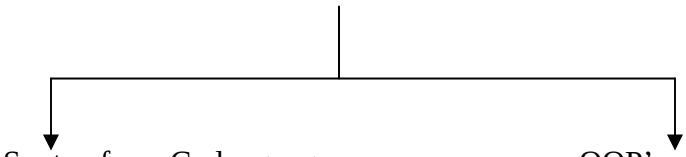
Java notes

Genesis of JAVA

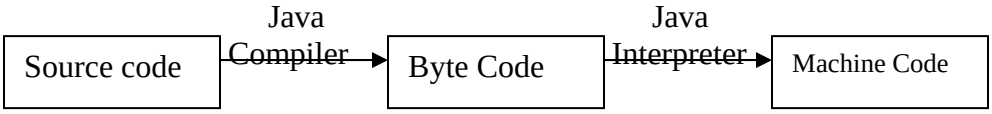
1. *JAVA*



Syntax form C –language

OOP's concepts features are influences from C++ language.

2. C, Pascal , Basic, Fortran are the based on either compiler or interpreter. But Java programs are based on compiler and interpreter both.



3. Java program is platform independent to achieve the independency the Byte codes area used.
4. Java Compiler is generally known as JVM(Java Virtual Machine).

Importance of java in Internet

One of the java advantages for use over the Internet is that platform independent, meaning that programs created with the java can run on any machine (computers).Java achieves its independence by creating programs designed to run on the JVM rather than any specific computer system.

These java programs once compiled and stored in a compact machine independent format known as Byte codes. Byte code files area easily transferred across the Internet. The preloaded interpreter run the byte code in the local machine then plays them.

Java Features

- **Simple** - Java's developers deliberately left out many of the unnecessary features of other high-level programming languages. For example, Java does not support pointer math, implicit type casting, structures or unions, operator overloading, templates, header files, or multiple inheritance.

- **Object-oriented.** Just like C++, Java uses classes to organize code into logical modules. At runtime, a program creates objects from the classes. Java classes can inherit from other classes, but multiple inheritance, wherein a class inherits methods and fields from more than one class, is not allowed.

- **Statically typed** - All objects used in a program must be declared before they are used. This enables the Java compiler to locate and report type conflicts.

- **Compiled** - Before you can run a program written in the Java language, the program must be compiled by the Java compiler. The compilation results in a "byte-code" file that, while similar to a machine-code file, can be executed under any operating system that has a Java interpreter. This interpreter reads in the byte-code file and translates the byte-code commands into machine-language commands that can be directly executed by the machine that's running the Java program. You could say, then, that Java is both a compiled and interpreted language.

- **Multi-threaded** - Java programs can contain multiple threads of execution, which enables programs to handle several tasks concurrently. For example, a multi-threaded program can render an image on the screen in one thread while continuing to accept keyboard input from the user in the main thread. All applications have at least one thread, which represents the program's main path of execution.

- **Garbage collector** - Java programs do their own garbage collection, which means that programs are not required to delete objects that they allocate in memory. This relieves programmers of virtually all memory-management problems.

- **Robust** - Because the Java interpreter checks all system access performed within a program, Java programs cannot crash the system. Instead, when a serious error is discovered, Java programs create an exception. This exception can be captured and managed by the program without any risk of bringing down the system.

- **Secure** - The Java system not only verifies all memory access but also ensures that no viruses are hitching a ride with a running applet. Because pointers are not supported by the Java language, programs cannot gain access to areas of the system for which they have no authorization.

- **Extensible** - Java programs support native methods, which are functions written in another language, usually C++. Support for native methods enables programmers to write functions that may execute faster than the equivalent functions written in Java. Native methods are dynamically linked to the Java program; that is, they are associated with the program at runtime. As the Java language is further refined for speed, native methods will probably be unnecessary.

- **Well-understood** - The Java language is based upon technology that's been developed over many years. For this reason, Java can be quickly and easily understood by anyone with experience with modern programming languages such as C++.

- **Java Applet** – Another advantage of the java for the internet is its ability to create applets. Applets are the small programs that area designed to be embedded into a Web Page. Any java enabled browser can load and run java applets directly form web pages. Applets provide World Wide Web pages with a level of interactivity never before possible on the internet.

Object Oriented Programming Concepts

What is the encapsulation?
Definition :-
Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.

Encapsulation is the combining of data and the code that manipulates that data into a single component-that is, an object. Encapsulation also refers to the control of access to the details of an object's implementation.

```
//Example of Encapsulation
class data {
        int idno, mark;
        String name;
        data() {
                name="raja";
                idno=12;
                mark=900;
        }
        void show() {
                System.out.println("Name of student is="+name);
                System.out.println("id no. of student is="+idno);
                System.out.println("mark of student is="+mark);
        }
}

class firstdemo {
        public static void main(String args[]) {
        data obj=new data();
        obj.show();
        }
}
```

Save this program named as ***firstdemo.java*** and complied as ***c:\>javac firstdemo.java*** and run above program as ***c:\>java firstdemo*** **{where object call the data with the help of objects}**

What is the Polymorphism?

Polymorphism build in two words =poly+morphism here poly means many and morphism means shapes. And merely means using the same one name to refer to different methods.
"Name Reuse" would be better terms to use for polymorphism.
There are two types of polymorphism in java.

1. **Overloading:(Same name but different parameter)**

In java, it is possible to create methods that have same name but different definition. That is called method overloading. Method overloading is used where objects area required performing similar task but using different input parameters. When we call a method in an object, JVM matches up the method name first and then the number and type of parameters to decide which one of the definition to execute.

```
//Example of overloading.

class student {
        int idno, mark;
        String name;
        void setData() {
                idno=90;mark=89;name="raja";
        }
        void setData(int id,int m,String n) {
                idno=id;mark=m;name=n;
        }
        void show() {
                System.out.println("\n\nStudent RollNo.="+idno);
                System.out.println("Student Name.="+name);
                System.out.println("Student RollNo.="+mark);
        }
}//end of student class

class demoOverload {
        public static void main(String args[]){
                student obj=new student();
                obj.setData();
                obj.show();
                student obj1=new student();
                obj1.setData(1,67,"vinu");
                obj1.show();
        }
}
```

```
C:\corejava\oops>javac demoOverload.java
C:\corejava\oops>java  demoOverload
Student RollNo.=90
Student Name.=raja
Student RollNo.=89

Student RollNo.=1
Student Name.=vinu
Student RollNo.=67
```

//Example of overloading.

```
class Room {
        int length;
        int width;
        int area;
        void setData(int l,int w) {
                length=l;
                width=w;
        }
        void setData(int x) {
        length=width=x;}
        void show() {
                area=length*width;
                System.out.println("\n\nArea of rectangle="+area);
        }
}//end of Room class

class RoomArea {
        public static void main(String args[]) {
                Room obj=new Room();
                obj.setData(25,15);
                obj.show();
                Room obj1=new Room();
                obj1.setData(20);
                obj1.show();
        }
}
```

C:\corejava\oops>javac RoomArea.java

C:\corejava\oops>java RoomArea

Area of rectangle=375

Area of rectangle=400

2. Overriding:(Same methods name and same parameter)

There may be occasion where we want an object to respond to the same method but have different behaviour when that method is called. That means we should override the methods defined in the superclass. This is possible by defining a method in a subclass that has the same name, same signature and same return type as methods in the super class.

```
//Example of overriding
class Super {
        int x;
        Super(int x) {
                this.x=x;
        }
        void show( ) {
                System.out.print("X="+x);
        }
} //End of Super class

class Sub extends Super {
        int y;
        Sub(int x,int y) {
                super(x);
                this.y=y;
        }
        void show( ) {
                System.out.println("X="+x);
                System.out.print("Y="+y);
        }
}// End of Sub class
class override {
        public static void main(String args[]) {
                Sub obj=new Sub(100,130);
                obj.show( );
        }
}
```

OUTPUT:-

```
C:\corejava\oops>javac override.java
C:\corejava\oops>java override
X=100
Y=130
```

Difference between method overloading and method overriding

Method Overriding	Method Overloading
1. In method overriding methods have same name and have the same parameter list as a super class method.	1. In method overloading methods have same name but have different parameter lists.
2. Appear in subclass only.	2. Appear in the same class or a subclass.
3. Inheritance concept involves.	3. Inheritance concept is not compulsory.
4. have the same return type as a super class method.	4. Can have different return types.
5. Late binding or redefine a method is possible.	5. Not possible.

Abstract class

An abstract class is a class that has at least one abstract method, along with concrete methods.
An abstract method is a method that is declared with only its signature, it has no implementation. That means method without functionality.
Since abstract class has at least one abstract class has at least one abstract method, an abstract class cannot be instantiated.

Characteristics of Abstract class

1. There can be no objects of an abstract class.
2. An abstract class cannot be directly instantiated with the **'new'** operator.
3. **abstract** keyword use as preface in class declaration to create a abstract class.
4. **abstract** keyword use as preface in method declaration to create a abstract method.
5. An abstract class is not fully defined.
6. An abstract class provide the frame for subclasses and subclass which extends super abstract class have must functionality for abstract method of super abstract class.

Example # 1

```
abstract class A {
        abstract void callme(); // this is abstract method
        void callmetoo() // this is concrete method
        {
        System.out.println("this is concrete method of abstract class");
        }
}
class B extends A  {
        void callme() {
                System.out.println("give the functionality of super class abstract method");
```

```
	}
}
class abstractDemo {
	public static void main(String args[]) {
		B obj=new B();
		obj.callme();
		obj.callmetoo();
	}
}
```

Example # 2

```
abstract class shape {
	int a,b;
	shape(int x,int y) {
		a=x;
		b=y;
	}
	abstract int area();
}

class Triangle extends shape {
	Triangle(int x,int y) {
		super(x,y);
	}
	int area() {
		return (a*b)/2;
	}
}
class Rectangle extends shape {
	Rectangle(int x,int y) {
		super(x,y);
	}
	int area() {
		return a*b;
	}
}

class abstractDemo2 {
	public static void main(String args[]) {
	// shape obj=new shape(10,29);  it is not possible bez
	Rectangle r1=new Rectangle(12,3);

	/*System.out.println("Area of Rectangle is="+r1.area());*/
	/* suppose we want to call area method of super abstract class.
	so call it with the help of reference variable of superclass which
	hold the object of subclass */
```

```
        shape s;
        s=r1;
        System.out.println("Area of Rectangle is="+s.area());
        Triangle t1=new Triangle(12,3);
        System.out.println("Area of Triangle is="+t1.area());
        }
}
```

Super Keyword:

Whenever a sub class needs to refer to its immediate superclass, we make use of the keyword **super.** The subclass constructor uses the keyword **super** to invoke variable or method of the super class.

The keyword **super** is used subject to the following condition :-

- super may only be used with in a subclass constructor method.
- The call to superclass constructor must appear as the first statement with in the subclass constructor.
- The parameter in the super call must match the order and type of the instance variable declare in the superclass.all must match the order and type of the instance variable declare in the superclass.

Example ##1

```
class S {
        int x;
        S(int x) {
                this.x=x;
        }
void show() {
        System.out.println("x="+x);
        }
} // end of super class

class Sub extends S {
        int y;
        Sub(int x,int y) {
                super(x); // superclass constructor call in sub class constructor by using the
                super key.
                this.y=y;
        }
        void show() // show method override here.
        {
                System.out.println("Super class variable x="+x);
                System.out.println("Sub class variable y="+y);
        }
}
```

```
class override {
        public static void main(String args[]) {
                Sub obj=new Sub(100,50);
                obj.show();
        }
}
```

Example # 2

```
// this is the prg for invoking superclass variable in subclass by using  super keyword.

class A {
        int a=100;
         void show() {
                System.out.println("Method in super class");
                System.out.println("value of a in super class="+a);
        }
} // end of super class
class B extends A {
        int a=200;
        void show() {
                System.out.println("Method in sub class");
                System.out.println("value of a in SUB class="+a);
                System.out.println("value of a in super class(invoke by super)="+super.a);
        }
}
class SuperDemo {
        public static void main(String args[]) {
                B obj=new B();
                obj.show();
        }
}
```

Example # 3

```
// this is the prg for invoking superclass constructor in subclass by using super keyword.
class Room {
        int length;
        int breath;
        Room(int x,int y) {
                length=x;
                breath=y;
        }
        int area() {
                return length*breath;
        }
}// end of superclass
class vol extends Room {
        int height;
```

```
        vol(int x,int y,int z) {
                super(x,y); // invoke superclass constructor.
                height=z;
        }
        int volume() {
                return length*breath*height;
        }
}//end of subclas.
class SuperDemo {
        public static void main(String args[]) {
                vol r1=new vol(2,2,3);
                System.out.println("Area="+r1.area());
                System.out.println("Volume="+r1.volume());
        }
}
```

this keyword:-

All instance methods received an implicit agument called **this**, which may be used inside any method to refer to the current object, i.e. the object on which method was called. Inside an instance of method, any unqualified reference is implicitly associated with the this reference.

Characteristics of this keyword.

1. when you need to pass a reference to the current object as an argument to another method.
2. when the name of the instance variable and parameter is the same, parameter hides the instance variable.

this and super keyword

1. Both are used for object reference.
2. “this” reference is passed as an implicit parameter when an instance method is invoked. It denotes the object on which the method is invoked. “super” can be used in the body of an instance method in a subclass to access variables/methods of superclass.
3. “this” is used in the constructor then it should be the first statement in the constructor, it is used to invoked the overloaded constructor of the same class. “super” is used in the constructor then it should be the first statement in that constructor, it is used to invoke the immediate super class constructor.
4. Subclass without any declared constructor would fail it the super class doesn’t have default constructor.
5. “this” and “super” statements cannot be combined.

Final Keyword

Final is a keyword which can be used to create constant value.

Basically final keyword used for 3 purpose.

1. It can be used to create the equivalent of a named constant.
2. using final to prevent from Method Overriding.
3. Using final to prevent from class inheritance.

1. It can be used to create the equivalent of a named constant

final double PI=3.14;

2. Using final to prevent from Method Overriding.
When we want to prevent method from occurring or to disallow a method from being overriding, specify final as a modifier at the start of its declaration. If methods declare as final cannot be overridden.

Example

```
class A {
        final void show() {
                statement 1;
                statement 1;
        }
}
class B extends A {
        void show() {
                // Error ! cannot override the super class method bez final keyword.
        }
}
```

3 Using final to prevent from class inheritance.
Sometimes you will want to prevent a class from being inherited for that purpose, the class declare with final key word. Declaring a class as final implicitly declare all of its methods as final too.

```
final class A {
        final void show() {
                statement 1;
                statement 1;
        }
}
class B extends A // it is not possible because Class A is declared as final.
{
        void show() {
                // Error ! cannot override the super class method bez final keyword.
        }
}
```

What is an Arrays?

An array is a finite set of elements of homogeneous data types. The elements of array are accessed by index.
Arrays index starts with 0 in java. In java arrays are treated as objects.

Creating an Array:->
Creation of array involves the three steps.

1. **Declare the array.**

There are two ways to declare an arrays

- <*data type*>[] arrayvariablename;

example:-int[] a;

- <*data type*> arrayvariablename[];

example:-int a[];

Remember: we do not enter the size of array in the declaration. you cannot assign the values to the array unless the array is created, because array is an object in java it should be create using new operator.

2. **create location into the memory.**

After decalartion an arrays we needed to create it in the memory. Java allows as to create arrays using new operator.

```
Arrayname=new <data type>[size];
```

Example:- a =new int[5];
It is array holds the 5 integer values.

➢ **It is possible to combine the two steps(declaration and creation) into one as follows:-**

int a[]=new int[5]; OR int[] a =new int[5];

3. **put values into the memory location.(Initialization of Arrays)**

The final steps is to put values into the array created. This process in known as initialization. This is done using the array subscripts as shown below.

```
Arrayname[subscript]=values;
Example:--  number[0]=67;
            number[1]=7;
            number[2]=6;
            number[3]=37;
```

Note:-we can also initialize arrays automatically in the same way as the ordinary variables when they are declare.

Data Type ArrayName*[]={*list of values};
***Example::-int numbers*[]={2,4,5,6,7};**

Array length:-

In java all arrays store the allocate size in a variable named length. We can access the size of array using following syntax.

int variableName=array.length;

example :- int n=number.length;

here n assign value 5;

Some key feature of Array in JAVA:-

- *Once array is created its size cannot be changed.*
- *The size of an array give by array property called length.*
- *Array has index from 0 to length-1.*
- *By default array elements are initialized with zero or NULL.*
- *When user access the any element other then its range, java compiler give ArrayOutOfBoundsException.*

Example # 1

```
//EXAMPLE OF SINGLE DIMENSIONAL ARRAY FIND THE AVG OF ARRAY ELEMENTS.
class Avgarraydemo {
    public static void main(String args[]) {
            int a[]={12,12,12,12,12};
            double avg=0;
            int i;
        for(i=0;i<a.length;i++) {
            avg=avg+a[i];
        }
    System.out.println(i);
    avg=avg/i;
    System.out.print("Average="+avg);
    }
}
```

Example # 2

```
//Example for set elements of array in ascending order and descending order.
class AscDsc {
    int i,j,a[]={22,33,21,5,26};
    int l;
    void a() {
            l=a.length;
            for(i=0;i<l;i++) {
                    System.out.println("Before the arrange array is a["+i+"]="+a[i]);
            }
            System.out.println("After sorting array in Ascending order::");
            for(i=0;i<l;i++) {
                    for(j=0;j<l-1;j++)
```

```
                                        if(a[j]>a[j+1]) {
                                                int temp=a[j+1];
                                                a[j+1]=a[j];
                                                a[j]=temp;
                                                }
                        }
                }



        void d() {
                //l=a.length;
                System.out.println("\nAfter sorting array in Dscending order::");
                        for(i=0;i<l;i++) {
                                for(j=0;j<l-1;j++)
                                        if(a[j]<a[j+1]) {
                                                int temp=a[j+1];
                                                a[j+1]=a[j];
                                                a[j]=temp;
                                        }
                        }
                }

        void show() {
                for(i=0;i<l;i++) {
                         System.out.print(a[i]+" ");
                }
        }

        public static void main(String agrs[]) {
                AscDsc b=new AscDsc();
                b.a();
                b.show();
                b.d();
                b.show();
        }
}
```

TWO DIMENSIONAL

Example # 3

```
//Example to find the order of matrix.
class Text {
	public static void main(String args[]) {
		int[][] a=new int[7][9];
		System.out.println("a.lenght ="+a.length);
		System.out.println("a[0].lenght ="+a[0].length);
	}
}
//Example to initialize the value of matrix and print the matrix at output.
class Text1 {
	public static void main(String args[]) {
		int[][] a={{77,33,88},
			   {11,55,22,99},
			  {66,44}};
		for(int i=0;i<a.length;i++) {
			System.out.println("\t"+a.length+"hai==i="+i);
			for(int j=0;j<a[i].length;j++)
				System.out.print("\t"+a[i][j]);
			System.out.println();
		}

	}
}
```

Example # 4

```
//Example of two dimensional array MATRIX ADDITION, when values of matrix are initialized.
class MatrixM {
	public static void main(String args[]) {
		int a[][]={{2,2},{2,2}};
		int b[][]={{1,1},{1,1}};
		int c[][]=new int[2][2];
		int n=a.length;
		int m=a[0].length;
		if(n==m){
			try{
				for(int i=0;i<n;i++) {
					for(int j=0;j<m;j++) {
						c[i][j]=a[i][j]+b[i][j];
						System.out.print("\t"+c[i][j]);
					}
				System.out.println();
```

```
                        }
                    }
                    catch(Exception e){System.out.print("Some Error"+e);}
                }
                else{
                    System.out.print("\n Addition of matrix is not possible");
                }
        }
}
//this is the program for matrix addition.
```

Example # 5

//Example of two dimensional array MATRIX ADDITION When value of matrixs are given by the user.

```
import java.io.*;
class M {
        int a[][]=new int[3][3];
        int b[][]=new int[3][3];
        int c[][]=new int[3][3];
        int i,j,k;
        void getdata() {
                DataInputStream in=new DataInputStream(System.in);
                System.out.println("Enter the matrix value of a:");
                try{
                        for(i=0;i<3;i++)
                                for(j=0;j<3;j++)
                                        a[i][j]=Integer.parseInt(in.readLine());
                }
                catch(Exception e){}
                System.out.println("Enter the matrix value of b:");
                try{
                        for(i=0;i<3;i++)
                                for(j=0;j<3;j++)
                                        b[i][j]=Integer.parseInt(in.readLine());
                }
                catch(Exception e){}
        }

        void cal() {
                for(i=0;i<3;i++)
                        for(j=0;j<3;j++)
                                c[i][j]=a[i][j]+b[i][j];
        }
        void show() {
                System.out.println("addition of matrix is::");
```

```
            for(i=0;i<3;i++) {
                  for(j=0;j<3;j++)
                        System.out.print(" "+c[i][j]);
            System.out.println();
            }
      }
      public static void main(String args[]) {
            M obj=new M();
            obj.getdata();
            obj.cal();
            obj.show();
      }
}
```

Vector CLASS:-

In C and C++ generic utility function that with variable arguments can be uses to pass different arguments depending upon the calling situation. Java does not support the concept of variable arguments to a function. This feature can be achieved in java through the use of the Vector class. Which one find in the java.util. package.

Vector class can be used to create a generic dynamic array known as vector that can hold objects of any type and any number.

Vector are created by the help of constructor.

Constructor

Vector list=new Vector() // default Constructor.
Vector list=new Vector(int Size);

Advantages of Vector class over the arrays:

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

Methods-

1. **list.addElement(String item);** => adds the item into the list at the end.
2. **list.elementAt(int n);** => Gives the name of element which hold n^{th} position.
3. **list.size();** => Gives the number of objects present in list.
4. **list.removeElement(String item);** => Remove the specific item in the list.
5. **list.removeElementAt(int n);** => Remove the item in specific position.
6. **list.removeAllElements();** => Remove all elements in the list.
7. **list.copyInto(array_variable);** => Copies all item of list into array.
8. **list.insertElementAt(String item,int n);** => insert the item at n^{th} position
9. **list.indexOf(String item);** => return the index number of the object.

Example # 1

```
//Example of Vector class using the print of list of element of vectors.
import java.util.*;
class vdemo {
        public static void main(String args[]) {
                Vector list=new Vector();
                list.addElement("One");
                list.addElement("Two");
                list.addElement("Four");
                list.addElement("Three");
                list.addElement("Five");
                System.out.println(list);
                int n=list.indexOf("Three");
                list.insertElementAt(list.elementAt(n),2);
                System.out.println(list);
                int l=list.size();
                list.removeElementAt(l-2);
                System.out.println(list);
                int m=list.size();
                String str[]=new String[m];
                list.copyInto(str);
                for(int i=0;i<m;i++) {
                        System.out.println(str[i]);
                }
        }
}
```

output

```
C:\>java vdemo
[One, Two, Four, Three, Five]
[One, Two, Three, Four, Three, Five]
[One, Two, Three, Four, Five]
One
Two
Three
Four
Five
```

Example # 2

//Example of vector when value gives through args[];

```
import java.util.*;
class Vectordemo {
        public static void main(String args[]) {
                Vector list=new Vector();
                int n=args.length;
                for(int i=0;i<n;i++) {
                        list.addElement(args[i]);
                }
                list.insertElementAt("COBOL",2);
                int size=list.size();
                String a[]=new String[size];
                list.copyInto(a);
                System.out.println("List of languges are::");
                for(int i=0;i<size;i++) {
                        System.out.println(a[i]);
                }
        }
}

/*run
c:\corejava\class>java Vectordemo Ada BASIC FORTRAN JAVA C++


OUTPUT
List of languges are::
Ada
BASIC
COBOL
FORTRAN
JAVA
C++
*/
```

Using objects as parameter of methods.

So far we have only been consider the simply types of method where we give some parameter for the iteration. But here we consider passes objects as parameter to the methods .

IMP..

```
// Example checks objects are equals are not true and false value should be return.
class Text {
        int a,b;
        Text(int x,int y)
                {a=x;b=y;}
        boolean cal(Text obj) {
                 if(obj.a==a && obj.b==b)
                        return true;
                else
                        return false;
        }
}//end of  class Text

class objMethodDemo {
        public static void main(String args[]) {
                Text ob1=new Text(100,22);
                Text ob2=new Text(11,11);
                Text ob3=new Text(100,22);
                System.out.println("ob1==ob2 is"+ob1.cal(ob2));
                System.out.println("ob1==ob3 is"+ob1.cal(ob3));
        }
}//end of class
```

Wrapper classes

Vector class cannot handle the primitive data types like int, float, long, char and double.
So primitive data types may be converted into objects using the **wrapper classes.** Wrapper classes are defined in the java.lang package. Each of Java's eight primitive data types has a corresponding class called a wrapper class.

Primitive Data Type	wrapper Class
1. byte	1. Byte
2. Int	2. Integer
3. Short	3. Short
4. Long	4. Long
5. Float	5. Float
6. Double	6. Double
7. Boolean	7. Boolean
8. Char	8. Character

(I) Converting Primitive number to Object by using Constructor method.

```
int  i = 5;
Integer obj=new Integer(i);
```

(II) Converting Object to Primitive number by using datatypeValue() method.

```
Integer obj=new Integer(5);
int  i = obj.intValue();
```

(III) Converting Object to String by using toString() method

```
Integer obj=new Integer(5);
String str =obj.toString();
```

(IV) Converting String to Object by using valueOf() static method of wrapper class.

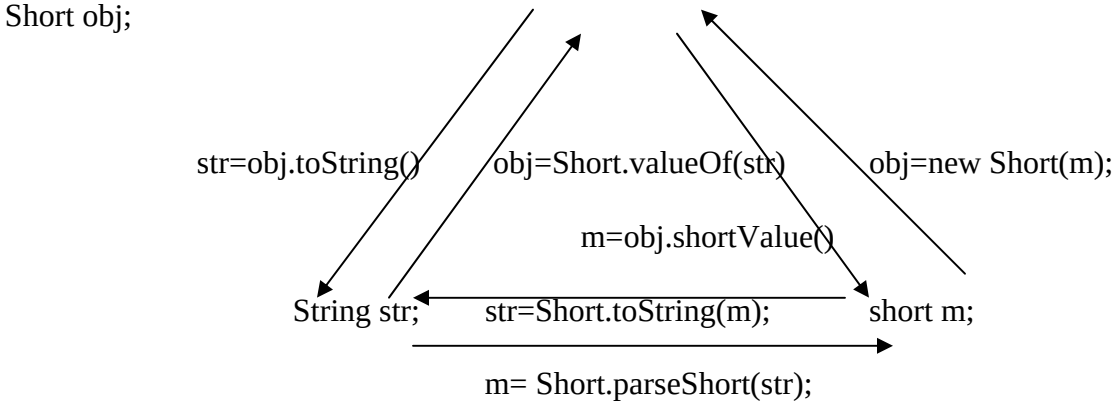
```
String str =”5”;
Integer obj =Integer.valueOf(str);
```

(V) Converting Primitive number to String by using toString() static method

```
int i=5;
String str =Integer.toString();
```

(VI) Converting String to Primitive number by using Parsing() static method

```
String str=”5”;
int i=Integer.parseInt(str);
```



Example # 1

```
// this is prg for wrapper class which use the all 6 methods
class wdemo {
    public static void main(String args[]) {
        short m=12;
        //1.Convert number to object
        Short obj=new Short(m);
        System.out.println("Numerical object="+obj);

        //2.Convert object to number
        m=obj.shortValue();
        System.out.println("premative Numerical value="+m);

        //3.Convert object to string
        String s=obj.toString();
        System.out.println("String through object="+s);

        //4.Convert  string to object
        obj=Short.valueOf(s);
        System.out.println("object by string="+obj);

        //5.Convert Number to string
        s=Short.toString(m);
        System.out.println("String through number="+s);

        //6.Convert string to number
        m=Short.parseShort(s);
        System.out.println("number through string="+m);
    }
}
```

Example # 2

```
// this prg from addition of two value which enter as arguments
class wdemo1 {
    public static void main(String args[]) {
        int a,b,result;
        String s1,s2;
        if(args.length==0) {
            System.out.println("Enter two number at run time");
        }
        else {
            s1=args[0];
            s2=args[1];
            a=Integer.parseInt(s1);
            b=Integer.parseInt(s2);
```

```
                result=a+b;
                System.out.print(result);
            }
        }
    }
```

Example # 3

```
    import java.lang.*;
    class wrapperclass1 {
        public static void main(String args[]) {
                int n=59;
        System.out.println("convert number to string by  toString
                method :"+Integer.toString(n));
        System.out.println("convert number to binary by toBinaryString
                method:"+Integer.toBinaryString(n));
        System.out.println("convert number to Octal by toOctalString
                method:"+Integer.toOctalString(n));
        System.out.println("convert number to hexadecimal by  toHexString
                method:"+Integer.toHexString(n));
        System.out.println("12:"+Integer.toString(n,12));
        System.out.println("20:"+Integer.toString(n,20));
        n=Integer.parseInt("3b",16);
        System.out.println("decimal no of 3b:"+n);
        }
    }
```

Example # 4

```
    import java.io.*;
    import java.lang.*;
    class SimpleIntrest {
        public static void main(String args[]) {
            int year=0;
            float principal=0,rate=0;
            try {
                DataInputStream in=new DataInputStream(System.in);
                System.out.print("Enter the principal");
                String strp=in.readLine();
                principal=Float.parseFloat(strp);
                System.out.println("Enter the rate");
                String strr=in.readLine();
                rate=Float.parseFloat(strr);
                System.out.println("Enter the year");
                String stry=in.readLine();
                year=Integer.parseInt(stry);
                }
```

```
        catch(Exception e){}

        float value=si(principal,rate,year);
        line();
        System.out.println("Simple intrest="+value);
        line();
    }

    static float si(float p,float r,int y) {
        float sum;
        sum=p*r*y/100;
        return sum;
    }
    static void line() {
        for(int i=0;i<15;i++) {
                System.out.print("=");
        }
        System.out.println();
    }
}
```

String and StringBuffer Class

Both String and StringBuffer class contains the sequence of 16 bits Unicode character.
String represents a sequence of character. The easiest way to represent a sequence of character in java is by using a char array.

Example :-

```
char  charArr[]= new char[4];
charArr={'J','A','V','A'};
```

In java, strings are class objects and implemented by using two classes, namely String and StringBuffer. A java string is an instantiated object of the String Class. Java strings as compare to C and C++ strings are more reliable. This is basically due to C's lack of bound checking .
A java string is not a character array and is not terminated by null character. Strings may be declared and creates as follows.

Syntax:=

```
String string_variable;
String_variable = new String("String");
```

Example:==

```
String firstName;
firstName=new String("Anil");
```

These two statements are combined together as follows:-

String firstName= new String("Anil");

It is possible toget the length of string by using the length() method of String class. Java strings can be concatenated using the + operator.

System.out.println(firstName+"Kumar");

String Methods:-

1. ***String s2=s1.toUpperCase();*** Convert given string into uppercase.
2. ***String s2=s1.toLowerCase();*** Convert given string into lowercase.
3. ***String s2=s1.replace('x','y');*** Replace all 'x' char with 'y'.
4. ***String s2= s1.tring();*** Remove white space.
5. ***s1.equals(s2);*** Return 'true' if both are equal.
6. ***s1.equalsIgnoreCase(s2);*** Return 'true' if both are equal ignoring the case of characters.
7. ***s1.length();*** Returns the no. of character present in string
8. ***s1.charAt(n);*** Returns the n^{th} character of string s1.
9. ***s1.compareTo(s2);*** ***R***eturns negative if s1>s2, return positive s1>s2, and zero s1=s2.
10. ***s1.concat(s2);*** concatenates s1 and s2; + operator is also used is the same manner.
11. ***s1.substring(n);*** Returns substring from n^{th} character.
12. ***s1.substring(n,m);*** Gives substring from n^{th} character and ending m^{th}
13. ***int s1.indexOf(char ch)***; Returns the index of a character ch in string
14. ***s1.toCharArray();*** Convert string into char array.

Example # 1

```
class alpha {
        public static void main(String args[]) {
                String a="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                System.out.println(a);
                System.out.println("This string contains"+a.length()+"characters");
                System.out.println("The character in 4 index="+a.charAt(4));
                System.out.println("The index no. of z is="+a.indexOf('Z'));

                String s=a.toLowerCase();
                System.out.println(s);
                int i=a.indexOf('S');
                System.out.println(i);
                System.out.println(a.replace('S','#'));

        }
}
```

output

```
C:\>java alpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ
This string contains26characters
The character in 4 index=E
The index no. of z is=25
abcdefghijklmnopqrstuvwxyz
18
ABCDEFGHIJKLMNOPQR#TUVWXYZ
```

Example # 2

```
class st {
    public static void main(String args[]) {
            String s1="DishaCollege";
            String s2=s1.toUpperCase();
            System.out.println(s1);
            System.out.println(s2);
            String s3=s1.replace('D','M');
            System.out.println(s3);
            System.out.println( s1.equals(s2));
            System.out.println( s1.equalsIgnoreCase(s2));
            int n=s1.length();
            System.out.println(n);
            char ch=s1.charAt(4);
            System.out.println(ch);
            System.out.println(s1.compareTo(s2));
            System.out.println(s1.concat(s2));
            String sub1=s1.substring(4);
            String sub2=s1.substring(4,8);
            System.out.println(sub1);
            System.out.println(sub2);
            char c[]=new char[n];
            c=s1.toCharArray();
            for(int i=0;i<n;i++) {
                    System.out.println(c[i]);
            }
    }
}
```

Example # 3

```
// this is prg for check given name is palindrome or not
class prg1 {
    public static void main(String args[]) {
            String name=args[0];
            boolean b=true;
            int n=name.length();
            char a[]=new char[n];
            a=name.toCharArray();
            int l=n-1;
            for(int i=0;i<n;i++) {
                  if(a[l]!=a[i])   {
                       b=false;
                       break;
                  }
             l--;
            }
            if(b)
                       System.out.println("name is palindrome");
            else
                       System.out.println("Name is not palindrome");
    }
}
```

output

```
C:\>java prg1 malayalam
name is palindrome

C:\>java prg1 raja
Name is not palindrome
```

String Array:-

We can also declare and use array that contain strings.

String itemArray[]= new String[];
itemArray={“ram”,”sita”,”gita”};

Example # 1

```
// this is prg for arrange the names is alphabetical order.
class prg2 {
    public static void main(String args[]) {
            String a[]={"Rama","Raju","Ajay","Vijay","Kiran"};
            int n=a.length;
             for(int i=0;i<n;i++)   {
               for(int j=i+1;j<n;j++)   {
                       if(a[i].compareTo(a[j])>0)
                      {
```

```
                        String temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                    }
                }
            }
            for(int i=0;i<n;i++) {
              System.out.println(a[i]);
              }
    }
}
```

output

```
C:\>java prg2
Ajay
Kiran
Raju
Rama
Vijay
```

SubString

A substring is a string whose characters form a contiguous part of another string. The string class includes a substring() method for extracting substrings.

Example # 1

```
class SubString {
        public static void main(String args[]) {
          String a="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
          System.out.println("1."+a);
          System.out.println("2. This substring from index 4 to index 8 is:::"+a.substring(4,8));
          System.out.println("3. substring from index 4 to index 4 is:::"+a.substring(4,4));
          System.out.println("4. substring from index 4 to index 5 is:::"+a.substring(4,5));
          System.out.println("5. substring from index 0 to index 8 is:::"+a.substring(0,8));
          System.out.println("6. substring from index 8 to end is:::"+a.substring(8)
        }
}
```

output

C:\>java SubString
1.ABCDEFGHIJKLMNOPQRSTUVWXYZ
2. This substring from index 4 to index 8 is:::EFGH
3. substring from index 4 to index 4 is:::
4. substring from index 4 to index 5 is:::E
5. substring from index 0 to index 8 is:::ABCDEFGH
6. substring from index 8 to end is:::IJKLMNOPQRSTUVWXYZ

StringBuffer Class

An instance of Java's StringBuffer Class represents a string that can be dynamically modified. StringBuffer is a peer class of String class. While String class creates strings of fixed_ length, StringBuffer class creates strings with flexible length that can be modified in terms of both length and context. We can insert characters and substrings in the middle of a string or append another string to the end.

StringBuffer class has two constructor.

StringBuffer() => 16 bits.
StringBuffer(int i) => fixed by int value.
StringBuffer("Str"); => 16+str(length of str).

Methods.

1. s1.setCharAt(n,'x');
2. s1.append(s2);
3. s1.insert(n, s2);
4. s1.setLength(n)
5. s1.capacity();
6. s1.insert(int index, String str);
7. s1.insert(int index, char ch);
8. s1.insert(int index, object obj);

Example # 1

```
// this prg demonstrate how to modify the content of a buffer.
class sprg1 {
        public static void main(String args[]) {
                StringBuffer buf=new StringBuffer();
                buf.append("It was the best of time");
                System.out.println("buf=="+buf);
                System.out.println("buf.length()="+buf.length());
                System.out.println("buf.capacity()="+buf.capacity());
                String str=buf.toString();
```

```
                int n=str.indexOf('b');
                System.out.println("index position of 'b'="+n);
                buf.setCharAt(11,'w');
                System.out.println("buf=="+buf);
                buf.setCharAt(12,'o');
                System.out.println("buf=="+buf);
                buf.insert(13,'r');
                System.out.println("buf=="+buf);
        }
}
```

output

```
C:\>java sprg1
buf==It was the best of time
buf.length()=23
buf.capacity()=34
index position of 'b'=11
buf==It was the west of time
buf==It was the wost of time
buf==It was the worst of time
```

Example # 2

```
class sprg2 {
        public static void main(String args[]) {
                StringBuffer buf=new StringBuffer("it was the age of wisdom.");
                System.out.println("buf=="+buf);
                System.out.println("buf.length()=="+buf.length());
                System.out.println("buf.capacity=="+buf.capacity());
                String str=buf.toString();
                System.out.println("str=="+str);
                int n=str.indexOf('f');
                System.out.println(n);
                buf.append(",, "+str.substring(0,18)+"foolish");
                System.out.println("buf=="+buf);
                System.out.println("buf.length()=="+buf.length());
                System.out.println("buf.capacity=="+buf.capacity());
        }
}
```

Output

```
C:\>java sprg2
buf==it was the age of wisdom.
buf.length()==25
buf.capacity==41
str==it was the age of wisdom.
16
```

buf==it was the age of wisdom.,, it was the age of foolish
buf.length()==53
buf.capacity==84

Interface

An interface is a way of describing 'what' classes should do, without specifying how they should do it. A class can implements one or more interfaces. You can then use objects of these implementing classes anything that conformance to the interface is required.

CAUTION

In the interface declaration the method was not declared "public" because all methods in an 'interface' are automatically public. However when implementing the interface, you must declare the methods as 'public'.
Otherwise the compiler assumes that the method has package visibility – the default for a class. Then the compiler complains that you try to supply a weaker access privilege.

Properties of interface

Interface are not classes. You can never use the 'new' operator to instantiate an interface. However, you can't construct interface objects, therefore you can still declare interface variable.

Difference between Interface and Abstract Class

```
interface inner1 {
	void meth1();
}

interface inner2 {
	void meth2();
}

class client implements inner1,inner2 {
	int a,b;
	public void meth1() {
		System.out.println("Result of inner1="+(a+b));
	}

	public void meth2() {
		System.out.println("Result of inner2="+(a*b));
	}
	void show() {
		System.out.println("Implementing class also define the other methods");
	}
}
```

```
class interdemo {
        public static void main(String args[]) {
                client obj=new client();
                obj.a=8;obj.b=4;
                obj.meth1();
                obj.meth2();
                inner1 ref;
                ref=obj;
                ref.meth1();
                ref.meth2();// this method is invalid because interface reference variable invoke
                only interface method.
                obj.show();
        }
}
```

OUTPUT

```
C:\>javac interdemo.java
interdemo.java:36: cannot find symbol
symbol  : method meth2()
location: interface inner1
ref.meth2();// this method is invalid because interface reference variable invok
e only interface method
```

.

Inheritance:-

Anonymous classes:-

There are occasion where you need to define a class for which you will only ever want to define one object in your program and only use for the object is to pass it directly as an argument to a method. In this case as long as your class extends an exiting class, or implements an inter face you have the option of defining the class as an anonymous class appear in the next expression in the statement where you create and use the object of class, so the there is not necessity to provide a name for the class.
If the anonymous class extends an exiting class the syntax is much the same.
An anonymous class can definition is short and simple you should not use the approach to define classes of any complexity, as it will make the code very difficult to understand.

Example # 1

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

```
class demo extends Frame implement WindowListener {
demo() {
        setTitle("demo of anonymous class");
        setSize(300,400);
        setVisible(true);
        addWindowListener(new windowAdaptor(){
                public void windowClosing(WindowEvent we){
                        System.exit(0);
                        }
                });
                validate();
        }
public static void main(String args[]) {
        demo fr=new demo();
}
}
```

packages and interfaces:-

Definition of CLASSPATH:- "The classpath is the collection of all base directories whose subdirectories can contain class files."

Class path depends on your compilation environment.

Setting the classpath:

1. on the win95/win98 Operating System:

Add the line [set classpath=c:\user\classfiledir;] and make sure that not to put any space around the "=" in autoexec.bat file.

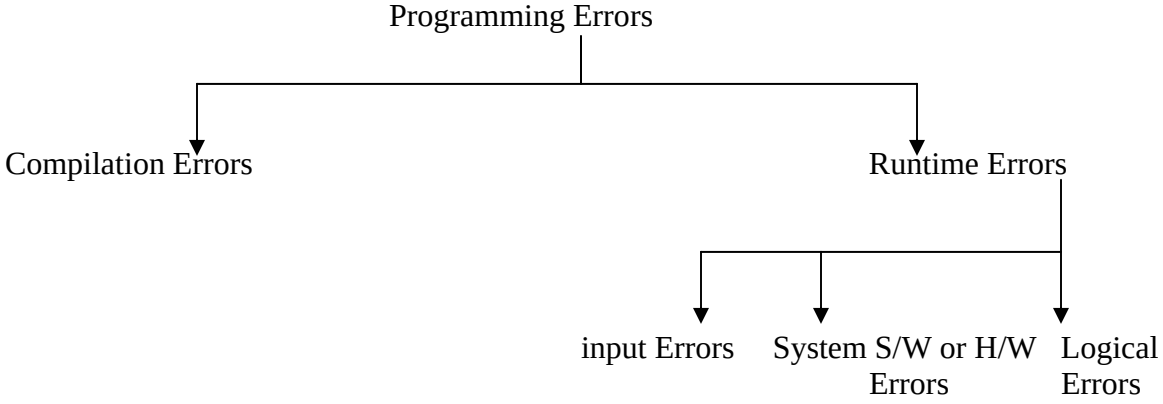
2. On the winNT/win2000:-

Open the control Panel. Then open the system icon and select the Environment tab. In the variable field, type classpath. In the value field, type the following desired class path such us [c:\user\classfilesdir].

Packages

Java provides a powerful means of grouping related classes and interfaces together in a single unit: packages. Put simply, packages are groups of related classes and interfaces. Packages provide a convenient mechanism for managing a large group of classes and interfaces while avoiding potential naming conflicts. The Java API itself is implemented as a group of packages.

Exception Handling:-



Compilation error:- that errors occur during compilation are called ***compilation errors or syntax error.*** *These errors are easy to detect because the compiler give the indication where the errors came from and why they are there. It tells the reasons for them.*
Example:-

```
class showerror {
        public static void main(String args[]) {
                i=20;   //missing the datatype int
                System.out.println("a"+a);
        }
}
```

Errors: undefined variable i;

Runtime Errors:- The errors that occur during the execution of a program is called ***Runtime Errors.*** *These errors cause a program to terminate abnormally.*

1) *Input Errors:- An input errors occurs when the user enters an unexpected input value that the program cannot handle.*
   example:
   a) It the program expects to read an integer, but the user enters a string.
   b) Typing errors, incorrect non-exiting URL's.
2) *System Software and Hardware Errors:- These kind of Errors occur rarely happen. However unreliable system software and hardware malfunction can cause a program to abort. System Software Errors are beyond the programmers control, and there is little you can do when you encounter them.*
   example:
   a) System Hanging
   b) Hard Disk crashes.
   c) Space not available in main memory.
   d) Device Errors.
3) *Logical Errors:-Logical Errors cause the program to either generate incorrect results or terminate abnormally. The errors are called bugs. The process of finding logical errors is called debugging.*
   a) if file name and class name are different

b) File you try to open is not exist.
c) An operand is not in the legal prescribed for operations. Ex. array element out of bound and dividing with zero.

Exception Handling

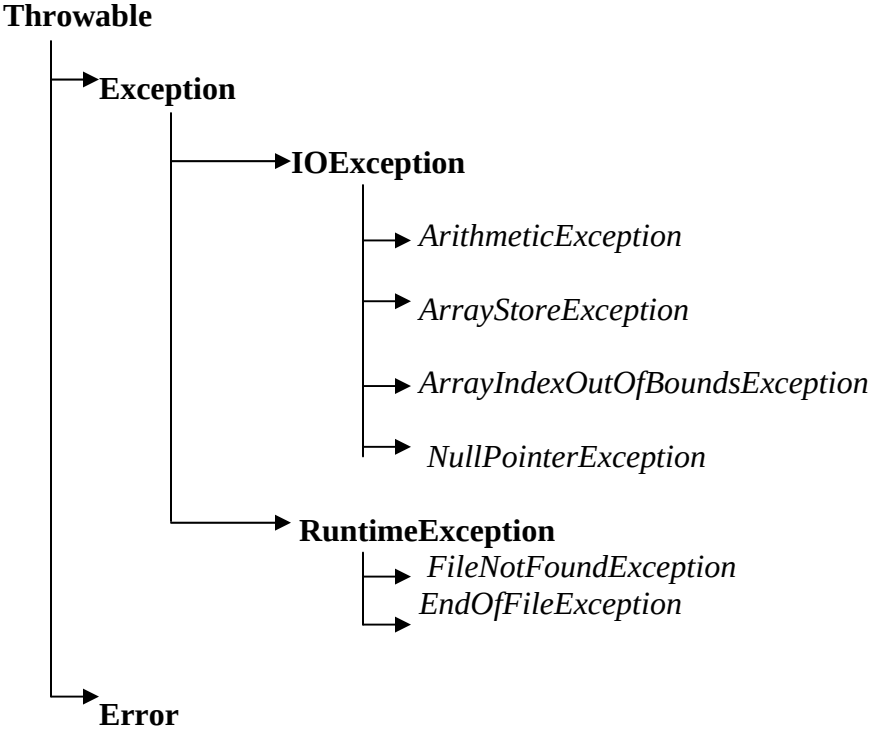
An exception is a condition that is caused by a runtime error in the program, When the java interpreter encountered an error. It create an exception object and throws it.
A java Exception is an object that describes an exceptional condition that has occurred in a piece of code. At some point the exception is caught and processed is called **Exception handling.**

Exception can be generated by the JVM or they are manually generated.

Exceptional handling mechanism perform the following task:

1. Find the problem (hit the exception).
2. Inform that an error has occurred (throw the exception)
3. Receive the error information (catch the exception)
4. Take corrective action (handle the exception).

Types of Exception or Hierarchy of Exception class:-
Exception is objects with hierarchical relationship. They have their own hierarchy. The root class of all the Exception classes is the "Throwable".
Java.lang.*;



Throwable:-
A java exception is an instance of the class derived from Throwable. This class is contained in java.lang package and subclasses of Throwable contained in various packages.

Exception Class:- This class describes the errors caused by programs. These errors can be caught and handled by the program. This is also the class that you will subclass to create our own custom exception types.

IOException : - This class describes errors related to Input / Output operation such as invalid input or inability to read from a file.

1. Arithmetic Exception:- Caused by math errors, such as divided by zero.
2. ArrayStroeException:- Caused when a programmer tries to store the wrong data types in an array.
3. ArrayIndexOutOfBoundsException:- Caused by bad array 'Indexes'.
4. NullPointerException:- Caused by referencing a null object.

RunTimeException :

1. FileNotFoundException: - Caused by an attempt to access a nonexistent file.
2. EndOfFileException:- Caused as inability to read from a file.

Error: It is not in our control. The Error class describe internal system errors, which rarely occur. Stack overflow is the best example of errors.

Java Exception handling is managed by five keywords.

1. try 2. catch 3. finally 4. throw 5. throws.

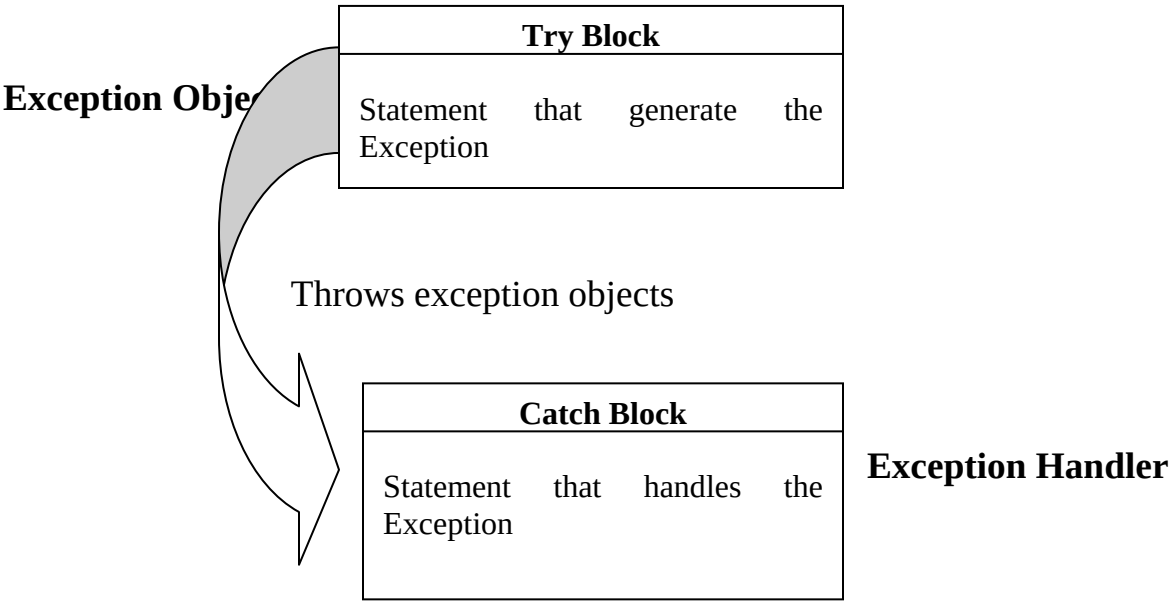
try – catch mechanism

1. In java keyword try to preface a block of code that is likely generate an error condition and 'throw' an exception. A catch block defined by catch handles it appropriate way.

2. The try block can have one or more than one statements that could generate an exception . If any one statement generate an exception, the remaining statement in the block are skipped and exception jumps to catch block

3. Every try statement should be followed by at least one catch statement /finally otherwise compilation error will occur.

4. The catch statement is passed a single parameter, which is reference to the exception object thrown by the try block.



Example # 1

```
//it is example of runtime error
class prg1 {
        public static void main(String args[]) {
                int a=10;
                int b=0;
                int c=a/b;
                System.out.println(a);
                System.out.println(b);
                System.out.println(c);//it is throw the Arithmetic Exception:/by zero
        }
}
```

output: *C:\corejava\exception>javac prg1.java*
C:\corejava\exception>java prg1
Exception in thread "main" java.lang.ArithmeticException: / by zero
at prg1.main(prg1.java:8)

Multiple catch Statements

It is possible to have more than one catch statement in catch block. When an exception in a try block is generated, the java treats the multiple catch statements like case in a switch –case statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statements will skipped.

Example # 2

```
class prg2 {
        public static void main(String args[]) {
                int a[]={5,10};
                int n=args.length;
                int c;
                try {
                        c=a[1]/n;
                        System.out.println("Value of c="+c);
                        c=a[2]+a[0];
                        System.out.println("Value of c="+c);
                }
                catch(ArithmeticException e) {
                        System.out.println("Error="+e);
                }
        catch(ArrayIndexOutOfBoundsException e){
                System.out.println("Error="+e);
        }
        }
}
```

Output

```
C:\>java prg2
Error=java.lang.ArithmeticException: / by zero

C:\>java prg2 1 2
Value of c=5
Error=java.lang.ArrayIndexOutOfBoundsException: 2
```

finally keyword:

If we want some code to be execute regardless of whether the exception occurs and whether is caught the code in the finally block is executed under all circumstances.

throw Keyword

You can manually throw an exception with the help of using throw keyword. In the method that has claimed the exception you can throw an object of the exception if the exception arises.

Syntax:-

```
throw  new MyException();
 OR
MyException e=new MyException();
throw e;
```

Example # 3

```
class MyException extends Exception {
        int detail;
        MyException(int a) {
                detail=a;
        }
        public String toString() {
                return "MyException["+detail+"]";
        }
}

class prg3 {
        void compute(int a) throws MyException {
                System.out.println("Called compute("+a+")");
                if(a>10) {throw new MyException(a);}
                System.out.println("Compute again");
        }

        public static void main(String args[]) {
                prg3 obj=new prg3();
        try {
                obj.compute(1);
                obj.compute(5);
                obj.compute(20);
                obj.compute(2);
        }
        catch(MyException e) {
                System.out.println("caught"+e);
        }
        }
}
```

Output

```
C:/>java  prg3
Called compute(1)
Compute again
Called compute(5)
Compute again
Called compute(20)
caughtMyException[20]
```

Example # 4

```
class BCAException extends Exception {
        BCAException(String message) {
                super(message);
        }
}
class tdemo {
        public static void main(String args[]) {
                int x=5;
                int y=1000;
                try     {
                        float z=(float)x/(float)y;
                        if(z<0.01) {
                                throw new BCAException("number is too small");
                        }
                }
                catch(BCAException e) {
                        System.out.println("Caught my exception"+e);
                        System.out.println(e.getMessage());
                }
        }
}
```

throws keyword

If method is capable of causing an exception that it does not handle it must specify this behavior so that caller of the method can guarded themselves against the exception by using throws keyword.

Syntax:-

Retrun_type **method_name**(Parameter list) throws exception List {
    body of the method.
}

Multithreaded Programming

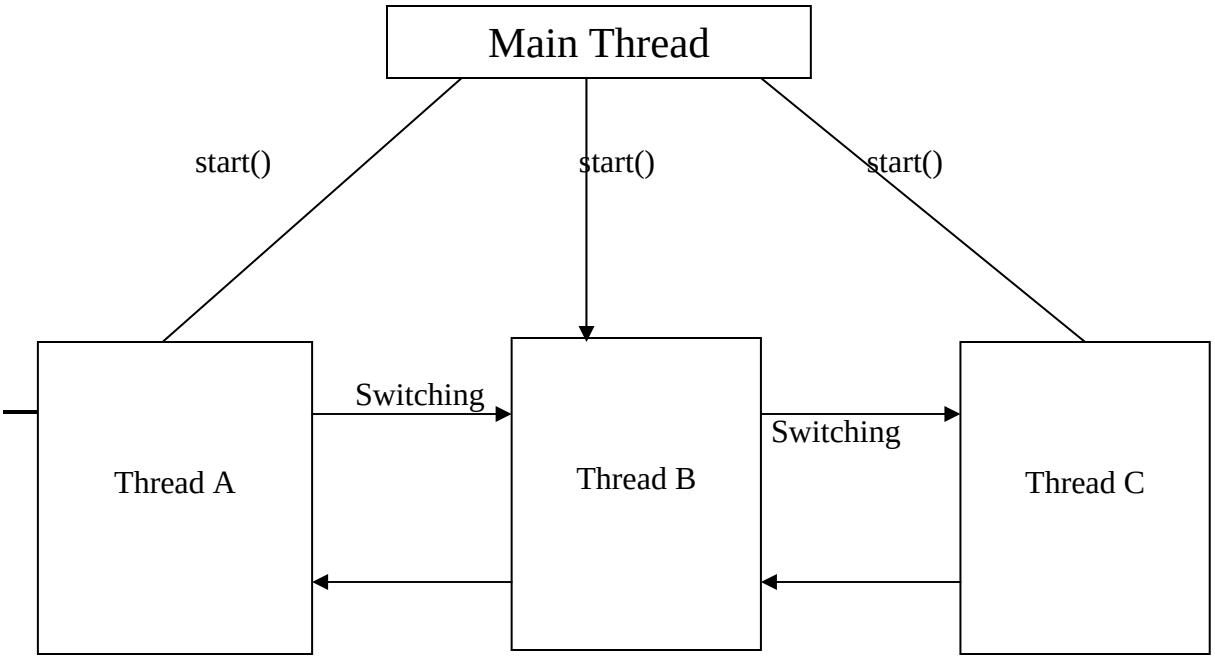
A typical program is executed sequentially and is single threaded. All modern Operating System may execute multiple programs simultaneously, even if there is only a single CPU available to run all the application. Multithreading allows multiple tasks to execute concurrently within a single program. The main advantage of multiple threads in a program is that it utilizes system resources(like CPU) better because other threads can grab CPU time when one line of execution is blocked. By using multithreading, you can create programs showing animation, play music, display
Document and download files from the network simultaneously.
Java was designed from the start to accommodate multithreading. Synchronized method to avoid collision in which more than one thread attempt to modify the same object at the same time.

What is Thread Model ?

Multithreading allows a running program to perform several task at the same time. Java uses threads to enable the entire environment to be asynchronous. It makes maximum use of CPU because idle time(waiting time) cam be kept to minimum. This helps to reduce the wastage of CPU life cycle.

The java runtime system depends on threads for many things and all class libraries are designed with multithreading in mind. A thread exists in several states.

- Running Thread
- Ready to run as soon as it gets CPU time
- Suspended thread
- Suspended thread can be resumed
- Thread can be blocked
- Thread can be terminated

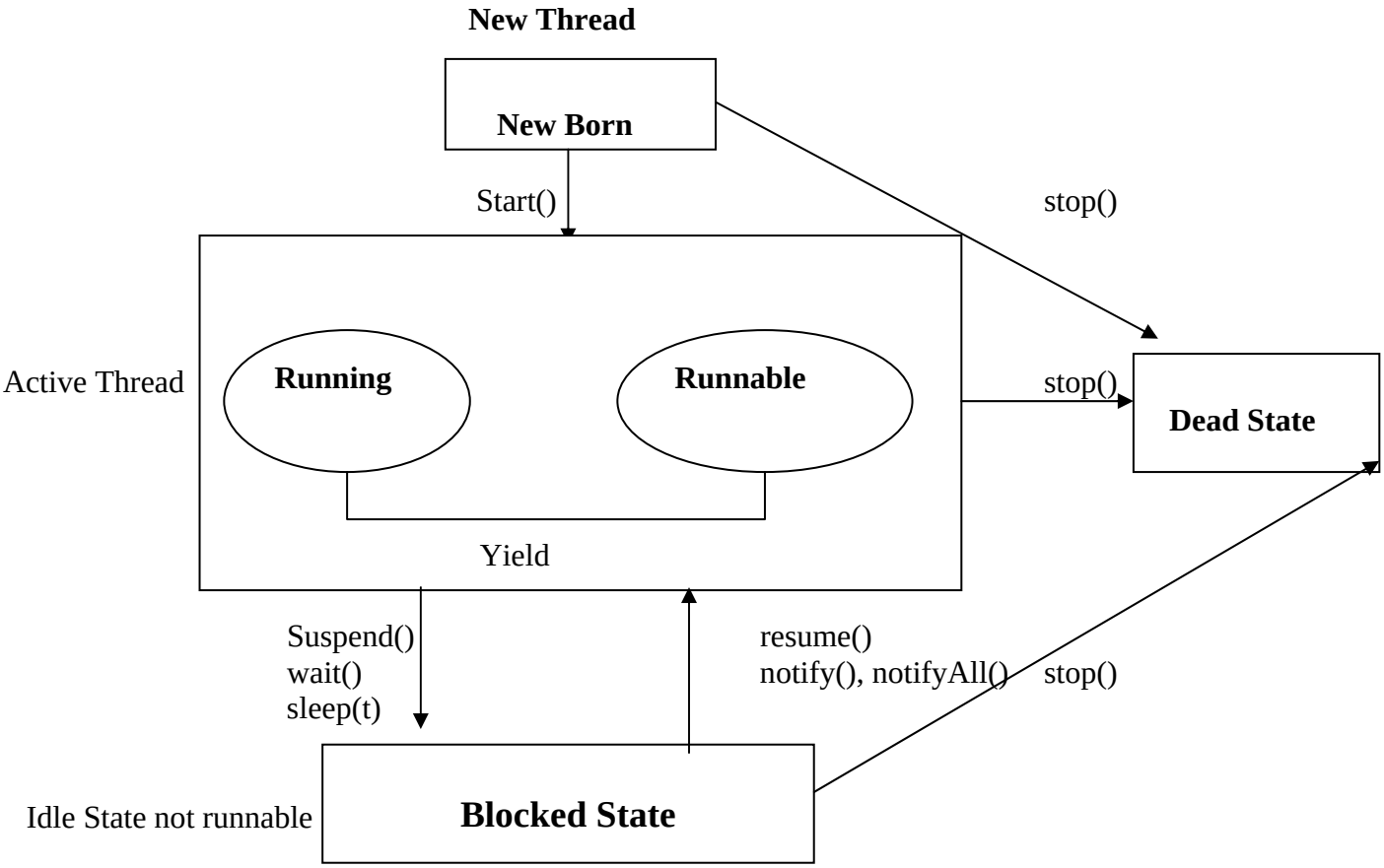


What is thread?

Any single path of execution is called thread. The path is defined by elements such as stack and set of registers to store temporary memory. Every program has at least one thread. Each thread has its own stack, priority and virtual set of registers. Threads subdivided the runtime behaviour of a programs into separate, independently running subtasks. Switching between threads is called out automatically by JVM. Even in a single processor system, thread provides an illusion of carrying out several tasks simultaneously.

Thread Life Cycle.

A thread is always in one of five states.



New Born State:- when we create a thread object, it is said to be in **newborn state.** At this stage we can start the thread using start() method or kill it using stop() method.

Runnable State:- The thread in runnable state means that the thread is ready for execution and is waiting for CPU time.

Running State:- Running state means CPU has given its time to the thread for execution.

Blocked State:- A thread is said to be in blocked state when it is suspended, sleeping or waiting in order to satisfy some condition. The suspend thread can be revived by using the resume() method. The thread which is sleeping can enter into runnable state as soon as the specified time period is completed. The thread in the wait() state can be schedule to run again by using the notify() method.

Dead State:- A thread can be killed in any state by using the stop() method.

The Thread class and Runnable interface

Thread programming has two parts. In the first part we create a thread class. In the second part we actually run the thread. A new thread can be created in the two different ways.

- Define a class that extends thread class and override its run() method.

The Thread class defines several methods. getName(), getPriority(), isAlive(), join(), run(), start(). etc.

- By converting a class to a thread by implementing the Runnable interface. The Runnable interface has only one method run().

Thread class methods

1. Thread.currentThead() => obtain the reference of the main thread.
2. setName(String name) => set the name of the thread.
3. getName() = > get the name of the thread.
4. getPriority() =>
5. setPriority(int n)
6. isAlive() => returns true if the thread is started but is not yet dead.
7. join() => Waits for the thread to terminate.
8. start() = > puts the new thread into a runnable state.
9. run() = > When a thread is started, the scheduler calls its run() method.
10. sleep() = > suspends a thread for a period of time.

Example # 1

```
// this is prg for which using the currentThread() and setName() method
// this prg is used single thread so here not need to extends Thread class or implement
//Runnable
class ThreadName {
    public static void main(String args[])      {
            Thread t=new Thread();
             t=Thread.currentThread();
            System.out.println("The current running thread is=::"+t);
            t.setName("MyThread");
           System.out.println("After changing thread name is=::"+t);
           System.out.println("only name of the thread is="+t.getName());
   }
}
```

Example # 2

```
// Using Thread methods yield(), stop() and sleep().
class A extends Thread {
        public void run() {
          for(int i=0;i<5;i++) {
                  if(i==1)
                          yield();
                  System.out.println("\t From Thread A :i="+i);
```

```
        }
        System.out.println("Exit from A");
    }
}

class B extends Thread {
        public void run() {
                for(int j=0;j<5;j++) {
                        System.out.println("\t From Thread B :j="+j);
                        if(j==3) stop();
                }
                System.out.println("Exit from B");
        }
}

class C extends Thread {
        public void run() {
                for(int k=0;k<5;k++) {
                        System.out.println("\t From Thread C :k="+k);
                        if(k==1)
                        try{
                                Thread.sleep(1000);
                        }
                catch(InterruptedException e) {}
                }
        System.out.println("Exit from C");
        }
}
class ThreadMethod {
        public static void main(String args[]) {
                A obj1=new A();
                B obj2=new B();
                C obj3=new C();
                System.out.println("Start of Thread A");
                obj1.start();
                System.out.println("Start of Thread B");
                obj2.start();
                System.out.println("Start of Thread B");
                obj3.start();
                System.out.println("End of main Thread");
        }
}
```

Output

```
C:\>java  ThreadMethod
Start of Thread A
```

```
Start of Thread B
Start of Thread C
End of main Thread
        From Thread A :i=0
        From Thread B :j=0
        From Thread B :j=1
        From Thread B :j=2
        From Thread B :j=3
        From Thread C :k=0
        From Thread C :k=1
        From Thread A :i=1
        From Thread A :i=2
        From Thread A :i=3
        From Thread A :i=4
Exit from A
        From Thread C :k=2
        From Thread C :k=3
        From Thread C :k=4
Exit from C
```

Example # 3

```
class newThread implements Runnable {
        String name;
        Thread t;
        newThread(String s) {
                name=s;
                t=new Thread(this,name);
                System.out.println("Thread ::"+t.getName()+"is start.");
                t.start();
        }
        public void run() {
                try {
                        for(int i=0;i<5;i++) {
                                System.out.println(name+"::"+i);
                        }
                        t.sleep(1000);
                }
                catch(Exception e) {
                        System.out.println(e);
                }
        }
}

class AliveJoinThread {
        public static void main(String args[]) {
                newThread obj1,obj2,obj3;
```

```
                obj1=new newThread("one");
                obj2=new newThread("two");
                obj3=new newThread("three");
                System.out.println("Thread one is live="+obj1.t.isAlive());
                System.out.println("Thread two is live="+obj2.t.isAlive());
                System.out.println("Thread three is live="+obj3.t.isAlive());
                try {
                        System.out.println("Wait for thread for finished");
                        obj1.t.join();obj2.t.join();obj3.t.join();
                }
                catch(Exception e){System.out.println(e);}
                System.out.println("Thread one is live="+obj1.t.isAlive());
                System.out.println("Thread two is live="+obj2.t.isAlive());
                System.out.println("Thread three is live="+obj3.t.isAlive());
        }
}
```

output

```
C:\>java  AliveJoinThread
Thread ::oneis start.
Thread ::twois start.
Thread ::threeis start.
Thread one is live=true
Thread two is live=true
one::0
two::0
three::0
one::1
two::1
three::1
one::2
two::2
three::2
one::3
two::3
three::3
one::4
two::4
three::4
Thread three is live=true
Wait for thread for finished
Thread one is live=false
Thread two is live=false
Thread three is live=false
```

Creating Threads

Creating a thread extends the thread class

Creation of thread is a two step process.

1. Create the new class:
a. Define a subclass of thread.
b. Override its run() method.
2. Instantiate and run the thread.
a. Create an instance of class.
b. Call its start() method.

Example # 2

```
class A extends Thread {
        public void run() {
                for(int i=0;i<5;i++) {
                        System.out.println("\t From Thread A :i="+i);
                }
                System.out.println("Exit from A");
        }
}
class B extends Thread {
        public void run() {
                for(int j=0;j<5;j++) {
                        System.out.println("\t From Thread B :j="+j);
                }
        System.out.println("Exit from B");
        }
}

class C extends Thread {
        public void run() {
                for(int k=0;k<5;k++) {
                        System.out.println("\t From Thread C :k="+k);
                }
        System.out.println("Exit from C");
        }
}
class ThreadTest {
        public static void main(String args[]) {
                A obj1=new A();
                B obj2=new B();
                C obj3=new C();
                obj1.start();
```

```
                obj2.start();
                obj3.start();
        }
}
```

output

```
C:\>javac ThreadTest.java
C:\>java  ThreadTest
        From Thread A :i=0
        From Thread A :i=1
        From Thread A :i=2
        From Thread A :i=3
        From Thread A :i=4
Exit from A
        From Thread B :j=0
        From Thread B :j=1
        From Thread B :j=2
        From Thread B :j=3
        From Thread B :j=4
Exit from B
        From Thread C :k=0
        From Thread C :k=1
        From Thread C :k=2
        From Thread C :k=3
        From Thread C :k=4
Exit from C
```

Creating a thread implementing the Runnable interface

To create a thread, declare a class that implement the runnable interface. To implement runnable, a class needs to only implements a single method called run(). After you create a class that implements runnable, you will instantiate an object of types thread using the constructor of thread class.

Thread(Object of Runnable interface);

Example # 3

```
class A extends Thread {
        public void run() {
                for(int i=0;i<5;i++) {
                        System.out.println("\t From Thread A :i="+i);
                }
                System.out.println("Exit from A");
        }
}
```

```
class RunnableTest {
        public static void main(String args[]) {
                A obj1=new A();
                Thread t=new Thread(obj1);
                t.start();
        }
}
```

Output

```
C:\>javac RunnableTest.java
C:\>java  RunnableTest
        From Thread A :i=0
        From Thread A :i=1
        From Thread A :i=2
        From Thread A :i=3
        From Thread A :i=4
Exit from A
```

Thread Priorities

In java each thread is assigned a priority. Thread priority are used by the thread scheduler is decide which and when thread should be allowed to run().

The threads of the same priority are gives equal treatment by the java scheduler and therefore they share the processor on a first come first serve (Round Robin) basis.

Java permits us to set the priority of a thread by using the setPriority( ) method.

Syntax:-

ThreadName.setPriority(int number);

number has a range between 1 to 10.

MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10

Higher priority thread gets more CPU time comparison to lower priority threads.

We can also capable to obtain the current setting of priority of thread by using the following method of thread class.

int getPriority( );

Example # 4

```
// this is example of using the thread priority methods.
class A extends Thread {
        public void run() {
                for(int i=0;i<5;i++) {
                        System.out.println("\t From Thread A :i="+i);
```

```
                }
                System.out.println("Exit from A");
        }
}
class B extends Thread {
        public void run() {
                for(int j=0;j<5;j++) {
                        System.out.println("\t From Thread B :j="+j);
                }
        System.out.println("Exit from B");
        }
}
class C extends Thread {
        public void run() {
                for(int k=0;k<5;k++) {
                        System.out.println("\t From Thread C :k="+k);
                }
        System.out.println("Exit from C");
        }
}
class ThreadPriority {
        public static void main(String args[]) {
                A objA=new A();
                B objB=new B();
                C objC=new C();
                objC.setPriority(Thread.MAX_PRIORITY);
                objB.setPriority(objA.getPriority()+1);
                objA.setPriority(Thread.MIN_PRIORITY);
                System.out.println("Start Thread A");
                objA.start();
                System.out.println("Start Thread B");
                objB.start();
                System.out.println("Start Thread C");
                objC.start();
                System.out.println("End of main Thread");
        }
}
```

output

```
C:\>java  ThreadPriority
Start Thread A
Start Thread B
        From Thread B :j=0
        From Thread B :j=1
        From Thread B :j=2
        From Thread B :j=3
```

```
        From Thread B :j=4
Exit from B
Start Thread C
        From Thread C :k=0
        From Thread C :k=1
        From Thread C :k=2
        From Thread C :k=3
        From Thread C :k=4
Exit from C
End of main Thread
        From Thread A :i=0
        From Thread A :i=1
        From Thread A :i=2
        From Thread A :i=3
        From Thread A :i=4
Exit from A
```

Synchronization

So far we have seen threads that use their own data and methods provided inside their **run()** methods. What happens when they try to use data and methods outside themselves? On such occasions, they may compete for the same resources and may lead to serious problems. For example one thread may try to read a record from a file while another is still writing to the same file. Depending on the situation, we may get strange results. Java enables us to overcome this problem using a technique known as *Synchronization.*

In case of java the keyword **synchronized** helps to solve such problems by keeping a watch on such locations. For example, the method that will read information from a file and the method that will update the same file may be declared as **synchronized.**

Syntax:

```
synchronized void update {
        ……..……..…..
        ……..……..…..…        //code here is synchronized
        ……..……..
}
```

Concept of monitor

When we declare a method synchronized, java creates a **"monitor"** and hands it over to the thread that calls the method first time. As long as the thread holds the monitor, no other thread can enter the synchronized section of the code. A monitor is like a key and the thread that holds the key can only open the lock.

Whenever a thread has completed its work of using synchronized method, it will hand over the **monitor** to the next thread that is ready to use the same resource.

An interesting situation may occur when two or more threads are waiting to gain control of a resource. Due to some reasons, the condition on which the waiting threads rely on the gain control does not happen. This results in what is known as ***deadlock.***

Example of Synchronization:

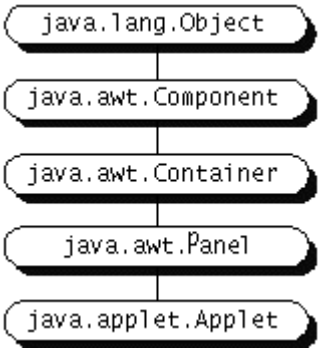
```
class A extends Thread {
        synchronized public void run() {
                for(int i=0;i<5;i++) {
                        System.out.println("\t From Thread A :i="+i);
                }
                System.out.println("Exit from A");
        }
}
class B extends Thread {
        synchronized public void run() {
                for(int j=0;j<5;j++) {
                        System.out.println("\t From Thread B :j="+j);
                }
                System.out.println("Exit from B");
        }
}
class C extends Thread {
        synchronized public void run() {
                for(int k=0;k<5;k++) {
                        System.out.println("\t From Thread C :k="+k);
                }
        System.out.println("Exit from C");
        }
}

class SynchronizedTest {
        public static void main(String args[]) {
                A objA=new A();
                B objB=new B();
                C objC=new C();
                System.out.println("Start Thread A");
                objA.start();
                System.out.println("Start Thread B");
                objB.start();
                System.out.println("Start Thread C");
                objC.start();
                System.out.println("End of main Thread");
        }
}
```

Overview of Applets

This lesson discusses the parts of an applet. If you haven't yet compiled an applet and included it in an HTML page, you might want to do so now. Step by step instructions are in Writing Applets.

Every applet is implemented by creating a subclass of the `Applet` class. The following figure shows the inheritance hierarchy of the `Applet` class. This hierarchy determines much of what an applet can do and how, as you'll see on the next few pages.



A Simple Applet

Below is the source code for an applet called `Simple`. The `Simple` applet displays a descriptive string whenever it encounters a major milestone in its life, such as when the user first visits the page the applet's on. The pages that follow use the `Simple` applet and build upon it to illustrate concepts that are common to many applets.

```
/*
 * 1.0 code.
 */

import java.applet.Applet;
import java.awt.Graphics;

public class Simple extends Applet {

    StringBuffer buffer;

    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }
```

```
    public void destroy() {
        addItem("preparing for unloading...");
    }

    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0, size().width - 1, size().height - 1);

        //Draw the current string inside the rectangle.
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

The Life Cycle of an Applet

You can use the `Simple` applet to learn about the milestones in every applet's life.

Methods for Milestones

The `Applet` class provides a framework for applet execution, defining methods that the system calls when milestones -- major events in an applet's life cycle -- occur. Most applets override some or all of these methods to respond appropriately to milestones.

Methods for Drawing and Event Handling

Applets inherit the drawing and event handling methods of the AWT `Component` class. (AWT stands for Abstract Windowing Toolkit; applets and applications use its classes to produce user interfaces.) *Drawing* refers to anything related to representing an applet on-screen -- drawing images, presenting user interface components such as buttons, or using graphics primitives. *Event handling* refers to detecting and processing user input such as mouse clicks and key presses, as well as more abstract events such as saving files and iconifying windows.

Methods for Adding UI Components

Applets inherit from the AWT `Container` class. This means that they are designed to hold `Components` -- user interface objects such as buttons, labels, pop-up lists, and scrollbars. Like other `Containers`, applets use layout managers to control the positioning of `Components`.

What Applets Can and Can't Do

For security reasons, applets that are loaded over the network have several restrictions. One is that an applet can't ordinarily read or write files on the computer that it's executing on. Another

is that an applet can't make network connections except to the host that it came from. Despite these restrictions, applets can do some things that you might not expect. For example, applets can invoke the public methods of other applets on the same page.

Test Driving an Applet

Once you've written an applet, you'll need to add it to an HTML page so that you can try it out. This section tells you how to use the `<APPLET>` HTML tag to add an applet to an HTML page.

The Life Cycle of an Applet

Loading the Applet

- An instance of the applet's controlling class (an `Applet` subclass) is created.
- The applet *initializes* itself.
- The applet *starts* running.

Leaving and Returning to the Applet's Page

- When the user leaves the page -- for example, to go to another page -- the applet has the option of *stopping* itself. When the user returns to the page, the applet can *start* itself again. The same sequence occurs when the user iconifies and then reopens the window that contains the applet. (Other terms used instead of *iconify* are *minaturize*, *minimize*, and *close*.)

Reloading the Applet

- Some browsers let the user reload applets, which consists of unloading the applet and then loading it again. Before an applet is unloaded, it's given the chance to *stop* itself and then to perform a *final cleanup*, so that the applet can release any resources it holds. After that, the applet is unloaded and then loaded again, as described in **Loading the Applet**, above.

Quitting the Browser

- When the user quits the browser (or whatever application is displaying the applet), the applet has the chance to *stop* itself and do *final cleanup* before the browser exits.

An applet can react to major events in the following ways:

- It can *initialize* itself.
- It can *start* running.
- It can *stop* running.
- It can perform a *final cleanup*, in preparation for being unloaded.

Methods

```
public class Simple extends Applet {
    . . .
    public void init() { . . . }
    public void start() { . . . }
    public void stop() { . . . }
    public void destroy() { . . . }
    . . .
}
```

The `Simple` applet, like every other applet, features a subclass of the `Applet` class. The `Simple` class overrides four `Applet` methods so that it can respond to major events:

`init`
: To *initialize* the applet each time it's loaded (or reloaded).

`start`
: To *start* the applet's execution, such as when the applet's loaded or when the user revisits a page that contains the applet.

`stop`
: To *stop* the applet's execution, such as when the user leaves the applet's page or quits the browser.

`destroy`
: To perform a *final cleanup* in preparation for unloading.

The `init` method is useful for one-time initialization that doesn't take very long. In general, the `init` method should contain the code that you would normally put into a constructor. The reason applets shouldn't usually have constructors is that an applet isn't guaranteed to have a full environment until its `init` method is called. For example, the `Applet` image loading methods simply don't work inside of a applet constructor. The `init` method, on the other hand, is a great place to call the image loading methods, since the methods return quickly.

Overview of Swing

What Are the JFC and Swing?

JFC is short for Java™ Foundation Classes, which encompass a group of features to help people build graphical user interfaces (GUIs). The JFC was first announced at the 1997 JavaOne developer conference and is defined as containing the following features:

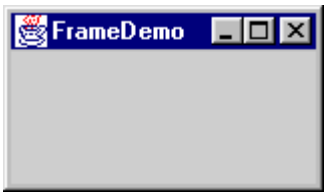
The Swing Components

Include everything from buttons to split panes to tables.

- ***Top-Level Containers***

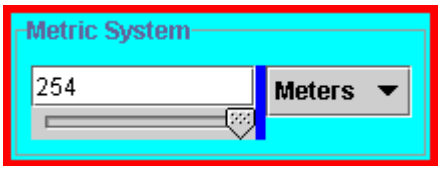
Applet

Dialog



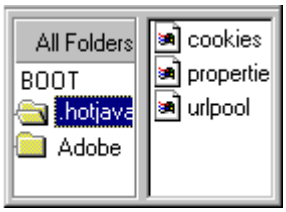
Frame

- ***General-Purpose Containers***

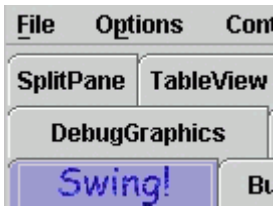


Panel

Scroll pane



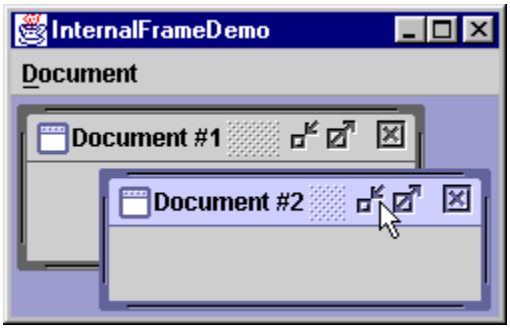
Split pane



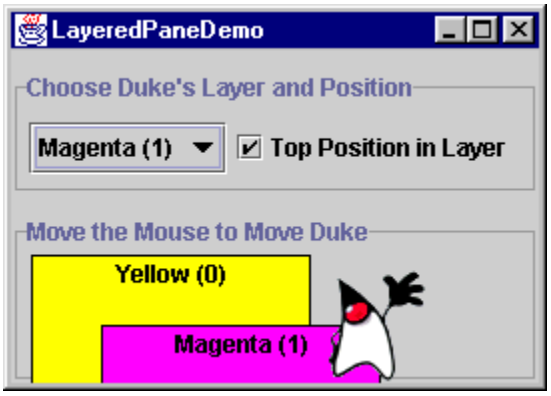
Tabbed pane

Tool bar

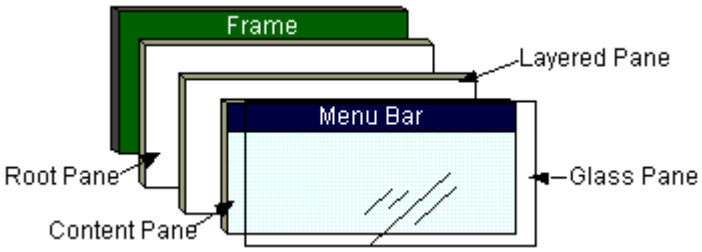
- ***Special-Purpose Containers***



Internal frame

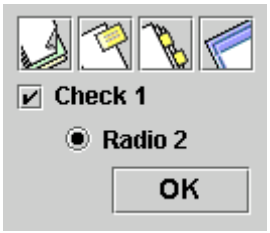


Layered pane

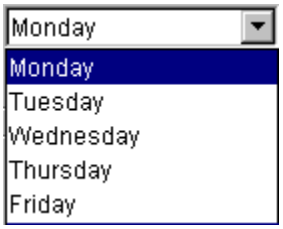


Root pane

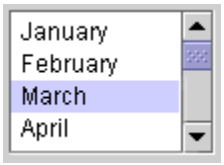
- ***Basic Controls***



Buttons

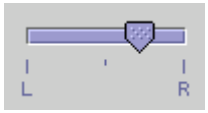


Combo box



List

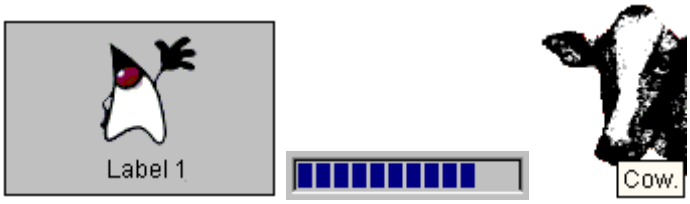
Menu



Slider

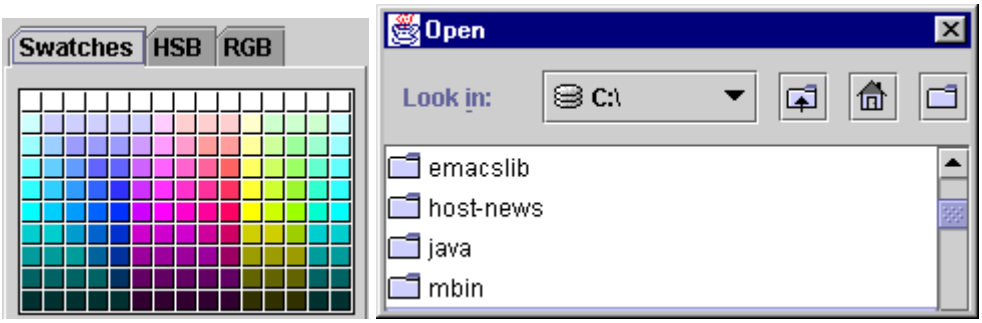
Text fields

- ***Uneditable Information Displays***



Label Progress bar Tool tip

- ***Editable Displays of Formatted Information***



Color chooser File chooser

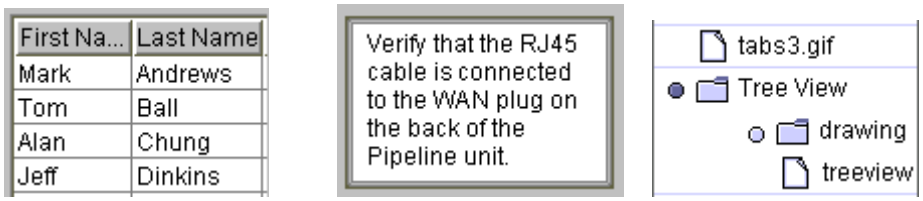


Table Text Tree

Pluggable Look and Feel Support

Gives any program that uses Swing components a choice of looks and feels. For example, the same program can use either the Java™ look and feel or the Windows look and feel. We expect many more look-and-feel packages -- including some that use sound instead of a visual "look" -- to become available from various sources.

Accessibility API

Enables assistive technologies such as screen readers and Braille displays to get information from the user interface.

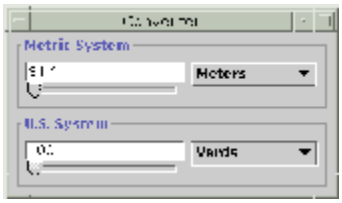
Java 2D™ API (Java 2 Platform only)

Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and in applets.

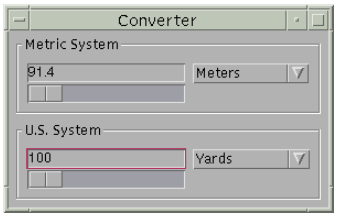
Drag and Drop Support (Java 2 Platform only)

Provides the ability to drag and drop between a Java application and a native application. The first three JFC features were implemented without any native code, relying only on the API defined in JDK 1.1. As a result, they could and did become available as an extension to JDK 1.1. This extension was released as JFC 1.1, which is sometimes called "the Swing release." The API in JFC 1.1 is often called "the Swing API." This trail concentrates on the Swing components. We help you choose the appropriate ones for your GUI, tell you how to use them, and give you the background information you need to use them effectively. We discuss the Pluggable look and feel and Accessibility support when they affect how you write programs that use Swing components. This trail does not cover the JFC features that appear only in the Java 2 Platform. The following snapshots show three views of a GUI that uses Swing components. Each picture shows the same program, but with a different look and feel.

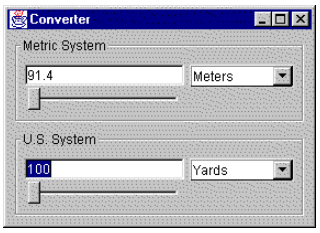
Java look and feel



CDE/Motif look and feel



Windows look and feel



Which Releases Contain the Swing API?

The Swing API is available in two forms:

- As a core part of the Java 2 Platform (standard edition of either v 1.2 or v 1.3)
- JFC 1.1 (for use with JDK 1.1)

Which release you use depends on whether you need to use JDK 1.1 or the Java 2 Platform, and on whether you're willing to be a beta tester for SDK v 1.3. It's a bit simpler to use the Java 2 Platform because the JFC is built into the Java 2 Platform and you don't need to add libraries to be able to use the Swing API. However, if you need to use JDK 1.1, then adding the Swing API (using JFC 1.1) isn't difficult. This trail describes the Swing 1.1 API, which is the version present in the Java 2 Platform v 1.2 and in the release called "JFC 1.1 (with Swing 1.1)." Except where noted, the code in this trail works unchanged with either release and subsequent compatible releases, such as SDK v 1.3 and JFC 1.1 (with Swing 1.1.1).

Sun has released many versions of JFC 1.1, which are identified by the version of Swing API they contain. One previous version, for example, was called "JFC 1.1 (with Swing 1.0.3)." The last JFC 1.1 release was Swing version 1.1.1. It had the same API as Swing 1.1, but added many bug fixes, some performance improvements, and a few new capabilities such as HTML text in labels that required no API changes. The following table shows some of the important releases containing Swing API. Bold font indicates the releases typically used in shipping products.

Swing API Version	**Corresponding JFC 1.1 Release**	**Corresponding Java 2 Platform Version (Standard Edition)**	**Comments**
Swing 1.0.3	**JFC 1.1 (with Swing 1.0.3)**	*none*	The release of JFC 1.1 included in **Java Plug-in™ 1.1.1**.
Swing 1.1 ***Note:*** *This is the API this trail covers.*	**JFC 1.1 (with Swing 1.1)**	**v 1.2**, **v 1.2.1**	The first releases containing the final Swing 1.1 API supported for use in shipping products. **Java Plug-in 1.1.2** and **Java Plug-in 1.2** provide applet support for JDK 1.1 + Swing 1.1 and Java 2 Platform v 1.2, respectively.
Swing 1.1.1 ***Note:*** *This trail includes notes about this API.*	**JFC 1.1 (with Swing 1.1.1)** ***Note:*** *This is the last release that supports JDK 1.1.*	**v 1.2.2**	Adds performance enhancements, many bug fixes, and selected new functionality (requiring no API changes) to Swing 1.1. **Java Plug-in 1.1.3** and **Java Plug-in 1.2.2** provide applet support for JDK 1.1 + Swing 1.1.1 and Java 2 Platform v 1.2.2, respectively.
no separate "Swing" version number	*none*	v 1.3 Beta	Adds significant performance enhancements and bug fixes, along with some new features and API additions. For more information, see the release documentation, especially Swing Changes and New Features. Java Plug-in 1.3 Beta provides applet support for this release.

What Swing Packages Should I Use?

The Swing API is powerful, flexible -- and immense. For example, the 1.1 version of the API has 15 public packages:
`javax.accessibility`, `javax.swing`, `javax.swing.border`, `javax.swing.colorchooser`, `javax.swing.event`, `javax.swing.filechooser`, `javax.swing.plaf`, `javax.swing.plaf.basic`, `javax.swing.plaf.metal`, `javax.swing.plaf.multi`, `javax.swing.table`, `javax.swing.text`, `javax.swing.text.html`, `javax.swing.tree`, and `javax.swing.undo`.

Fortunately, most programs use only a small subset of the API. This trail sorts out the API for you, giving you examples of common code and pointing you to methods and classes you're likely to need. Most of the code in this trail uses only one or two Swing packages:

- `javax.swing`
- `javax.swing.event` (not always required)

The AWT components are those provided by the JDK 1.0 and 1.1 platforms. Although the Java 2 Platform still supports the AWT components, we strongly encourage you to use Swing components instead. You can identify Swing components because their names start with `J`. The AWT button class, for example, is named `Button`, while the Swing button class is named `JButton`. Additionally, the AWT components are in the `java.awt` package, while the Swing components are in the `javax.swing` package.

The biggest difference between the AWT components and Swing components is that the Swing components are implemented with absolutely no native code. Since Swing components aren't restricted to the least common denominator -- the features that are present on every platform -- they can have more functionality than AWT components. Because the Swing components have no native code, they can be be shipped as an add-on to JDK 1.1, in addition to being part of the Java 2 Platform.

Even the simplest Swing components have capabilities far beyond what the AWT components offer:

- Swing buttons and labels can display images instead of, or in addition to, text.
- You can easily add or change the borders drawn around most Swing components. For example, it's easy to put a box around the outside of a container or label.
- You can easily change the behavior or appearance of a Swing component by either invoking methods on it or creating a subclass of it.
- Swing components don't have to be rectangular. Buttons, for example, can be round.
- Assistive technologies such as screen readers can easily get information from Swing components. For example, a tool can easily get the text that's displayed on a button or label.

Swing lets you specify which look and feel your program's GUI uses. By contrast, AWT components always have the look and feel of the native platform. Another interesting feature

is that Swing components with state use models to keep the state. A `JSlider`, for instance, uses a `BoundedRangeModel` object to hold its current value and range of legal values. Models are set up Automatically, so you don't have to deal with them unless you want to take advantage of the power they can give you. If you're used to using AWT components, you need to be aware of a few things when using Swing components:

- Programs should not, as a rule, use "heavyweight" components alongside Swing components. Heavyweight components include all the ready-to-use AWT components (such as `Menu` and `ScrollPane`) and all components that inherit from the AWT `Canvas` and `Panel` classes. This restriction exists because when Swing components (and all other "lightweight" components) overlap with heavyweight components, the heavyweight component is always painted on top.
- Swing components aren't thread safe. If you modify a visible Swing component -- invoking its `setText` method, for example -- from anywhere but an event handler, then you need to take special steps to make the modification execute on the event-dispatching thread. This isn't an issue for many Swing programs, since component-modifying code is typically in event handlers.
- The containment hierarchy for any window or applet that contains Swing components must have a Swing top-level container at the root of the hierarchy. For example, a main window should be implemented as a `JFrame` instance rather than as a `Frame` instance.
- You don't add components directly to a top-level container such as a `JFrame`. Instead, you add components to a container (called the *content pane*) that is itself contained by the `JFrame`.

Compiling and Running Swing Programs

This section tells you how to compile and run a Swing application. The compilation instructions work for all Swing programs -- applets, as well as applications. How you compile and run Swing programs depends on whether you're using JDK 1.1 or the Java 2 Platform. Using the Java 2 Platform is a bit simpler because Swing is built into it. Choose the instructions corresponding to the release you're using:

- **Java 2 Platform, v 1.2 or 1.3**
- **JDK 1.1 + JFC/Swing Release**

Both instructions tell you how to run a simple program, called SwingApplication, whose GUI looks like this:

Compiling and Running Swing Programs (Java 2 Platform)

Here are the steps for compiling and running your first Swing program with the Java 2 SDK, v 1.2 or 1.3:

1. Download the latest release of the Java 2 Platform, if you haven't already done so.
2. Create a program that uses Swing components.
3. Compile the program.
4. Run the program.

Download the Latest Release of the Java 2 Platform

Two versions of the Java 2 Platform are available, both for free. The first is the `Java 2 SDK, Standard Edition, v 1.2`, which is sometimes called "v 1.2" for short (or, incorrectly, "JDK 1.2"). The second, which is currently available only in pre-release form, is `Java 2 SDK, Standard Edition, v 1.3`, which is sometimes called simply "v 1.3".

Which version you choose depends on the features you need and your (or your company's) tolerance for not-quite-final software. For information about the differences between 1.2 and 1.3, see the `Java 2 SDK v 1.3 documentation`.

Create a Program That Uses Swing Components

You can use a simple program we provide, called `SwingApplication`. Please download and save this file: `SwingApplication.java`. The spelling and capitalization of the file's name must be exactly "SwingApplication.java".

Compile a Program That Uses Swing Components

Your next step is to compile the program. Compiling a Swing program with one of the Java 2 SDKs is simple, since the Swing packages are part of the Standard Edition of the Java 2 Platform. Here is an example:

```
javac -deprecation SwingApplication.java
```

If you can't compile `SwingApplication.java`, it's probably either because you're using a JDK 1.1 compiler instead of a v 1.2 or 1.3 compiler, or because you're using a beta release of v 1.2. Once you update a more recent release of the Java 2 Platform, you should be able to use the programs in this trail without change.

Run the Program

After you compile the program successfully, you can run it. This section tells you how to run an application.

Assuming that your program uses a standard look and feel -- such as the Java look and feel, Windows look and feel, or CDE/Motif look and feel -- you can use the v 1.2 or 1.3 interpreter to run the program without adding anything to your class path. For example:

```
java SwingApplication
```

Event Handling in Java

GUI classes

- Components that you can use to build a GUI app, are instances of classes contained in the javax.swing package:
    - JButton
    - JTextField
    - JRadioButton
    - JCheckBox
    - JComboBox
    - JLabel etc…

Layout Managers

You can use the layout managers to design your GUIs. There are several managers like:

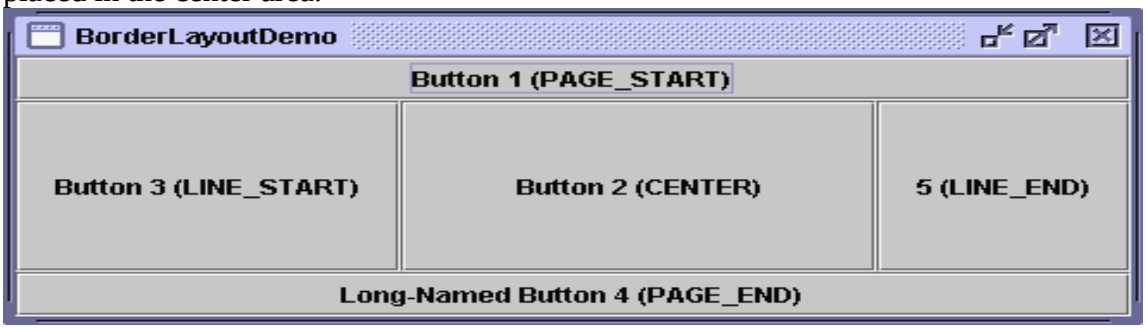
- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

FlowLayout

It is the Default layout of GUI application. Components laid out from the top-left corner, from left to right and top to bottom like a text.

BorderLayout

Places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area.

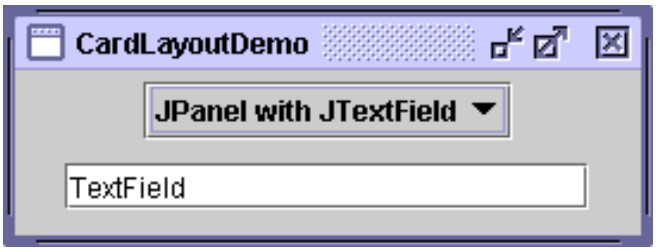


GridLayout

Simply makes a bunch of components equal in size and displays them in the requested number of rows and columns

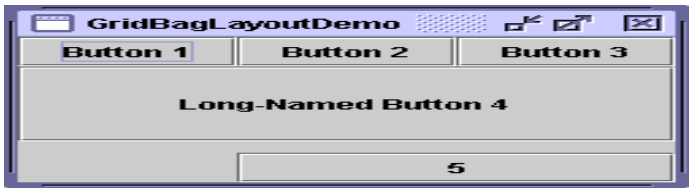
CardLayout

We can implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays.



GridBagLayout

It is a sophisticated, flexible layout manager. It aligns components by placing them within a grid of cells, allowing some components to span more than one cell.



To use the layout managers in the GUI Application we must use it into code:

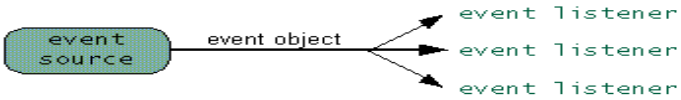
Container myCont = getContentPane();
myCont.setLayout(new FlowLayout());

The *delegation model*

- Sources: The mouse and keyboard and the GUI components (Buttons, lists, checkboxes etc.) The Source generates an event and sends it to the registered listener.
- Events: Objects that describe a state change in a source.
- Listeners: Objects notified when events occur.

Listeners

Any number of event listener objects can listen for all kinds of events from any number of event source objects. E.g. a program might create one listener per event source or a program might have a single listener for all events from all sources. Multiple listeners can register to be notified of events of a particular type from a particular source. Also, the same listener can listen to notifications from different objects.



Each type of listeners can be notified only for its corresponding types of events which can be generated by specific types of sources.

Multiple sources, single listener - Many buttons can register the same listener since all buttons generate the same type of event. This type of event may be generated by other types of sources as well.

Listeners as interfaces – We can implement an interface to create a listener. In the case of a single source that generates multiple types of events you can create a single listener that implements all interfaces (remember: a class may extend only one superclass but implement more than one interface).

Sources-events-listeners

Source	Event <state change>	Listener	Methods (argument: corresponding event)
Mouse	MouseEvent <mouse clicked, pressed, dragged, moved/ ntered, exited a component etc> MouseWheelEvent <page-up and down>	MouseListener	mouseClicked mousePressed mouseReleased etc
		MouseMotionListener	mouseDragged mouseMoved
		MouseWheelListener	mouseWheelMoved
Keyboard	KeyEvent	KeyListener	keyPressed keyReleased keyTyped
Button	ActionEvent <GUI button clicked>	ActionListener	ActionPerformed
List	ActionEvent <item double clicked>	ActionListener	ActionPerformed
	ItemEvent <item selected/deselected>	ItemListener	ItemStateChanged

Event handling steps

To handle an event you need 3 steps:

- Implement the appropriate interface that will produce your listener class.
- Create a listener object.
- Register the listener to the source of interest.

We must register a listener using the corresponding add function in the form: *component*.add*Some*Listener(*listener_object*);
You can find the source of an event by using the getSource method:
event_object.getSource();

Putting everything together

So we need two classes: The one that extends JFrame (see part one) which will create your GUI and The one(s) that will implement your listener(s) or one that does them both!

Inner classes
You may create all different classes independently and put them in separate files, or you can implement your listeners inside the class that extends your JFrame, making it an *inner class*. This enables you to put everything in one class (and hence file).

Adapter classes
Adapter classes are fully abstract classes that correspond to listener interfaces. They are extended (not implemented) and thus you can ignore methods that do not interest you. You can use them instead of listeners in the case that only some of the methods of the interface are to be used. You don't use the abstract specifier and of course you cannot use multiple inheritence.
Some adapter classes:

- KeyAdapter (instead of KeyListener)
- MouseAdapter (MouseListener)
- MouseMotionAdapter (MouseMotionListener) etc…

Create your window

```
//Extend a JFrame:
public class MyApp extends JFrame {
        //Declare all your components private:
        private JButton b1;
        //Create your constructor:
        Public MyApp(){
                super("SwingApplication");
                //Inside your constructor:
                //Get your container:
                Container myCont = getContentPane();
                //Inside your constructor, set a layout:
                GridBagLayout layout = new GridBagLayout();
                myCont.setLayout(layout);
                // Inside your constructor, create your element objects:
                b1=new JButton("I'm a Swing button!"); ...
                // Inside your constructor, Add them on your content pane:
                myCont.add(b1); ...
                // Inside your application class, add the main method that is the entry point of
                your //program and creates the application object:
        }
        public static void main(String[] args){
                MyApp a =new MyApp();
                a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Event handling in practice

When an event is generated by a source it is handled by the listener registered by this source. As shown, the listener is an implemented interface (or an extended adapter) and thus you have to implement some methods. These methods take the event as an argument and this is where you put the code to be executed when the event happens. In other words, you define what you want to happen eg. when a button is pressed, inside the actionPerformed method implemented inside the listener . To be able to do that efficiently, the event classes define specific methods to extract information about the events.

- EventObject:

  // superclass of all events, so the following method is inherited by all event objects
  - Object getSource()

- MouseEvent:
  - int getX(), int getY()
  - int getButton()

- KeyEvent:
  - char getKeyChar()

- ActionEvent:
  - String getActionCommand()
  - long getWhen()