitives can be included in activities reserved for objects, like as being added to Collections. (ii) To provide an assortment of utility functions for primitives. Most of these functions are related to various conversions: converting primitives to and from String.

→ The Byte, Short, Integer, Long, Float, and Double wrapper classes are all subclasses of the Number class.

→ All the wrapper classes are declared final.

→Wrapper classes convert numeric strings into numeric values.

→The way to store primitive data in an object.

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
float	Float
double	Double
long	Long
char	Character
boolean	Boolean

xxxValue() method

When you need to convert the value of a wrapped numeric to a primitive, use one of the many xxxValue() methods. All of the methods in this family are no-arguments methods. Each of the six numeric wrapper classes has six methods, so that any numeric wrapper can be converted to any primitive numeric type.

```
Integer iob = new Integer(100);    // Auto Boxing
int i = iob.intValue( );           // Auto Unboxing
```

Now there will be value 100 in i variable and in iob object.

AutoBoxing is the automatic conversion that the Java compiler makes between the primitive types to their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on

AutoUnboxing is the automatic conversion that the Java compiler makes between the object wrapper class to their corresponding primitive type. For example, converting an Integer to an int.

xxx Parse xxx(String) method

If you do not need to store a value in a wrapper but just want to perform a quick operation on it, such as converting the type, you can do it by using an appropriate static method of the appropriate wrapper class. For example, all the wrapper classes except Character offer a static method that has the following signature:

static <type> parse<Type>(String s)

The <type> may be any of the primitive types except char (byte, short, int, long, float, double, or boolean), and the <Type> is the same as <type> with the first letter uppercased; for example:

static int parseInt (String s)

Each of these methods parses the string passed in as a parameter and returns the corresponding primitive type.

For example, consider the following code:

```
String s = "123";
int i = Integer.parseInt(s);
```

The second line will assign an int value of 123 to the int variable i.

Wrapper classes also has some constant variables like

MIN_VALUE - Returns the minimum value that the variable can hold.
MAX_VALUE - Returns the maximum value that the variable can hold.
TYPE – It returns Data type name for the class.

Java.util

It is utility package. It provides wide range of classes like to manage date and time, calendar etc to handle formatted data. It provides Java's most powerful subsystem: *The Collection Framework.*
A *collection* is an object that represents a group of objects (such as the classic Vector class). It is used to manipulate group of data as a single unit.

Date Class	Random Class	Calendar Class	Gregorian Class
Vector Class	Stack Class	Hash Table Class	Stream Tokenizer

The Collection Framework is made up of a set of **interfaces** for working with the groups of objects.

Collection	Enumeration	Map	Queue
List	Set	RandomAccess	

1) Date Class

The Date class creates an object that represents the current date and time. Its constructors are

(1) Date( )
(2) Date(long millisec)

The first constructor creates the object that has the current date and time. The second constructor creates the object that has date and time after number of milliseconds from the **Epoch(Midnight, 1st January, 1970)** specified by the parameter.

Method	Description
long getTime( )	It returns number of milliseconds passed after Epoch.
boolean before(Date obj)	Returns true if the invoking date object is earlier then parameter object
boolean after(Date obj)	Returns true if the invoking date object is after then parameter object
boolean equals(Date obj)	Returns true if both objects have same date and time else returns false.
int compareTo(Date obj)	Returns zero if both objects are same, returns positive if invoking object is later and returns negative if invoking object is earlier.

```
Import java.uitl.*;
class DateEx
{
	public static void main(String args[ ])
	{
		Date today = new Date( );
		Date epoch = new Date(0);
		Date d = new Date(24*60*60*1000);
		System.out.println("Today's date is" + date);
		System.out.println("Date of Epoch is" + epoch);
		System.out.println("1 day after epoch is" + d);

		long ms = new d.getTime();
		System.out.println("Number of milliseconds " + ms);
		System.out.println("Today is before Epoch?" + today.before(epoch));
		System.out.println("Today is after Epoch?" + today.after(epoch));
		System.out.println("Today is equals Epoch?" + today.equals(epoch));
		System.out.println("Today compare to Epoch?" + today.compareTo(epoch));
	}
}
```

2) Calendar Class

This class deals with date and time related functions. This class is abstract class and it has no constructors.

Method	Description
Calendar getInstance( )	It returns an object of default local and time zone. This method is used to create object of calendar class.
final int get(int Field)	Returns the value of the component specified by field
final int set(int Field), int value)	Sets the value of the component specified by field
void add(int Field, int value)	If adds value specified to calendar component specified by field.
final void clear( )	It sets all components to zero.

Component	Meaning		Component	Meaning
DATE	Specifies date		**DAY_OF_MONTH**	Specifies day of month
MONTH	Specifies month		**DAY_OF_YEAR**	Specifies day of year
YEAR	Specifies year		**HOUR**	Returns hour of time
DAY_OF_WEEK	Specifies day of week		**MINUTE**	Returns min of time.

```
class CalendarEx
{
        public static void main(String args[ ])
        {
                Calendar cal = Calendar.getInstance( );
                System.out.println("The date is " + cal.get(Calendar.DATE) + "/");
                System.out.println(cal.get(Calendar.MONTH) + 1 + "/" + cal.get(Calendar.YEAR) );

                cal.set(Calender.MONTH,6);
                cal.set(Calendar.DATE, 10);
                System.out.println("NOW date is " + cal.get(Calendar.DATE) + "/");
                System.out.println(cal.get(Calendar.MONTH) + 1 + "/" + cal.get(Calendar.YEAR) );

                cal.add(Calender.MONTH, 2);
                System.out.println("After adding month is " + cal.get(Calendar.MONTH) + 1);
                cal.clear(Calender.MONTH); // clearing month
        }
}
```

3) Gregorian Calendar Class

This class is subclass of Calendar class. This class has four constructors.

- **GregorianCalendar( )**
- **GregorianCalendar(int year, int month, int date)**
- **GregorianCalendar(int year, int month, int date, int hour, int min)**
- **GregorianCalendar(int year, int month, int date, int hour, int min, int sec)**

4) Random Class

This class generates numbers randomly. The process of generating random numbers is done by some algorithm. This algorithm is based to some **seed** value also known as **starting value**. Thus these numbers are also known as **pseudorandom numbers.**
It has following constructors

- **Random( )**
- **Random(long seedvalue);**

5) Vector Class

This class creates dynamic array. The simple array you create is of a fixed size, but you can create dynamic array using the Vector class. Vector automatically grows when needed. In simple array you can store elements of same data types only, but in vector you can have different data type. It has following constructors.

- **Vector( )**
- **Vector(int initialSize)**
- **Vector(int initialSize, int incrValue)**

The first constructor creates a vector of **initial size of 10** elements. While in second constructor you can define size of vector. But the **vector size if doubled** when it reaches the initial size specified. To avoid this thing use third constructor which also **specifies increment value** when vector reached its size?

Method	Description
final void addElement(Object value)	It is used to add object to the vector.
final int capacity( )	Returns the capacity of the vector.
final int size( )	Returns the total element that vector currently holding.
final void insertElementAt(Object value, ind index)	Used to insert element at specified index
final void removeElementAt(int index)	Used to remove element from specified index.
final void removeElement(Object value)	Used to remove a particular element by specifying its value.
final void removeAllElements( )	Used to remove all elements from vector.
final int indexOf(Object value)	It returns index of element specified by the value if found else return -1.

```
import java.util.*;
class VectorMethod
{
        public static void main(String args[ ])
        {
                Vector v1 = new Vector( );          // Default size 10 elements
                Vector v2 = new Vector(5);         // Initial size set to 5
                Vector v3 = new Vector(5,2);       // Initial size 5 and increment value 2

                v3.addElement(“Java”);
                v3.addElement(“Computers”);
                v3.addElement(“123”);
                v3.addElement(“92.5”);
                System.out.println(“Vector V3 is” + v3);

                System.out.println(“Size of V3 is” + v3.size( ));
                System.out.println(“Capacity  of V3 is” + v3.capacity( ));

                v3.insertElementAt(“Happy”, 2);
                v3.removeElementAt(1);
                v3.removeElement(“Happy”);
                System.out.println(“Now after inserting and removing Vector V3 is” + v3);
        }
}
```

6) Stack Class

The stack class extends the vector class and it provides all methods of vector class. Stack follows LIFO (Last In First Out) System. This class has only **one default constructor.**

Method	Description
Object push(Object element)	This method is used to insert element in stack.
Object pop( )	It removes and returns the element at the top of the stack.
Oject peek( )	It returns the object from the top of stack but does not delete.
int search(Object element)	It returns position of element in stack if found else -1
boolean empty( )	Returns true if stack is empty.

Applet

There are two types of programs can be developed in java **(i) Application Program (ii) Applet Program.**

An **applet is a GUI (Graphical User Interface**) based Java program that runs on the Internet by a Java powered browser. An applet is secure as it does not let your computer to be infected by a virus or any other malicious program.

Apple is secure because Java applets are executed in a ***sandbox*** by most web browsers, preventing them from accessing local data like clipboard or file system

A Simple Applet Program

➔ **Two way to make applet program**

(1)

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("Hello World",10,25);
        }
}
```

➔ Compile above code in command prompt **javac MyApplet.java**

Make New HTML File RunApp.html

```
<html>
        <head>
                <title>"Hello"</title>
        </head>
        <body>
                <applet code = "MyApplet.class" height= 300 width = 200>
                </applet>
        <body>
<html>
```

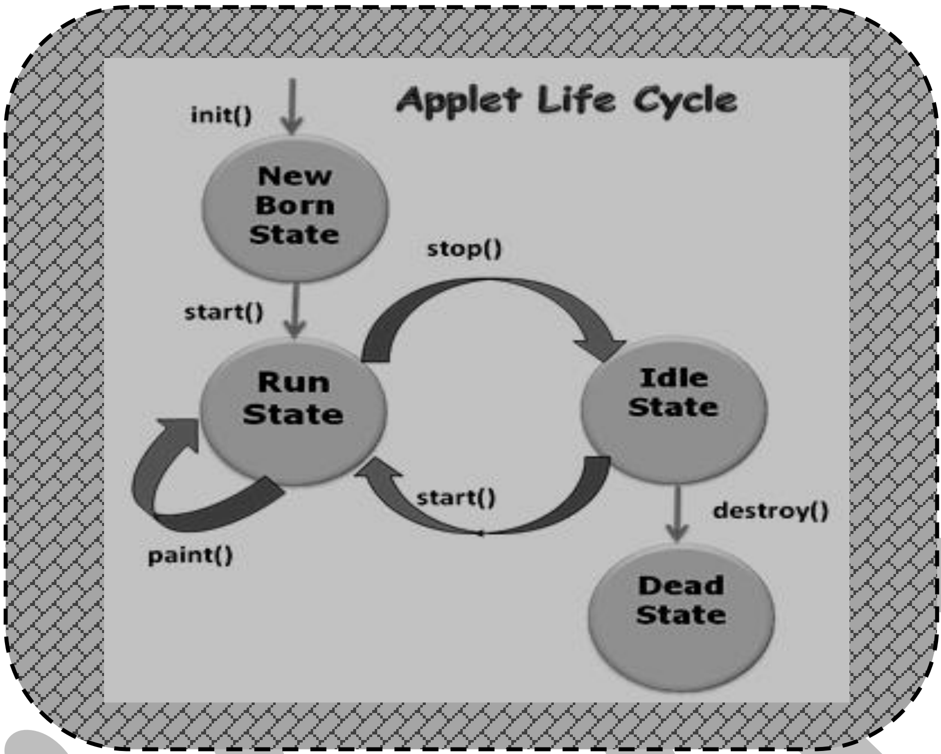
Now either **run** the above HTML file in your **browser** OR
Run the appletviewer command in command prompt **appletviewer RunApp.html**

(2)

```
import java.awt.*;
import java.applet.*;
/*
        <applet code = "MyApplet.class" height= 300 width = 200>
        </applet>
*/
public class MyApplet extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("Hello World",10,25);
        }
}
```

➔ Compile above code in command prompt **------ javac MyApplet.java**
➔ To run the above code in command prompt **---- appletviewer MyApplet.java**

Life Cycle Of Applet (5 marks)



Each applet has a life cycle. So every time an applet is initialised it goes through four states and finally after its execution apple is destroyed. The applet calls methods **init( ), start( ), stop( ) and destroy( ).** These methods are of applet class. In addition to these four methods the **paint( )** method of Component class of java.awt package.

(1) The init( ) method: This method is the first method to be called by applet. In this method initialisation process is done. This method is called only once.

(2) The start( ) method : After init( ) method the start( ) method is called. This method starts the applet. This method can be called more than once since it needs to restart each time it is stopped

(3) The stop( ) method : The stop( ) method is called when you minimize the applet window or leave the applet page and go to another page. After stopping an applet it can e restarted by start( ) method.

(4) The destroy( ) method : This method is called when your applet us to be terminated and needs to removed from memory. This method is also called once.

(5) The paint( ) method : This method is of Component class in java.awt package. This method is called when the applet execution starts or contents is to be redrawn. This method takes parameter of type Graphics class.

```
import java.applet.*;
import java.awt.*;
/*
<applet code = AppletMethod height = 100 width = 400>
</applet>
/*
public class AppletMethod extends Applet
{
        String msg;
        public void init( )
        {
                msg += “init( ) called”;
        }
        public void start( )
        {
                msg += “start( ) called”;
        }
        public void stop( )
        {
                msg += “stop( ) called”;
        }
        public void destroy( )
        {
                msg += “destroy( ) called”;
        }
        public void paint(Graphics g )
        {
                g.drawString(msg,20,30);
        }
}
```

Some Other Methods of Applet class

Method	Description
URL getCodeBase( )	It returns URL(path) of directory of the applet file
URL getDocumentBase( )	It returns URL(path) of HTML document that executes the applet.
String getParameter (String paramName)	It returns the parameter value of specified parameter.
boolean isActive( )	It returns true if applet is started else false
void showStatus (String status)	It displays the status in status bar.

Graphics Class

This class is in java.awt package and it contains several methods that can help to perform graphics related operations in the applet. These methods include drawing text, line and many other shapes.

1. drawLine(int x1, int y1, int x2, int y2)
Draw a line between points (x1,y1) and (x2,y2)

2. drawRect (int x, int y, int width, int height)
Draws a rectangle, (x,y) are the coordinates of the top left corner, the bottom right corner will be at (x+width,y+height).

3. fillRect (int x, int y, int width, int height)
Draws filled rectangle, (x,y) are the coordinates of the top left corner, the bottom right corner will be at (x+width,y+height).

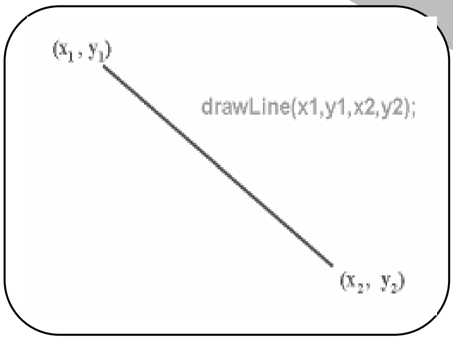
4. drawOval (int x, int y, int width, int height)
Draws an oval bounded by the rectangle specified by these parameters.

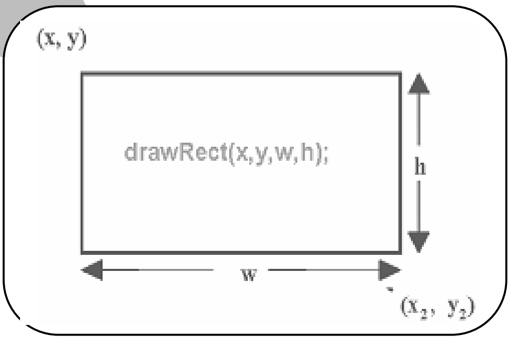
5. fillOval (intx, int y, int width, int height)
Draws filled oval bounded by the rectangle specified by these parameters.

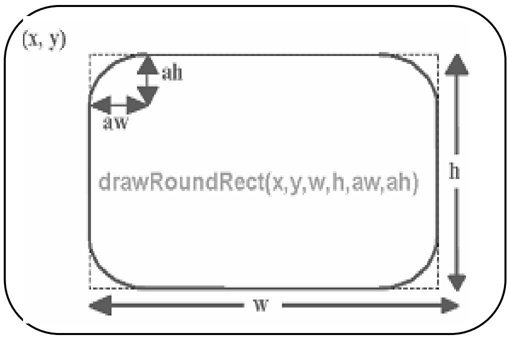
6. drawArc(int x,int y,int width, int height, int startAngle, int arcAngle)
An arc is formed by drawing an oval between a start angle and a finish angle. The start angle is measured from the positive x-axis and is expressed in degrees. The arc angle is expressed in degrees from the start angle. Angles extend from the center of the bounds box.

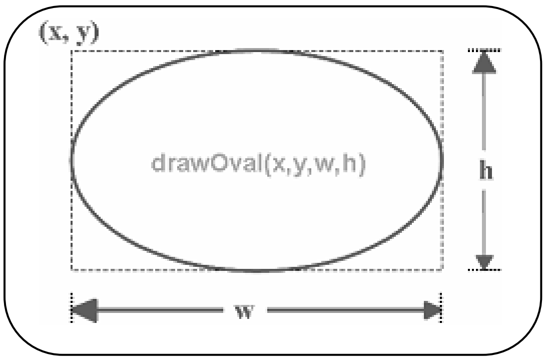
7. drawPolygon (int [ ] x, int [ ] y, int N)
Draws lines connecting the points given by the x and y arrays. Connects the last point to the first if they are not already the same point.

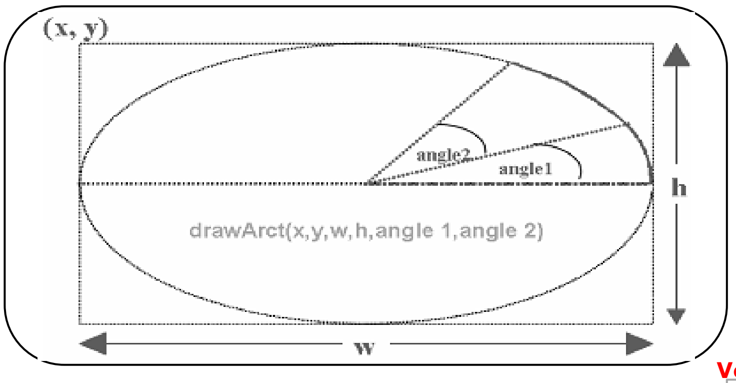
8. fillPolygon (int [ ] x, int [ ] y, int N)
Draws lines connecting the points given by the x and y arrays. Connects the last point to the first if they are not already the same point and also fills color.











```
import java.applet.*;
import java.awt.*;
/*  <applet code = ShapesEx width = 400 height = 500>
</applet> */
public class ShapesEx extends Applet
{
        public void paint(Graphics g)
        {
                g.drawString("Welcome to shape example of applet",10,20);
                g.drawString("This is line",10,40);
                g.drawLine(120,40,220,40);
                g.drawString("This is Rectangle",10,80);
                g.drawRec(120,60,100,50);
                g.drawString("This is Filled Rectangle",10,40);
                g.fillRec(120,120,100,50);
                g.drawString("This is an ellipse",10,200);
                g.drawOval(120,180,100,50);
                g.drawString("This is filled oval",10,280);
                g.fillOval(120,240,75,75);
                g.drawString("This is Arc",10,350);
                g.drawArc(120,330,75,75,0,180);
                int x[ ] = {170,220,220,120,120}
                int y[ ] = {380,430,480,480,430};
                g.drawString("This is Pentagon",10,430);
                g.drawPolygon(x,y,5);
        }
}
```

The Color Class

This class is in java.awt package and it used to deal with colors. The color class lets you make any color you want. Its constructors are

- **Color(int red, int green, int blue)** **// The value must be between 0 to 255**
- **Color(int RGBValue)**

The Color class has some constants such as black, blue, red, green, white, magenta, cyan, yellow etc. You can use these constants like Color.red, Color.yellow etc.

To set the Graphics color there is a method of Graphics class which sets the color of foreground
void setColor(Color object)

The Font Class

This class allows you to specify the size and type of the font displayed in the applet window. The font class is in java.awt package. To create a specific font object following constructor is used.

Font(String name, int style, int size)

The name specify the name of the font e.g. Arial, Verdana, Monotype Corsiva etc

The style specifies the font style. It can be Font.BOLD, Font.ITALIC or Font.PLAIN. If you want your font to be both bold and italic then use Font.BOLD | Font.ITALIC.

The size specify the size of font in points.

After creating a font object you have to set it by the **setFont( )** method.

```
import java.applet.*;
import java.awt.*;
/*
<applet code = “FontColorEx.class” width = 500 height = 100>
</applet>
*/
public class FontColorEx extends Applet
{
        public void paint(Graphics g)
        {
                Font f1= new Font(“Monotype Corsiva”,Font.BOLD,26);
                g.setFont(f1);
                Color c1 = new Color(255,0,0);
                g.setColor(c1);
                g.drawString(“This is Monotype Corsiva in red color”,10,25);
                Font f2= new Font(“Arial”,Font.ITALIC,48);
                g.setFont(f2);
                Color c2 = new Color(0,255,0);
                g.setColor(c2);
                g.drawString(“This is Arial in green color”,10,55);
                Font f3= new Font(“Times New Roman”,Font.BOLD | Font.ITALIC,48);
                g.setFont(f2);
                g.setColor(Color.Blue);
                g.drawString(“This is Times New Roman in Blue color Bold and Italic”,10,85);
        }
}
```

<u>APPLET tag (5 marks)</u>

```
   <APPLET
             [CODEBASE = codebaseURL]
             CODE = appletFile
             [ALT = alternateText]
             [NAME = appletInstanceName]
             WIDTH = pixels  HEIGHT = pixels
             [ALIGN = alignment]
             [VSPACE = pixels]
             [HSPACE = pixels]
   >
  <PARAM NAME = appletAttribute1 VALUE = value>
  <PARAM NAME = appletAttribute2 VALUE = value>
  <PARAM NAME = appletAttributeN VALUE = value>
  </APPLET>
```

1) CODEBASE = codebaseURL

This OPTIONAL attribute specifies the base URL of the applet--the directory that contains the applet's code. If this attribute is not specified, then the document's URL is used.

2) CODE = appletFile

This REQUIRED attribute gives the name of the file that contains the applet's compiled Applet subclass. This file is relative to the base URL of the applet. It cannot be absolute. One of CODE or OBJECT must be present. The value appletFile can be of the form classname.class or of the form packagename.classname.class..

3) ALT = alternateText

This OPTIONAL attribute specifies any text that should be displayed if the browser understands the APPLET tag but can't run Java applets.

4) NAME = appletInstanceName

This OPTIONAL attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

5) WIDTH = pixels HEIGHT = pixels

These REQUIRED attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

6) ALIGN = alignment

This OPTIONAL attribute specifies the alignment of the applet. The possible values of this attribute are: *left, right, top, middle, baseline, bottom.*

7) VSPACE = pixels HSPACE = pixels

These OPTIONAL attributes specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE). They're treated the same way as the IMG tag's VSPACE and HSPACE attributes.

```
<PARAM NAME = appletAttribute1 VALUE = value>

<PARAM NAME = appletAttribute2 VALUE = value> . . .
```

This tag is the only way to specify an applet-specific attribute. Applets access their attributes with the getParameter() method.

Passing Parameters to applet OR PARAM tag (3 marks)

```
import java.applet.*;
import java.awt.*;
public class appletParameter extends Applet
{
        public void paint(Graphics g)
        {
                String msg = getParameter(“msg”);
                String font = getParameter(“font”);
                int size = Integer.parseInt(getParameter(“size”));
                Font f = new Font(font,Font.BOLD,size);
                g.setFont(f);
                g.setColor(Color.green);
                g.drawString(msg,20,25);
         }
}

<HTML>
        <HEAD>
                <TITLE>Passing Parameter in Java Applet</TITLE>
        </HEAD>
        <BODY>
                <APPLET code="appletParameter.class" width="800" height="100">
                   <PARAM  name = "msg " value = "Welcome in Passing parameter in java applet.">
                   <PARAM  name = "font " value = Monotype Corsiva>
                   <PARAM  name = "size " value = 28>
                </APPLET>
        </BODY>
</HTML>
```

YOR COMPUTER CLASSES

❖ C Language	❖ Oracle
❖ C++	❖ Java
❖ HTML	❖ Adv. Java
❖ Data Structure	❖ C# .NET
❖ COA	❖ ASP.NET
❖ Visual Basic 6.0	❖ SQL Server
❖ PHP	❖ MS-Office
✓ Practical Lab with latest technology	
✓ Library with all kinds of reference books & materials	
✓ Personal attention on each student	
✓ Best material will be provided	

YOR COMPUTER CLASSES

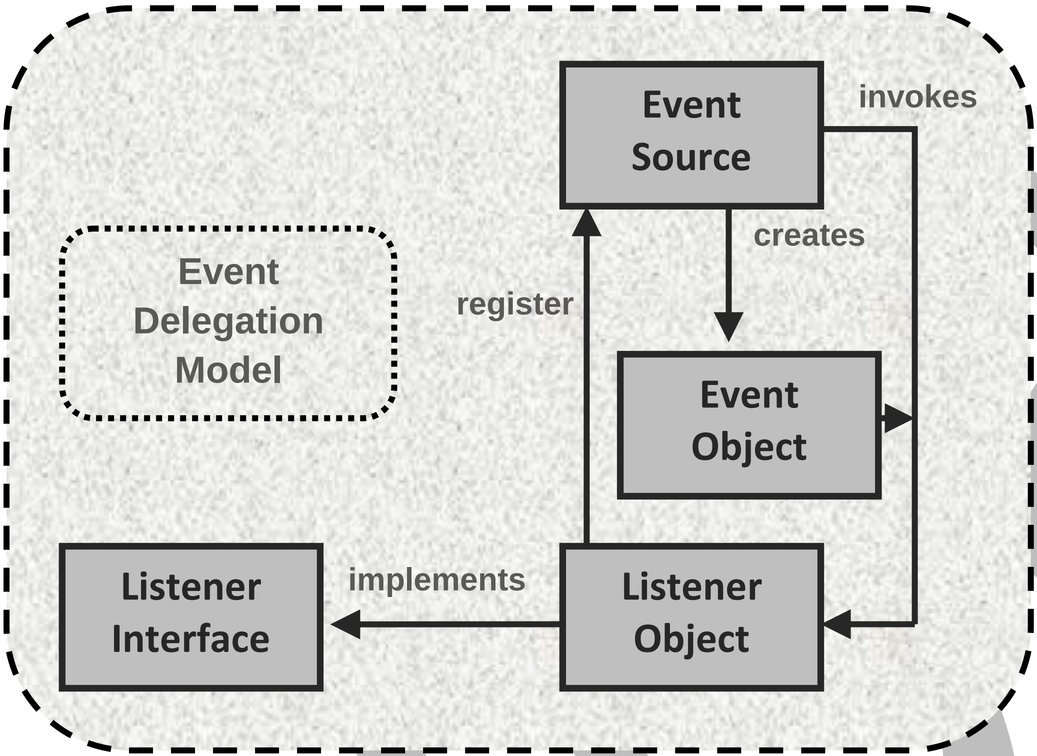
Vishal Sir(MCA) 9427732231	Naman Sir(MCA) 9408526428

101 Shiromani Complex, Nr. Balaji Mandir,
Karansinghji Highschool Chowk,
Above YOR Restaurant, Rajkot - 360001

Event Handling

It is very important feature of applet programming. The classes and interfaces related to event handling are contained in the java.awt.event packages. The events are handled by a model known as Event Delegation Model.

Event Delegation Model



The event delegation model is a mechanism that handles the generating and processing of events. This mechanism works of simple concept. An event source generates as event and it is send to one or more listeners. When the listener listens the event it processes the event. To listen to an event the listener must be register with the event source. The event delegation model deals with three things **Events, Source of Event and Event Listener**

(1) Event : An event is said to be happened when a phenomenon occurs like the mouse is moved, mouse is clicked, a key is pressed etc. In java an event is an object that describes this change of state in a source that generated the event.

(2) Source of Event : A source is an object that can generate an event. A source can generate more than one type of event. An event source must register itself to its listener in order to listen to its event. There are methods for each type of event listener.

To **register** we have method **public void addEventTypeListener(EventTypeListener Object)**

To **unregister** we have method **public void removeEventListener(EventTypeListener Object)**

(3) Event Listeners : An event listener is one that listens or understand the event. In java an event listener is an object. In order to receive the event notification, the event object has to be registered with the event and its class must implement all methods of the events listener interface.

Some Event Classes

(1) EventObject : This class is in java.util package and it is super class of all events.

Constructor is **EventObject(Event source)** here source is an object that generates an event**.**

Method is **Object getSource( )** it returns the event object that generated the event such as Button.

(2) AWTEvent : This class is subclass of EventObject class and it is defined in java.awt package.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container
FocusEvent	Generated when a component gains or losses keyboard focus
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, opened, or quit.

1) **ActionEvent**: It defines 4 integer constants that can be used to identify any modifiers associated with an action event
   - ALT_MASLK
   - CTRL_MASK
   - META_MASK
   - SHIFT_MASK

   There is an integer constant **ACTION_PERFORMED** (it is used to identify action event)

   Command name can be obtained by the **getActionCommand( )** method. Eg. If there are three button in application, through this method we can get which button has invoked it.

Another method is **getModifiers( )**, which indicates the modifier key(alt, ctrl, meta, shift). Another method is **getWhen( )**, which returns the time at which the event took place

2) AdjustmentEvent:

It has five types of adjustment events. It defines integer constants that can be used to identify them

• BLOCK_DECREMENT	when user click inside the scroll bar for decreasing value
• BLOCK_INCREMENT	when user click inside the scroll bar for increasing value
• TRACK	when slider was dragged
• UNIT_DECREMENT	the button at the end of scroll bar clicked to decrease value
• UNIT_INCREMENT	the button at the end of scroll bar clicked to increase value

There is a integer constant, **ADJUSTMENT_VALUE_CHANGED**, which indicates that a change has occurred

The method **getAdjustable()** method returns the object that generated the event. The **getAdjustmentType()** returns type of adjustment event. The **getValue()** returns amount of adjustment.

3) ComponentEvent

There are 4 types of component events and 4 integer constants

• COMPONENT_HIDDEN	component becomes hidden
• COMPONENT_MOVED	component moves
• COMPONENT_RESIZED	component resized
• COMPONENT_SHOWN	component becomes visible

It is the super class of ContainerEvent, FocusEvent, KeyEvent, MouseEvent, WindowEvent class

The **getComponent( )** method returns the component that generated the event

4) ContainerEvent:

There are two types of container events. It defines two types of constants

- COMPONENT_ADDED
- COMPONENT_REMOVED

The **getContainer()** gives reference to the container that generated this event. The **getChild( )** method returns a reference to the component that was added or removed

5) FocusEvent

There are two integer constants

- FOCUS_GAINED
- FOCUS_LOST

The **getOppositeComponent( )** determines the <type>. The **isTemporary( )** method indicates if this focus change is temporary. It returns true if temporary else false.

6) ItemEvent

There are two integer constants

- DESELECTED
- SELECTED

It has other constant **ITEM_STATE_CHANGED**.

The getItem() method used to obtain a reference to the item that generated an event. The getItemSelectable() method can be used to obtain a reference to the ItemSelectable object that genereated an event the getStateChange() returns the state change for the event

7) MouseEvent

There are 8 integer constants defined.
MOUSE_CLICKED
MOUSE_DRAGGED
MOUSE_ENTERED
MOUSE_EXITED
MOUSE_MOVED
MOUSE_PRESSED
MOUSE_RELEASED
MOUSE_WHEEL

the **getX()** and **getY()** are used to find co-ordinates at the time of click. The **getPoint()** returns co-ordinates of the mouse. The **translatePoint(int x, int y)** changes the location of the event. The **getClickCount()** returns no. of mouse click. The **isPopupTrigger()** tests if this event cause a pop-up menu to appear on the platform. The **getButton()** returns the button that caused the event.

8) KeyEvent

There are 3 types of key events, identified by integer constants
- KEY_PRESSED
- KEY_RELEASED
- KEY_TYPED

There are 8 integer constants
- VK_0 TO VK_9
- VK_A TO VK_Z
- VK_ENTER
- VK_ESCAPTE
- VK_CANCEL
- VK_UP
- VK_DOWN
- VK_LEFT
- VK_RIGHT
- VK_PAGE_DOWN
- VK_PAGE_UP
- VK_SHIFT
- VK_ALT
- VK_CONTROL

For KEY_TYPED events, code will be VK_UNDEFINED.
The **getKeyChar()** returns the character that was entered. The **getKeyCode()** returns the keycode.

9) TextEvent

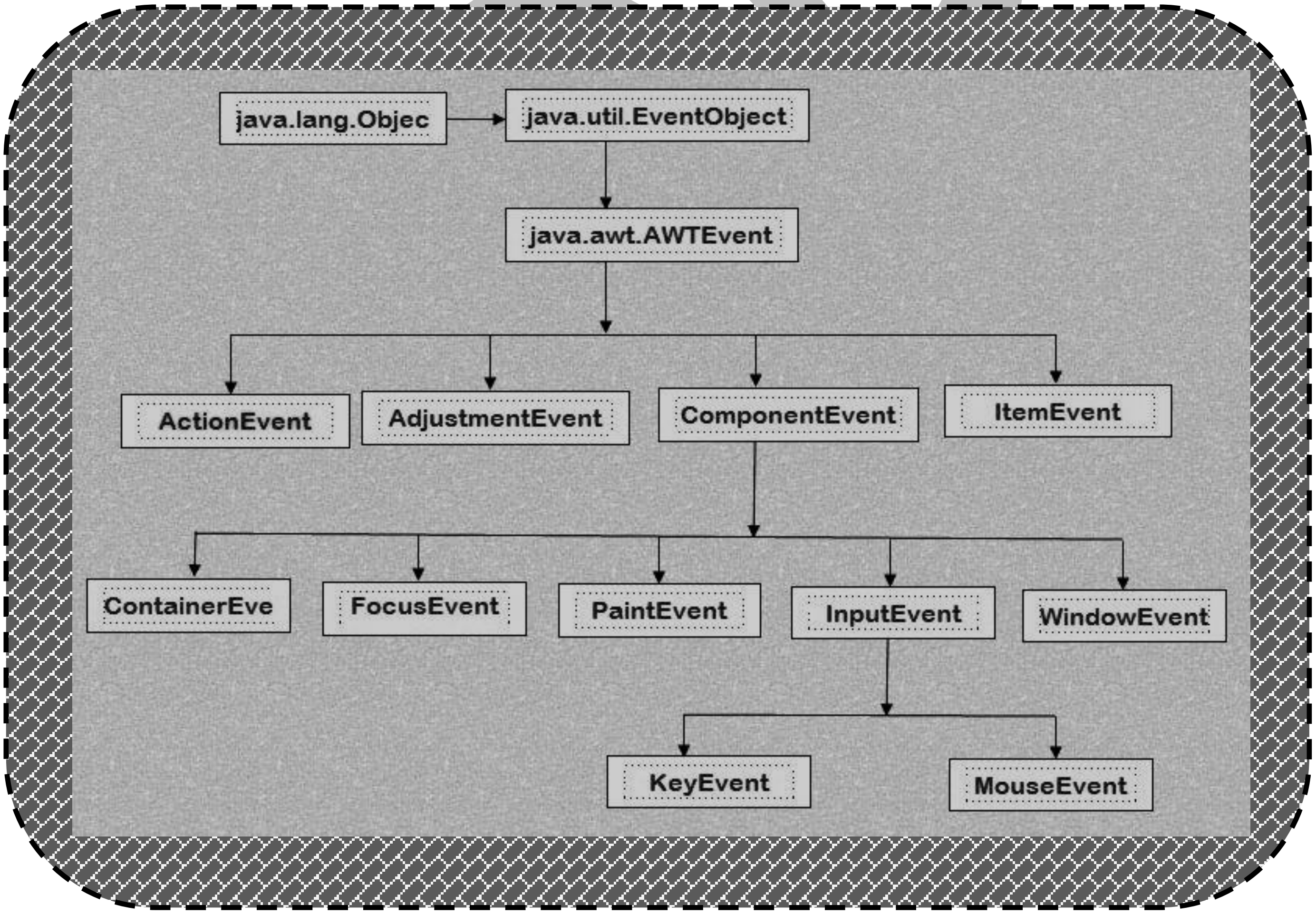
It defines integer constant TEXT_VALUE_CHANGED
The object does not include the character currently in the text component that generated the event.

10)WindowEvent

There are 10 types of window events.

- WINDOW_ACTIVATED
- WINDOW_CLOSED
- WINDOW_CLOSING
- WINDOW_DEACIVATED
- WINDOW_DEICONIFIED
- WINDOW_GAINED_FOCUS
- WINDOW_ICONIFIED
- WINDOW_LOST_FOCUS
- WINDOW_OPENED
- WINDOW_STATE_CHANGED

The methods are:
getOppositeWindow(), getOldState(), getNewState()



The Event Listener Interfaces

Event Listener Interface	Event Listener Methods
ActionListener	actionPerformed(ActionEvent evt)
AdjustmentListener	adjustmentValueChanged(AjustmentEvent evt)
ItemListener	itemStateChanged(ItemEvent evt)
TextListener	textValueChanged(TextEvent evt)
ComponentListener	componentHidden(ComponentEvent evt), componentMoved(ComponentEvent evt), componentResized(ComponentEvent evt), componentShown(ComponentEvent evt)
ContainerListener	componentAdded(ContainerEvent evt), componentRemoved(ContainerEvent evt)
FocusListener	focusGained(FocusEvent evt), focusLost(FocusEvent evt)
KeyListener	keyPressed(KeyEvent evt), keyReleased(KeyEvent evt), keyTyped(KeyEvent evt)
MouseListener	mouseClicked(MouseEvent evt), mouseEntered(MouseEvent evt), mouseExited(MouseEvent evt), mousePressed(MouseEvent evt), mouseReleased(MouseEvent evt)
MouseMotionListener	mouseDragged(MouseEvent evt), mouseMoved(MouseEvent evt)
WindowListener	windowActivated(WindowEvent evt), windowClosed(WindowEvent evt), windowClosing(WindowEvent evt), windowDeactivated(WindowEvent evt), windowDeiconified(WindowEvent evt), windowIconified(WindowEvent evt), windowOpened(WindowEvent evt)

```
// Demonstrate the Mouse Handler
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
```

```
public class MouseEvents extends Applet implements MouseListener
{
        String msg = "";
        int mouseX = 0, mouseY = 0; // coordinates of mouse
        public void init( )
        {
                addMouseListener(this);
                setBackground(Color.yellow);
        }
        // Handle mouse clicked.
        public void mouseClicked(MouseEvent me)
        {
                // save coordinates
                mouseX = 0;
                mouseY = 10;
                setBackground(Color.red);
                msg = "Mouse clicked.";
                repaint();
        }
        // Handle mouse entered.
        public void mouseEntered(MouseEvent me)
        {
                // save coordinates
                mouseX = 0;
                mouseY = 10;
                setBackground(Color.blue);
                msg = "Mouse entered.";
                repaint();
        }
        // Handle mouse exited.
        public void mouseExited(MouseEvent me)
        {
                // save coordinates
                mouseX = 0;
                mouseY = 10;
                setBackground(Color.green);
                msg = "Mouse exited.";
                repaint();
        }
        public void mousePressed(MouseEvent me)
        {
                // save coordinates
                mouseX = me.getX();
                mouseY = me.getY();
                msg = "Down";
                repaint();
        }
        // Handle button released.
        public void mouseReleased(MouseEvent me)
        {
                // save coordinates
                mouseX = me.getX();
                mouseY = me.getY();
                msg = "Up";
                repaint();
        }

        // Display msg in applet window at current X,Y location.
        public void paint(Graphics g)
        {
                g.drawString(msg, mouseX, mouseY);
        }
}
```

// Demonstrate the key event handlers.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="SimpleKey" width=300 height=100>
</applet>
*/
public class SimpleKey extends Applet implements KeyListener
{
        String msg = "";
        int X = 10, Y = 20; // output coordinates
        public void init( )
        {
                addKeyListener(this);
                requestFocus(); // request input focus
        }
        public void keyPressed(KeyEvent ke)
        {
                showStatus("Key Down");
                setBackground(Color.yellow);
        }
        public void keyReleased(KeyEvent ke)
        {
                showStatus("Key Up");
                setBackground(Color.blue);
        }
        public void keyTyped(KeyEvent ke)
        {
                msg += ke.getKeyChar();
                setBackground(Color.red);
                repaint();
        }
        // Display keystrokes.
        public void paint(Graphics g)
        {
                g.drawString(msg, X, Y);
        }
}
```

Adapter Classes

Java provides a special feature, called an **adapter class** that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

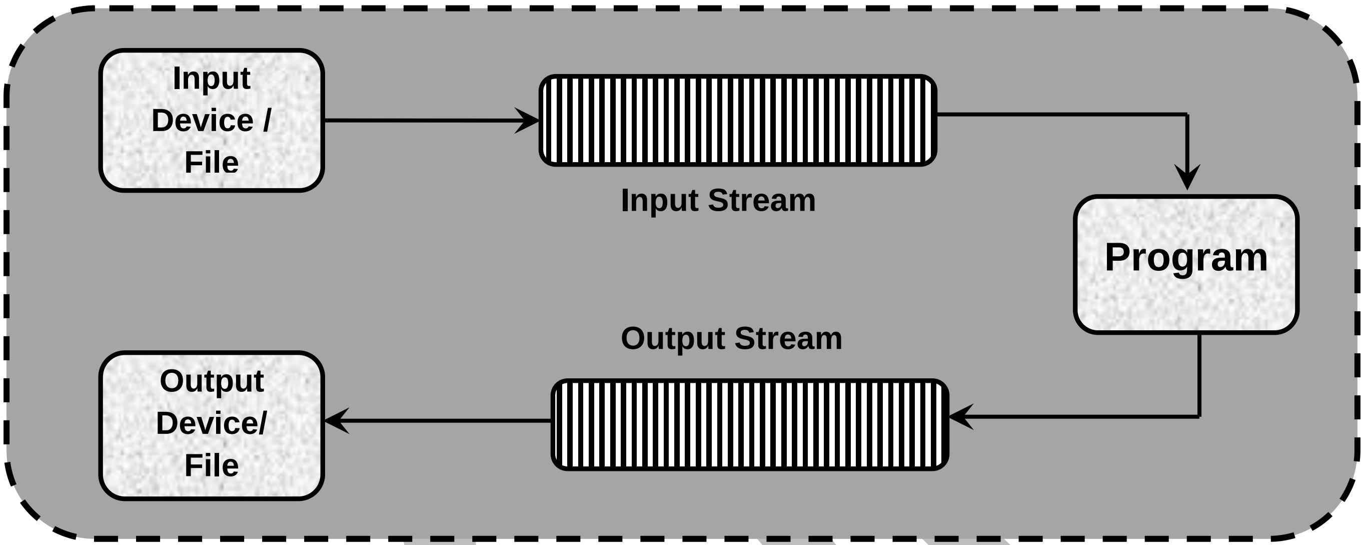
For example MouseListener interface has 5 methods but you want to use only two methods of that interface. If you are implementing MouseListener interface then you have to override all methods. But if you are using its adapter class MouseAdapter then you can override only those methods which you want, not all.

Event Listener interface	Event Listener Adapter
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

Input / Output

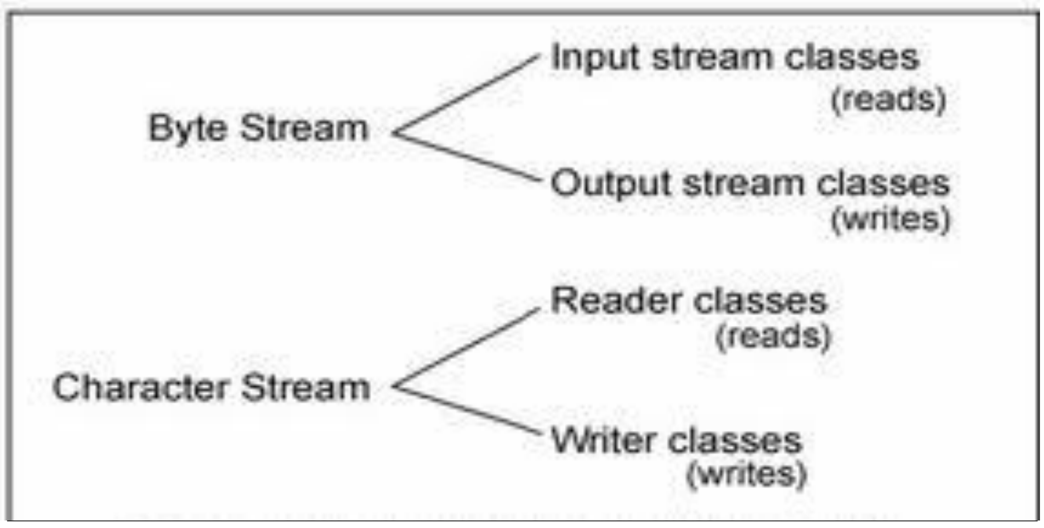
Streams

A stream is a path or a medium along which the data flows. So the data passes through streams from source to destination. The source is called input and the destination is called output of program. The input stream can be keyboard, mouse or file and the output stream can be monitor, printer or file.



There are two main categories of stream

1) **Character Stream**
2) **Byte Stream**

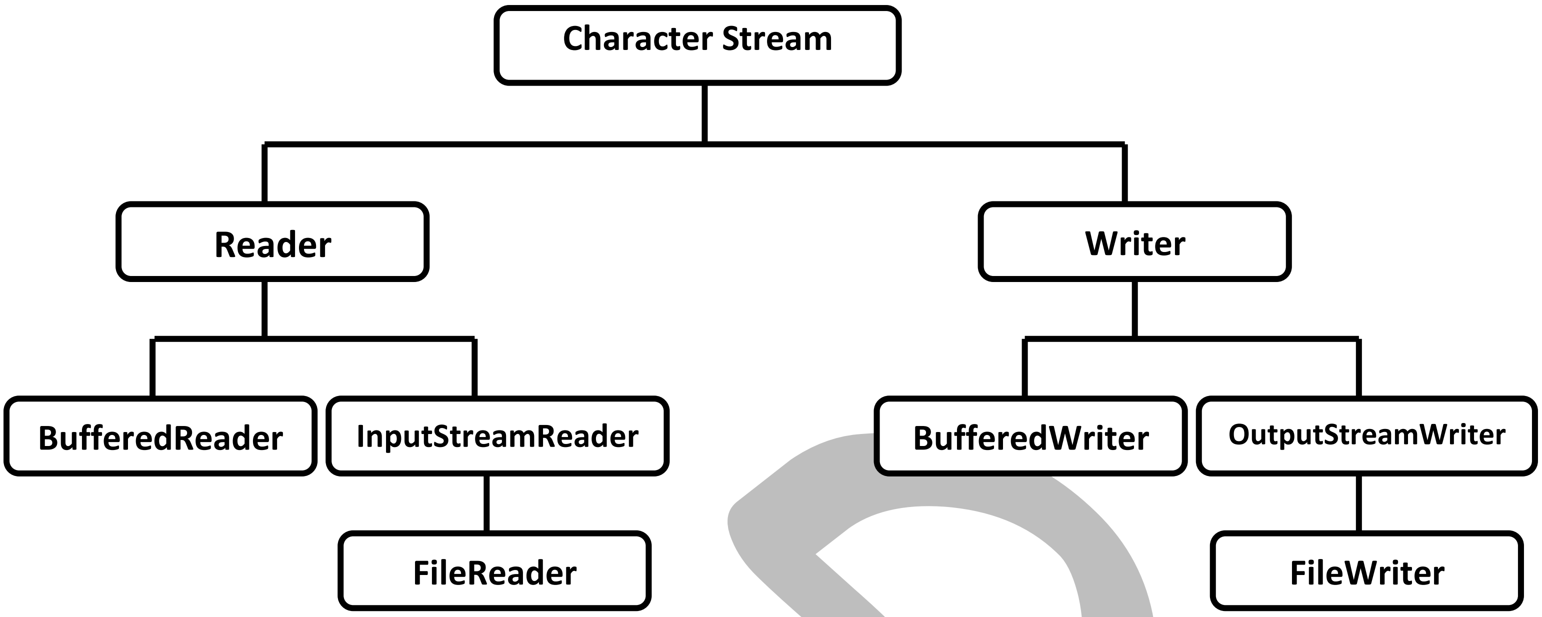


Character Stream:

The character stream manipulates the data as characters. A character in java is of 16 bits. Thus the character stream classes can work with 16 bit Unicode characters. The main two classes of character stream are **Reader and Writer.**

Byte Stream

The byte stream classes manipulate data byte by byte. A byte in java is of 8 bits. Thus a byte stream objects reads and writes data in binary form. The main two classes of byte stream are **InputStream and OutputStream**



Reader Class

The Reader class is abstract class, so we cannot create its object. Its subclasses are used to read character from file. The methods of Reader class are as follows.

(1) int read( ) : It reads the character from stream and returns integer value of that character. It returns -1 if EOF reached.

(2) abstract void close( ) : it closes the input source. If one tries to read a character after closing a file it will generate IOException.

(3) boolean ready( ) : It returns true if the next input is ready otherwise false.

(4) long skip(long numOfChars) : In this numOfChars will be skipped. It returns the number of characters actually skipped.

InputStreamReader Class

This class is the sub class of Reader class. It reads a byte from the input stream i.e. file and converts it to the character.

FileReader Class

This class is used to read characters from a file. It is a subclass of InputStreamReader class. Its constructors are

- **FileReader(String path)**
- **Filereader(File obj)**

```
import java.io.*;
class FileReadEx
{
	public static void main(String args[ ])
	{
		try
		{
			FileReader f = new FileReader(args[0]);
```

```
                    System.out.println("The content of file is ");
                    int ch;
                    do
                    {
                              ch = f.read( );
                              if(ch != -1)
                                        System.out.println((char)ch);
                    }while(ch!=-1);
          }
          catch(ArrayIndexOutOfBoundsException e)
          {
                    System.out.println("Please five file name as command line argument");
          }
          catch(FileNotFoundException e)
          {
                    System.out.println("File not found");
          }
          catch(IOException e)
          {
                    System.out.println("IO exception generated");
          }
     }
}
```

BufferedReader Class

This class is also used to take input from stream but it increases performance by using buffer to take input. The input first stored in buffer and then data is transferred from the buffer. Its constructors are

BufferedReader(Reader obj)
BufferedReader(Reader obj, int bufferSize)

The BufferedReader class adds a method readLine( ) which reads a line of characters from the keyboard. The reading of data is terminated when new line is encountered.

```
import java.io.*;
class ReadLineEx
{
          public static void main(String args[ ])
          {
                    String name = " ";
                    try
                    {
                              InputStreamReader  isr  = new InputStreamReader(System.in);
                              BufferedReader br = new BufferedReader(isr)
                              System.out.println("Please enter your name ");
                              name = br.readLine( );
                              System.out.println("hello : " + name + "  how are you");
                    }
                    catch(IOException e)
                    {
                              System.out.println("IO exception generated");
                    }
```

```
        }
}
```

Writer Class

This class is abstract class so we cannot create is object. Its subclasses are used to perform all output related tasks on a file. The methods of Writer class are as follows.

(1) void write(int ch) : It writes a single character in the output stream.

(2) void write(char buffer[ ]) : It writes a content of character array buffer to output stream.

(3) void write(String str) : It writes a string to output stream.

(4) void write(String str, int start, int numOfChars) : It writes numOfChars starting from the startindex of the string str to the output string.

(5) abstract void close( ) : It closes the output source. If one tries to write a character after closing a file it will generate IOException.

(6) abstract void flush( ) : It flushes the output buffer.

OutputStreamReader Class

This class is the subclass of writer class. It converts characters to bytes and then writes them to output stream.

FileWriter Class

This class is used to write characters in a file. Its constructors are

- **FileWriter(File object)**
- **FileWriter(String path)**
- **FileWriter(String path, boolean append)**

```
import java.io.*;
class FileWriterEx
{
        public static void main(String args[ ])
        {
                String str = “God is great”;
                char buffer[ ] = {‘a’,’b’,’c’,’d’};
                try
                {
                        FileWriter f = new FileWriter(“file1.txt”);
                        // BufferedWriter b = new BufferedWriter(f);
                        //b.write(str);
                        f.write(str);
                        f.write(buffer);
                        f.close( );
                }
                catch(IOException e)
                {
                        System.out.println("I/O exception occurred");
                }
        System.out.println("The data is successfully written in file");
        }
}
```

BufferedWriter Class

This class is the subclass of Writer class. It writes the characters into file using buffer so that the performance is increased. Its constructors are

- **BufferedWriter(Writer obj)**
- **BufferedWriter(Writer obj, int bufferSize)**

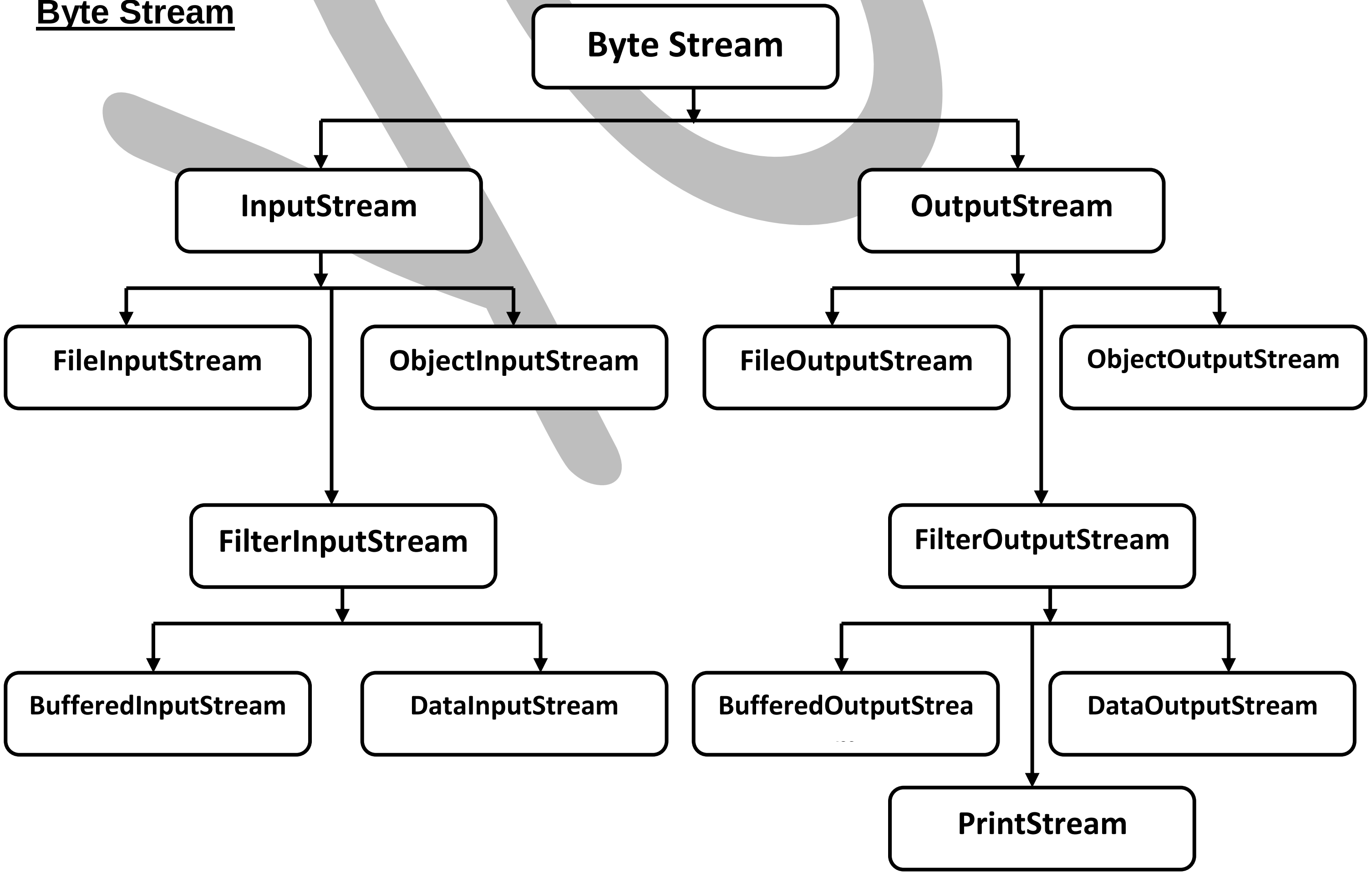
For example see program for FileWriter class.

PrintWriter Class

This class is used to display values of simple data types. It has methods to display data such as print( ) and println( ).

```
PrintWriter p = new PrintWriter(System.out);
p.println("Hello World");
```

Byte Stream



FileInputStream Class

This class is used to read bytes from a file. It is a subclass of InputStream class. Its constructors are

- **FileInputStream(String path)**
- **FileInputStream (File obj)**

```
import java.io.*;
class FileInputStreamEx
{
        public static void main(String args[ ])
        {
                try
                {
                        FileInputStream f = new FileInputStream(args[0]);
                        System.out.println("The content of file is ");
                        int ch;
                        do
                        {
                                ch = f.read( );
                                if(ch != -1)
                                        System.out.println((char)ch);
                        }while(ch!=-1);
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Please five file name as command line argument");
                }
                catch(FileNotFoundException e)
                {
                        System.out.println("File not found");
                }
                catch(IOException e)
                {
                        System.out.println("IO exception generated");
                }
        }
}
```

BufferedInputStream Class

This class is sub class of FilterInputStream. This class is also used to take input from stream but it increases performance by using buffer to take input. The input first stored in buffer and then data is transferred from the buffer. Its constructors are

<u>BufferedInputStream</u> (InputStream obj)
<u>BufferedInputStream</u> (InputStream obj, int bufferSize)

```
import java.io.*;
class BufferedInputEx
{
        public static void main(String args[ ])
        {
                try
                {
                        FileInputStream f  = new FileInputStream (file1.txt);
                        BufferedInputStream  br = new BufferedReader(f)
                        System.out.println("File content is ");
                        int ch;
                        do
                        {
                                ch = br.read( );
                                if(ch != -1)
                                        System.out.println((char)ch);
                        }while(ch!=-1);
                }
                catch(FileNotFoundException e)
                {
                        System.out.println("File not found");
                }
                catch(IOException e)
                {
                        System.out.println("IO exception generated");
                }
        }
}
```

DataInputStream Class

This class is the sub class of FilterInputStream class. This class allows reading of Java **Primitive** data value. It implements **DataInput** interface.

Its constructors is **DataInputStream (InputStream obj)**

Some Methods of DataInput interface are

(1) boolean readBoolean( )

(2) byte readByte( )

(3) char readChar( )

(4) short readShort( )

(5) int readInt( )

(6) float readFloat( )

(7) long readLong( )

(8) double readDouble( )

ObjectInputStream

ObjectInputStream is the sub class InputStream class. This class is used to read Object from the file. It has **readObject( )** method to read the data of object from file.

Its contructor **ObjectInputStream(InputStream obj)**

ObjectOutputStream

ObjectOutputStream is the sub class OutputStream class. This class is used to write Object to the file. It has **writeObject( )** method to write the data of object to file.

Its contructor **ObjectOutputStream(InputStream obj)**

```
import java.io.*;
class ObjectEx
{
        public static void main(String args[ ])
        {
                try
                {
                        Student  s1 = new Student(“XYZ”, 24, 75.45);
                        System.out.println(“Student 1 : \n” + s1);
                        FileOutputStream f = new FileOutputStream(“file5.txt”);
                        ObjectOutputStream  o = new ObjectOutputStream(f);
                        o.writeObject(s1);
                        o.flush( );
                        o.close( );
                }
                catch(Exception e)
                {
                        System.out.println(“Error occurred”);
                }
                try
                {
                        FileInputStream f = new FileOutputStream(“file5.txt”);
                        ObjectInputStream  o = new ObjectInputStream(f);
                        s2 = (Student)o.readObject(s1);
                        o.close( );
                        System.out.println(“Student 2 : \n” + s2);
                }
                catch(Exception e)
                {
                        System.out.println(“Error occurred”);
                }
        }
}
class Student implements Serializable
{
        String name;
        int rollno;
        double per;
        public Student(String name, int rollno, double per)
        {
                this.name = name;
                this.rollno = rollno;
                this.per = per;
        }
        public String toString( )
        {
                return "Name = " + name + "\n Roll No = " + rollno + "\n Percentage = " + per;
        }
}
```

How to Get Input From Keyboard

```
import java.io.*;
class ReadLineEx
{
        public static void main(String args[ ])
        {
                int num1,num2,num3;
                try
                {
                        InputStreamReader  isr  = new InputStreamReader(System.in);
                        BufferedReader br = new BufferedReader(isr);
                        System.out.println("Please Enter First number");
                        num1 = Integer.parseInt(br.readLine( ));
                        System.out.println("Please Enter Second number");
                        num2 = Integer.parseInt(br.readLine( ));
                        num3 = num1+num2;
                        System.out.println("Addition is : " + num3);
                }
                catch(IOException e)
                {
                        System.out.println("IO exception generated");
                }
        }
}
```

How to Copy one file to another file

```
import java.io.*;
class FileCopyEx
{
        public static void main(String args[ ]) throws IOException
        {
                int I;
                FileInputStream srcFile;
                FileOutputStream destFile;
                try
                {
                        try
                        {
                                srcFile = new FileInputStream(args[0]);
                        }
                        catch(FileNOtFoundException e)
                        {
                                System.out.println("Source file not found");
                                return;
                        }
                        try
                        {
                                destFile = new FileOutputStream(args[1]);
                        }
                        catch(FileNOtFoundException e)
                        {
                                System.out.println("Source file not found");
                                return;
                        }
                }
                catch(ArrayIndexOutOfBoundsException e)
                {
                        System.out.println("Please give both file names");
                        return;
                }
                try
                {
                        do
                        {
                                i = srcFile.read( );
                                if(i != -1)
                                        destFile.write(i);
                        }while(i != -1)
                }
                catch(IOException e)
                {
                        System.out.println(" IO error generated");
                }
                srcFile.close( );
                destFile.close( );

                System.out.println("The file is copied");
        }
}
```

RandomAccessFile Class

This class is used to read or write data from a file randomly. This class is not a subclass of InputStream or OutputStream but it is directly the subclass of Object class. Using this class you can access file randomly. Its constructors are

RandomAccessFile(File obj, String mode);
RandomAccessFile(String filename, String mode);

Here obj is the object of file class and filename is the name of the file. The mode is the access mode of the file. If the mode is "r" then read operation and if mode is "rw" then read and write both can be done.

Its methods are

(1) void seek(long NumOfByte) : It move the file pointer to desired location.

(2) int read( ) : Used to read a byte from the input stream.

(3) long getFilePointer( ) : It returns the current position of file pointer.

(4) long length( ) : It returns length of the file in byte.

(5) void setLength(long newLength) : Used to set length of file.

(6) void close( ) : It closes the file.

```
import java.io.*;
class RandomEx
{
        public static void main(String args[ ])
        {
                try
                {
                        RandomAccessFile r = new RandomAccessFile("file.txt","rw");
                        System.out.println("writing in file");
                        for(int i = 65; i<=77; i++)
                                r.write(i);
                        r.seek(0)
                        System.out.println("The first character is " + (char)r.read( ));
                        r.seek(r.length( ) -1);
                        System.out.println("The last character is " + (char)r.read( ));
                        System.out.println("The size of file is + r.length( ));
                        r.setLength(100);
                        System.out.println("Now The size of file is + r.length( ));
                        r.close( );
                }
                catch(IOException e)
                {
                        System.out.println("Error occurred");
                }
        }
}
```

File Class

This class is used to get information of a file such as length of file, is permission, directory path etc. You can also create a directory; delete file and directory using a file class. It has following constructors

- **File(String path)**
- **File(String directoryPath, String filename)**
- **File(File obj, String filename)**

The methods of file class are as below

1. **Boolean exist ()** - does the file exist return true if exists.
2. **Boolean canWrite( )** -return true if a file be written to else false.
3. **Boolean canRead( )** - retuen true if a file can be read else false.

4. **Boolean isFile( )** - does it represent a file or not
5. **Boolean isDirectory( )** - directory or not a directory
6. **String getName ( )** - get the name of the file (no path included)
7. **String getPath ( )** - get the abstract file path
8. **String getAbsolutePath ( )** - get the absolute file path
9. **long length( )** – It returns length of the file in bytes.

YOR COMPUTER CLASSES

❖ C Language	❖ Oracle
❖ C++	❖ Java
❖ HTML	❖ Adv. Java
❖ Data Structure	❖ C# .NET
❖ COA	❖ ASP.NET
❖ Visual Basic 6.0	❖ SQL Server
❖ PHP	❖ MS-Office
✓ Practical Lab with latest technology	
✓ Library with all kinds of reference books & materials	
✓ Personal attention on each student	
✓ Best material will be provided	

YOR COMPUTER CLASSES

Vishal Sir(MCA)	Naman Sir(MCA)
9427732231	9408526428

101 Shiromani Complex, Nr. Balaji Mandir,
Karansinghji Highschool Chowk,
Above YOR Restaurant, Rajkot - 360001

Swing

Swing is a set of classes that provides more **powerful** and **flexible** GUI components than AWT. There was a need of better approach than AWT. The solution was **SWING**. In this we have to study **javax.swing** package. The **javax** package is the extension package. This package contains lots of classes for swing components.

Swing is Built-On the AWT

Swing does not replace AWT; instead Swing is built on the foundation of the AWT. So the basic Swing also uses the same event handling mechanism as the AWT. So to understand Swing the basic knowledge of AWT and event handling is required.

AWT v/s Swing

AWT	Swing
Components are heavyweight	Components are lightweight
Component's look is depended on OS	Component's look do no depend on OS
Look and Feel of each component is Fixed	Look and Feel of each component is not fixed
Look of components could not be changed easily	Look of components could be changed easily without affecting the code

The MVC Connection

In general, a visual component is a combination of three aspects

- The way that the component looks when shown on screen
- The way that the component reacts to the user
- The state information associated with the component

In **MVC (Model View Controller)** terminology, the *model* corresponds to the state information associated with the component. The *view* determined how the component is displayed on screen. The *controller* determines how the component reacts to user. For example when the user clicks a check box, the controller reacts by changing the model to reflect the user's choice. This then results in the view being updated. The component is separated into model, view and controller for better programming.

The Container Class

The java.awt.Container class is the sub class of the java.awt.Component class. This container can contain any components within it. These components are arranged by layout managers.

We must extend **JApplet** class to use Swing components. The JApplet provides the support for various panes such as content pane, the glass pane etc. To add a component in the JApplet you have to use add ( ) method by a content pane object.
Container getContentPane( ) method is used to get content pane. To add component in this pane use **void add(componentObj)**

Concepts of Layouts (Layout Managers)

The layout manager decides how the control should be **arranged or positioned** within a container or a window. In some languages like Visual Basics you can arrange your controls as you like using form designer. But in Java there is no such form designer, but you have to write the code to arrange the control where you want them.

- To set any layout we can use **void setLayout(LayoutManager obj)** method. The obj is an object of the specified layout manager.
- If no layout is set by the setLayout( ) method the **default** layout manager is used which is **FlowLayout**.

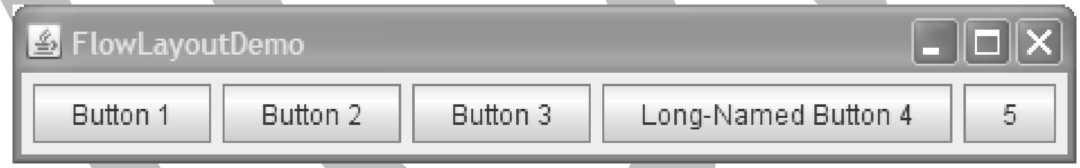
(1) FlowLayout : The FlowLayout is the default layout manager i.e. if no layout manager is set by the setLayout( ) method, this layout is used to arrange the controls. In this layout manager the components are arranged from the upper-left corner to the right side edge. When there is no space for component it is arranged from the next line.

The constructors for this layout are

- **FlowLayout( )**
- **FlowLayout(int align)**
- **FlowLayout(int align, hspace, vspace)**

Here, the align parameter specifies the alignment for the component. It can be **FlowLayout.CENTER(default), FlowLayout.LEFT** and **FlowLayout.RIGHT** constants.

The vspace and hspace specifies the horizontal and vertical spaces between the components. Default is **5 pixels**.



(2) BorderLayout : The BorderLayout arrange the components in five portions. These portions are the four edges of the window and the fifth is the center part. These portions are specified by the following constants:

BorderLayout.NORTH : The top edge of the window
BorderLayout.SOUTH : The bottom edge of the window
BorderLayout.EAST : The right side edge of the window
BorderLayout.WEST : The left side edge of the window
BorderLayout.CENTER : The center portion of the window

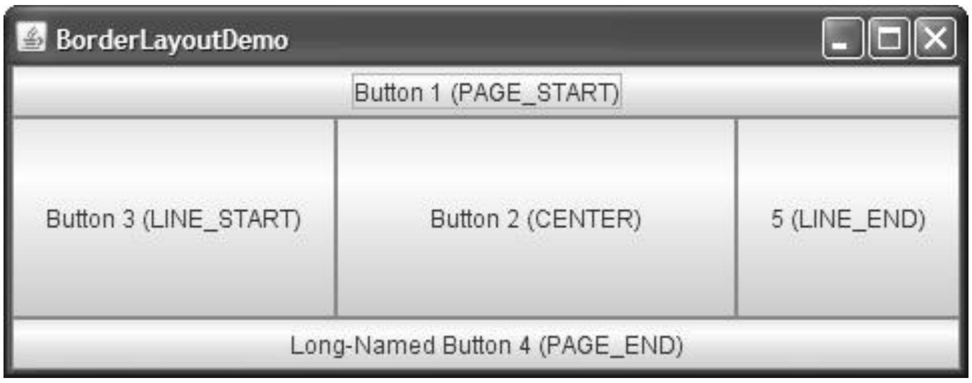
The constructors are **BorderLayout( )**
BorderLayout (int hspace, int vspace)

The component can be added by **void add(Component obj, Object where)**

Use of Insets

The insets class provides the facility to give some space between the edges of the applet window and your components. To do so you have to override the getInsets( ) method defined by the Container class.

Constructor for this class is	**Insets(int top, int left, int bottom, int right)**
Method you must override is	**Insets getInsets( )**

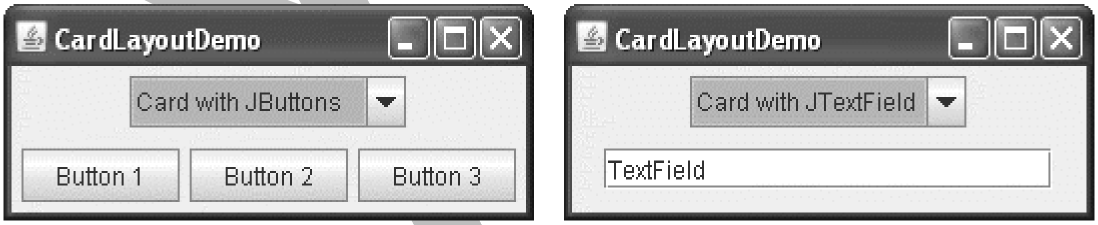


<u>(3) CardLayout :</u> This layout has some special capabilities that other layout do not have. The CardLayout creates a layout like the playing card. Assume more than one card on one another. Here only the card on the top is visible at a time. But you can shuffle the card to see other cards. Same way the CardLayout can switch among several panels. The constructors are

- **CardLayout( )**
- **CardLayout(int hspace, int vspace)**

To add components or panels to the card layout panel **void add(Component panelObj, Object panelname)** method is used.
Here the panelname is the name of the panel object.
Now after defining the panel object you can show or shuffle the card panels by one of the following

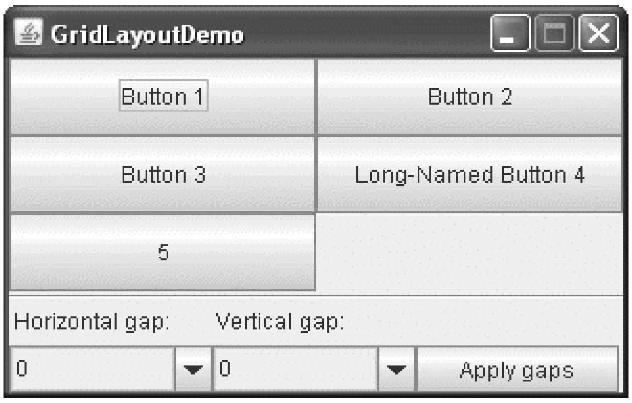
- **void show(Container panelObj, String panelName) :** it shows the panel specified by the panelObj and whose name is panelName
- **void first(Container panelObj) :** It shows first panel or card
- **void last(Container panelObj) :** It shows last panel or card
- **void next(Container panelObj) :** It shows next panel or card
- **void previous(Container panelObj) :** It shows previous panel or card



<u>(4) GridLayout :</u> It creates layout which has a grid of rows and columns. The constructors are

- **GridLayout( )**
- **GridLayout(int rows, int cols)**
- **GridLayout(int rows, int cols, int hspace, int vspace)**

The first constructor is default constructor which creates only one column grid. While in other two you can specify the number of rows and columns. Hspace and vspace specifies horizontal and vertical space between the components.



(5) GridBagLayout : The GridBagLayout is very flexible layout manager. It is an extension of GridLayout that provides some more features than the GridLayout. In the GridLayout all the cells have same height and width which is not necessary in GridBagLayout. Here you can have cells of arbitrary width and height. This can be done by specifying constraints. To specify constraints you have to create object of GridBagConstraints.

Its constructor is **GridBagConstraints( )**
GridBagLayout( )

You can set the following **GridBagConstraints** instance variables:

gridx, **gridy**

Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0. Use GridBagConstraints.RELATIVE (the default value) to specify that the component be placed just to the right of (for gridx) or just below (for gridy) the component that was added to the container just before this component was added. We recommend specifying the gridx and gridy values for each component rather than just using GridBagConstraints.RELATIVE; this tends to result in more predictable layouts.

gridwidth, **gridheight**

Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area. These constraints specify the number of cells the component uses, *not* the number of pixels it uses. The default value is 1. Use GridBagConstraints.REMAINDER to specify that the component be the last one in its row (for gridwidth) or column (for gridheight). Use GridBagConstraints.RELATIVE to specify that the component be the next to last one in its row (for gridwidth) or column (for gridheight). We recommend specifying the gridwidth and gridheight values for each component rather than just using GridBagConstraints.RELATIVE and GridBagConstraints.REMAINDER; this tends to result in more predictable layouts.

fill

Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component. Valid values (defined as GridBagConstraints constants) include NONE (the default), HORIZONTAL (make the component wide enough to fill its display area horizontally, but do not change its height), VERTICAL (make the component tall enough to fill its display area vertically, but do not change its width), and BOTH (make the component fill its display area entirely).

ipadx, **ipady**

Specifies the internal padding: how much to add to the size of the component. The default value is zero. The width of the component will be at least its minimum width plus ipadx*2 pixels, since the padding applies to both sides of the component. Similarly, the height of the component will be at least its minimum height plus ipady*2 pixels.

insets

Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area. The value is specified as an Insets object. By default, each component has no external padding.

anchor

Used when the component is smaller than its display area to determine where (within the area) to place the component. Valid values (defined as GridBagConstraints constants) are CENTER (the default), PAGE_START, PAGE_END, LINE_START, LINE_END, FIRST_LINE_START, FIRST_LINE_END, LAST_LINE_END, and LAST_LINE_START.

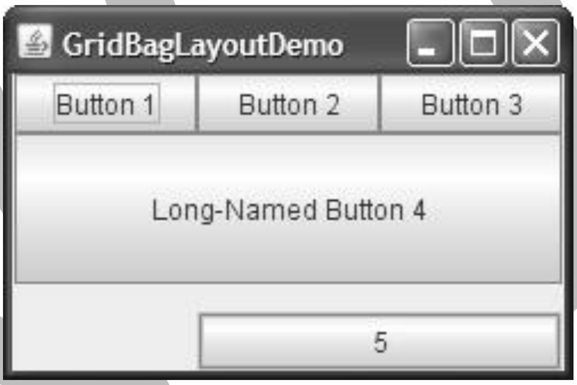
Here is a picture of how these values are interpreted in a container that has the default, left-to-right component orientation.

FIRST_LINE_START	PAGE_START	FIRST_LINE_END
LINE_START	CENTER	LINE_END
LAST_LINE_START	PAGE_END	LAST_LINE_END

weightx, **weighty**

The weightx and weighty specify the horizontal and vertical spaces between the edges of the container and the cell respectively. The default value is 0.0.

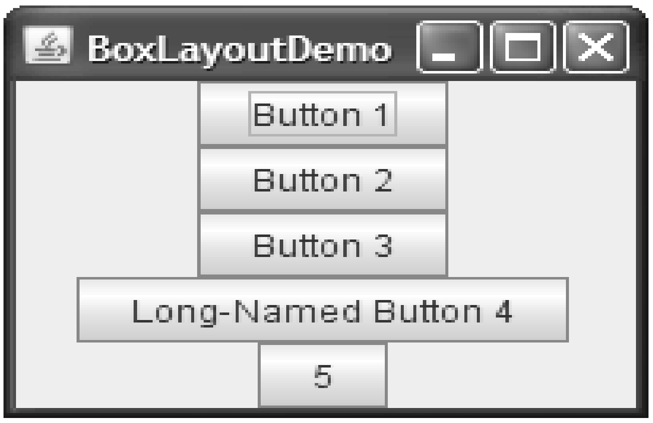
The constraints can be set by method **void setConstraints(Component obj, GridBagConstraints obj)**



(6) BoxLayout : It is general purpose layout manager which is an extension of FlowLayout. It places the components on the top of each other or places them one after another in a row.

Its constructor is **BoxLayout(Container obj, int placement)**

Here placement can be **BoxLayout.PAGE_AXIS** : - It places components on top of each other.
BoxLayout.LINE_AXIS : - It places components line by line

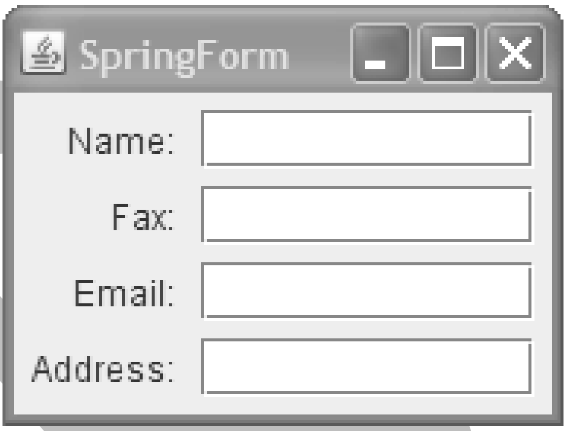


<u>(7) SpringLayout</u> : It is very flexible layout that has many features for specifying the size of the components. You can have different size rows and columns in a container. In SpringLayout you can define a fixed distance between the left edge of one component and right edge of another component.

Its constructor is **SpringLayout( )**

The constraints can be put by method

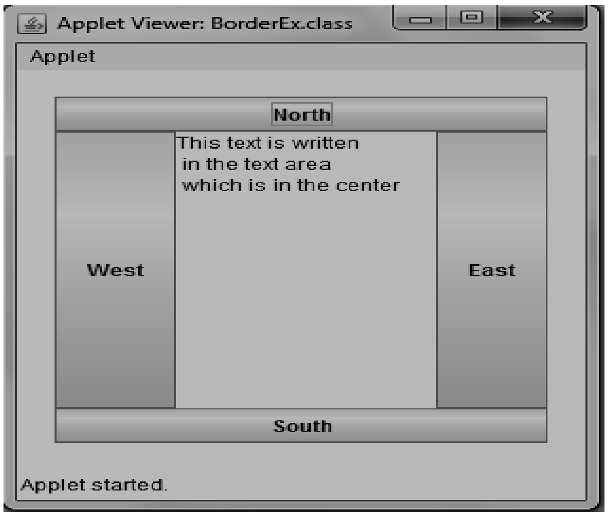
void putConstraints(String edge1, Component obj1, int distance, String edge2, Component obj2)



<u>Program for BorderLayout</u>

```
import java.awt.*;
import javax.swing.*;
/*
<applet code = "BorderEx.class" width = 300 height = 300>
</applet> */
public class BorderEx extends JApplet
{
        public void init( )
        {
                Container pane = getContentPane( );
                pane.setLayout(new BorderLayout( ));
                pane.add(new JButton("North"), BorderLayout.NORTH);
                pane.add(new JButton("South"), BorderLayout.SOUTH);
                pane.add(new JButton("East"), BorderLayout.EAST);
                pane.add(new JButton("West"), BorderLayout.WEST);
                String msg = "This text is written \n in the text area \n which is in the center";
                pane.add(new JTextArea(msg), BorderLayout.CENTER);
        }
        public Insets getInsets( )
        {
                Insets i = new Insets(20,20,20,20);
                return i;
        }
}
```

//Output

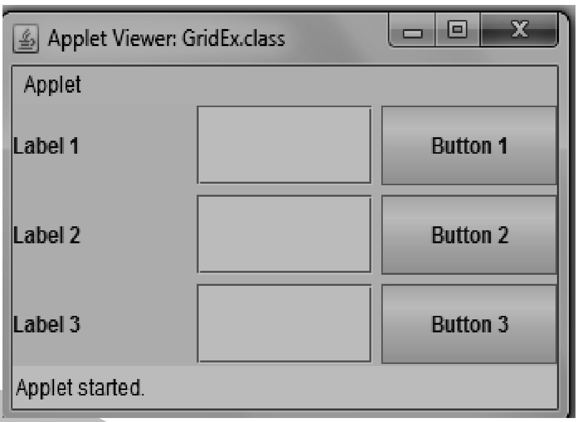


```
This text is written
 in the text area
 which is in the center
```

Program for GridLayout

```
import java.awt.*;
import javax.swing.*;
/*
<applet code = "GridEx.class" width = 300 height = 300>
</applet> */
public class GridEx extends JApplet
{
        public void init( )
        {
                Container pane = getContentPane( );
                pane.setLayout(new GridLayout(3,3,5,5));
                for(int i = 1; i<=3; i++)
                {
                        pane.add(new JLabel("Label " + i));
                        pane.add(new JTextField(5));
                        pane.add(new JButton("Button " + i));
                }
        }
}
```

Output



Components of Swing

JLabel

With the JLabel class, you can display un-selectable text and images. If you need to create a component that displays a string, an image, or both, you can do so by using or extending JLabel.

- Provide text instructions on a GUI
- Read-only text
- Programs rarely change a label's contents

JLabel Constructor

- **JLabel (String text)** Creates a JLabel instance with the specified text.
- **JLabel (Icon image)** Creates a JLabel instance with the specified image.
- **JLabel (String text, Icon i, int Align)** Creates a JLabel instance with the specified text, image, and alignment.

The Icon is an interface which is implemented by the ImageIcon class. This class draws an image. Its constructor are

- **ImageIcon(String filename)**
- **ImageIcon(URL url)**

JLabel Method

- **Icon getIcon**( ) -- Returns the graphic image (glyph, icon) that the label displays.
- **String getText**( ) -- Returns the text string that the label displays.
- **void setIcon**(Icon icon) -- Defines the icon this component will display.
- **void setText**(String text) -- Defines the single line of text this component will display.

JTextField

JTextField allows editing/displaying of a single line of text. New features include the ability to justify the text left, right, or center, and to set the text's font. JTextField is an input area where the user can type in characters. It can generate ActionEvent when user presses the enter key after input the data in the text field.

JTextField Constructor

- **JTextField( )** Constructs a new TextField.
- **JTextField(int columns)** Constructs a new empty TextField with the specified number of columns.
- **JTextField(String text)** Constructs a new TextField initialized with the specified text.
- **JTextField(String text, int columns)** Constructs a new TextField initialized with the specified text and columns.

JTextArea

JTextArea allows editing of multiple lines of text. It does not display any scroll bar. JTextArea can be used in conjunction with class JScrollPane to achieve scrolling.

JTextArea Constructor

- **JTextArea ( )** Constructs a new TextArea.
- **JTextArea (int rows, int columns)** Constructs a new empty TextArea with the specified number of rows and columns.
- **JTextArea (String text)** Constructs a new TextArea with the specified text displayed.
- **JTextArea (String text, int rows, int columns)** Constructs a new TextArea with the specified text and number of rows and columns.

JPasswordField

JPasswordField creates a text field which displays the disk (dots) instead of the actual characters. It is used when user interacts with password type of data.

JPasswordField Constructor

- **JPasswordField( )** Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
- **JPasswordField(int length)** Constructs a new empty JPasswordField with the specified number of characters.
- **JPasswordField(String text)** Constructs a new JPasswordField initialized with the specified text.
- **JPasswordField(String text, int length)** Constructs a new JPasswordField initialized with the specified text and length.

JButton

A button is a component the user clicks to trigger a specific action. Command buttons: is created with class JButton. It generates ActionEvent.

JButton Constructor

- **JButton(Icon i)** Creates a button with an icon.
- **JButton(String text)** Creates a button with text.
- **JButton(String text, Icon i)** Creates a button with initial text and an icon.

JCheckBox

A checkbox item can be selected and deselected, and it also displays its current state.

JCheckBox Constructor

- **JCheckBox(Icon i)**
  Creates an initially unselected check box with an icon.
- **JCheckBox(Icon i,boolean state)**
  Creates a check box with an icon and specifies whether or not it is initially selected.
- **JCheckBox(String text)**
  Creates an initially unselected check box with text.
- **JCheckBox(String text,boolean stae)**
  Creates a check box with text and specifies whether or not it is initially selected.
- **JCheckBox(String text,Icon i)**
  Creates an initially unselected check box with the specified text and icon.
- **JCheckBox(String text,Icon i,boolean state)**
  Creates a check box with text and icon, and specifies whether or not it is initially selected.

JRadioButton

It is used to create radio button. It is Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected. (Create a **ButtonGroup** object and use it's add method to include the JRadioButton objects in the group.)

Note: The **ButtonGroup** object is a logical grouping -- not a physical grouping. To create a button panel, you should still create a JPanel or similar container-object and add a Border to it to set it off from surrounding components.

JRadioButton Constructor

- **JRadioButton(Icon i)**
  Creates an initially unselected radio button with the specified image but no text.
- **JRadioButton(Icon i, boolean state)**
  Creates a radio button with the specified image and selection state, but no text.

- **JRadioButton(String text)**
  Creates an unselected radio button with the specified text.
- **JRadioButton(String text, boolean state)**
  Creates a radio button with the specified text and selection state.
- **JRadioButton(String text, Icon i)**
  Creates a radio button that has the specified text and image and that is initially unselected.
- **JRadioButton(String text, Icon i, boolean state)**
  Creates a radio button that has the specified text, image, and selection state

JComboBox

It creates a combo box which is a combination of a text field and a drop down list.

JComboBox Constructor

- **JComboBox( )** Creates a JComboBox with a default data model.
- **JComboBox(Vector items)** Creates a JComboBox that contains the elements in the specified Vector.

To add items later in the combo box use **void addItem(Object item)** method.

JScrollPane

A scroll pane is used to display other component or an image in a rectangular area. This pane can have horizontal as well as vertical scroll bars.

JScrollPane Constructor

- **JScrollPane (Component obj)**
- **JScrollPane (int vertScrollBar, int horiScrollBar)**
- **JScrollPane (Component obj, int vertScrollBar, int horiScrollBar)**

Its constants are

HORIZONTAL_SCROLLBAR_ALWAYS

HORIZONTAL_SCROLLBAR_AS_NEEDED

VERTICAL_SCROLLBAR_ALWAYS

VERTICAL_SCROLLBAR_AS_NEEDED

JList

The JList class creates a list box which can display more than one item in the box at a time. It also allows the user to select more than one item from the list. If the items in the list are too many to display, you have to add it in a scroll pane as it does not display it by default.

JList Constructor

- **JList ( )** Constructs a JList with an empty model.
- **JList (Object[ ] listData)** Constructs a JList that displays the elements in the specified array.
- **JList (Vector listData)** Constructs a JList that displays the elements in the specified Vector.

JPanel

The JPanel class is used to create a panel. We can add our components to a panel and panel should be added to a container such as frame. Now to draw the applet window **public void paintComponent(Graphics g)** method is used instead of paint( ) method.

JFrame

The JFrame class creates a frame which is a window. This window has a border, a title and buttons for closing, minimizing and maximizing the window.

JFrame Constructor

- **JFrame( )** Constructs a new frame that is initially invisible.
- **JFrame(String title)** Creates a new, initially invisible Frame with the specified title.

Frame Program

```
import javax.swing.*;
public class JFrameex
{
        public JFrameex()
        {
                JFrame fr= new JFrame("my frame");
                fr.setSize(250,150);
                fr.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
                JLabel lbl= new JLabel("this label is on frame");
                fr.add(lbl);
                fr.setVisible(true);
        }
        public static void main(String args[])
        {
                JFrameex obj= new JFrameex();
        }
}
```

JScrollBar

The JScrollBar is used to create vertical and horizontal scroll bars.

JScrollBar Constructor

- **JScrollBar ( )**
- **JScrollBar (int type) // type = JScrollBar.HORIZONTAL or JScrollBar.VERTICAL**
- **JScrollBar (int type, int initialPos, int sliderSize, int min, int max)**

JMenu, JMenuBar, JMenuItem

These classes provides functionalities to create a menu bar which contains menus. These menus in turns have menu items. The menu bar is created using JMenuBar class, menu is created using JMenu class, and the menu items are created using JMenuItems.

To Create a menu follow the following steps

1. **Create a JMenuBar object**
2. **Then set the menu bar using setJMenuBar( ) method.**
3. **Create menu object of JMenu class.**
4. **Create menu item using JMenuItem class.**
5. **Add menu items to particular menus.**
6. **Add menu to menu bar.**

JMenu, JMenuBar, JMenuItem Constructor

- **JMenuBar ( )**
  - **JMenu ( )**
  - **JMenu (String name)**
    - **JMenuItem ( )**
    - **JMenuItem (String itemName)**
    - **JMenuItem (Icon i)**
    - **JMenuItem (String itemName, Icon i)**

Menu Program

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Menuex extends JFrame implements ActionListener
{
        JLabel lbl;
        public Menuex(String title)
        {
                super(title);
                Container pane=getContentPane();
                pane.setLayout(new FlowLayout());
                lbl=new JLabel();
                JMenuBar mbar= new JMenuBar();
                setJMenuBar(mbar);
                JMenu file= new JMenu("file");
                JMenu mail= new JMenu("mail");

                JMenuItem i1=new JMenuItem("new");
                JMenuItem i2=new JMenuItem("open");
                JMenuItem i3=new JMenuItem("email");
                JMenuItem i4=new JMenuItem("inbox");
                file.add(i1);
                file.add(i2);
                mail.add(i3);
                mail.add(i4);

                mbar.add(file);
                mbar.add(mail);

                setSize(200,200);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                i1.addActionListener(this);
                i2.addActionListener(this);
                i3.addActionListener(this);
                i4.addActionListener(this);
        }
        public void actionPerformed(ActionEvent obj)
        {
                lbl.setText("you selected" + obj.getActionCommand());
        }

        public static void main(String args[])
        {
                Menuex obj1=new Menuex("my menu");
        }
}
```

Radio Button Program

```
import java.awt.*;
import javax.swing.*;

/* <applet code = “RadioEx.class” width = 300 height = 300>
</applet> */

public class RadioEx extends JApplet
{
        JLabel g,m;
        ButtonGroup gender, marital;
        JRadioButton male, female, single, married;
        public void init( )
        {
                Container pane = getContentPane( );
                pane.setLayout(new FlowLayout( ));
                male = new JRadioButton(“Male”);
                female = new JRadioButton(“Female”);
                single = new JRadioButton(“Single”);
                married = new JRadioButton(“Married”);
                gender = new ButtonGroup( );
                marital = new ButtonGroup( );
                gender.add(male);
                gender.add(female);
                marital.add(single);
                marital.add(married);
                g = new JLabel("Gender");
                m = new JLabel("Marital Status");
                pane.add(g);
                pane.add(male);
                pane.add(female);
                pane.add(m);
                pane.add(single);
                pane.add(married);
        }
}
```

Java Question Bank

- Characteristics of Java
- Explain JIT & JVM
- Why Java is platform independent language?
- Why public static void main (String args[ ]) ?
- Explain Type casting
- What is Array? Explain
- Explain new operator in Java
- Explain "this" keyword
- Define Garbage collection
- Define Finalise ( ) method
- Write a note on static members
- Write a note on Access Specifiers
- Explain final keyword with its uses.
- Write a note on Inheritance
- Explain super keyword with its uses
- What is abstract class and method?
- Explain Command Line Arguments
- Explain Dynamic Method Dispatch
- Write a note on Interface
- Define Native method, volatile and transient
- Write a note on Package
- How Java supports Exception Handling?
- Differentiate throw and throws
- Life cycle of Thread
- By how many ways you can create thread.
- Write a note on Priorities of Thread
- Define Deadlock
- Explain Inter Thread Communication
- What is Applet? Explain
- Life Cycle of Applet
- Explain Applet tag
- Explain Vector class
- Explain Event Delegation Model
- Explain Adapter class
- Write a note on stream
- What is Swing? Differentiate awt & swing
- Write a note on Layout available in Java
- Explain Border Layout in detail
- Write a note on JFrame